PORTOFOLIU DE LUCRĂRI

CANDIDAT Conf. dr. Alexandra Băicoianu

Doctor în Informatică

Titlul tezei de doctorat *Some Other Issues in Discrete Optimization*, Facultatea de Matematică și Informatică, Universitatea Babeș-Bolyai din Cluj-Napoca, ordin 3148/30.01.2017

Titlul tezei de abilitare *Applied Intelligence - Machine Learning Across Interdisciplinary Technologies and Systems*

Portofoliul de lucrări relevante din domeniul de doctorat **INFORMATICĂ**

1. **Multi-Texture Synthesis through Signal Responsive Neural Cellular Automata**
   Scientific Reports
   În recenzie, etapa II, minor revision
   Also, on arXiv https://arxiv.org/abs/2407.05991
2. **Fractal interpolation in the context of prediction accuracy optimization**
   Engineering Applications of Artificial Intelligence, 2024,
   10.14569/IJACSA.2019.0100514
   https://www.sciencedirect.com/science/article/pii/S0952197624005384
3. **Multisource Remote Sensing Data Visualization using Machine Learning**
   IEEE Transactions on Geoscience and Remote Sensing, 2024, Volume: 62, On Page(s): 1-12, Print ISSN: 0196-2892, DOI 10.1109/TGRS.2024.3372639
   https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10458686
4. **Medical Emergency Department Triage Data Processing using a Machine-Learning Solution**
   Heliyon, 2023, e18402, ISSN 2405-8440, https://doi.org/10.1016/j.heliyon.2023.e18402
   https://www.sciencedirect.com/science/article/pii/S2405844023056104
5. **A Concretization of an Approximation Method for Non-Affine Fractal Interpolation Functions**
   Mathematics, https://doi.org/10.3390/math9070767
   https://www.mdpi.com/2227-7390/9/7/767
6. **Learning about Growing Neural Cellular Automata**
   IEEE Access, 2024, DOI: 10.1109/ACCESS.2024.3382541
   https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10480399
7. **Towards Generating Executable Metamorphic Relations Using Large Language Models**
   QUATIC Conference 2024, Preprint version https://arxiv.org/abs/2401.17019
   https://link.springer.com/chapter/10.1007/978-3-031-70245-7_9
8. **DACIA5: a Sentinel-1 and Sentinel-2 dataset for agricultural crop identification applications**
   Big Earth Data, https://doi.org/10.1080/20964471.2025.2512685
   https://www.tandfonline.com/doi/full/10.1080/20964471.2025.2512685?scroll=top&needAccess=true#d1e525

9. **An Extended Survey Concerning the Significance of Artificial Intelligence and Machine Learning Techniques for Bug Triage and Management**

   IEEE Access, Print/Online ISSN: 2169-3536, Online ISSN: 2169-3536, DOI: 10.1109/ACCESS.2023.3329732

   https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10305170

10. **AI-Based Visualization of Remotely-Sensed Spectral Images**

    2023 International Symposium on Signals, Circuits and Systems (ISSCS), Iasi, Romania, 2023, pp. 1-4, 10.1109/ISSCS58449.2023.10190908

    https://ieeexplore.ieee.org/document/10190908

3.09.2025

Multi-Texture Synthesis through Signal Responsive Neural Cellular Automata

Mirela-Magdalena CATRINA*,
Ioana Cristina PLAJER†, Alexandra BĂICOIANU‡

Abstract

Neural Cellular Automata (NCA) have proven to be effective in a variety of fields, with numerous biologically inspired applications. One of the fields, in which NCAs perform well is the generation of textures, modelling global patterns from local interactions governed by uniform and coherent rules. This paper aims to enhance the usability of NCAs in texture synthesis by addressing a shortcoming of current NCA architectures for texture generation, which requires separately trained NCA for each individual texture. In this work, we train a single NCA for the evolution of multiple textures, based on individual examples. Our solution provides texture information in the state of each cell, in the form of an internally coded genomic signal, which enables the NCA to generate the expected texture. Such a neural cellular automaton not only maintains its regenerative capability but also allows for interpolation between learned textures and supports grafting techniques. This demonstrates the ability to edit generated textures and the potential for them to merge and coexist within the same automaton. We also address questions related to the influence of the genomic information and the cost function on the evolution of the NCA.

1 Introduction

Texturing is a demanding, complex, and fundamental process in computer graphics [1], referring to the process of mapping a texture, usually provided by an image file, onto a given object. The technique of producing high-quality textures of custom size, which are similar to the given examples without containing artifacts or unnatural repetition is called texture synthesis. The importance

*Faculty of Mathematics and Informatics, *Transilvania* University of Braşov, Romania, e-mail: mirela.catrina@student.unitbv.ro

†Department of Mathematics and Informatics, *Transilvania* University of Braşov, Romania, e-mail: ioana.plajer@unitbv.ro

‡Department of Mathematics and Informatics, *Transilvania* University of Braşov, Romania, e-mail: a.baicoianu@unitbv.ro

of this technique is rapidly increasing due to its extensive use in industries like game and film production, combined with the labor-intensive process of manual texture creation.

Texture synthesis by procedural generation offers multiple advantages, such as efficient sampling and use of time and memory. A few lines of shader code can create complex, satisfactory patterns and improve the system efficiency compared to sampling from 2D image textures, generated using example-based methods [1, 2, 3]. These parametric methods, varying in diversity and size, provide minimal errors and are highly scalable to any resolution output. It is not unexpected that neural networks have been shown to be a powerful approach to texture synthesis [4, 5, 6].

Recent work demonstrates the capabilities of neural cellular automata (NCA) for this task and highlights the advantages of using this approach, namely, efficient sampling, compactness of the underlying representations and easy usage through short shader programs [2, 3, 7, 8]. The NCA is employed as a differentiable image parametrization [3, 9] meaning it transforms a given image, which is initially, a gray uniform image, in the style of the given target or example. It is important to underline, that the automaton does not create a pixel-copy of the target image, but it generates an example-based texture. The NCA works as a generator and uses the gradients provided by a pre-trained differentiable model, in order to learn the style of the target. This means that each trained NCA generates one single targeted texture and, consequently, such an approach for large-scale multiple texture generation may become burdensome.

Nevertheless, after multiple experiments and research, we concluded that NCAs could perfectly fit the task of generating textures, given the fact that they model global patterns from local interactions governed by uniform, consistent rules, enhancing structured, near-regular patterns in a compact manner. Therefore, by this study we aim to enhance the usability of NCAs in texture synthesis by training a single neural cellular automaton to evolve into multiple textures, thus increasing the automaton's generalisation capacity. Our solution relies on providing texture information into the seed, as a genomically-coded internal signal that the NCA will interpret and thus yield the expected texture. Particularly, we define a few hidden channels of the cell's state to be genome channels, and use binary encoding for each example image index. As a result our NCA will be able to develop 2, 4, 8 (or any other power of two) number of textures. We detail the process of selecting an optimal baseline for our architecture in Section 2.1 and cover the implementation details of the proposed approach in Section 2.2. The results, consisting of an NCA that evolves up to 8 textures, are showcased in Section 3. Moreover, considering the increased complexity and extended range of behaviours exhibited by the NCA, we explore interpolation behaviour and grafting techniques, further emphasizing the editability of our generated textures and the possibility of multiple textures joining and coexisting in one single automaton. In Section 3, we also address questions relating to the extent to which our NCA uses the genomic information, and proper loss function selection based on the properties of selected example images.

2 Methodology

In this section we introduce the methodology and the theoretical aspects important to our scope. Firstly we introduce the architecture of an NCA for single texture generation and we describe the way it evolves in a given texture, specifically focusing on the loss function and cell perception methodologies.

In the following, we explain the NCA model for multiple texture generation, which is the main scope of our work. The experiments follow the proposed genomic coding in the seed for the development of multiple textures. We explain the reasoning for choosing this type of encoding, as well as the architecture and the training details. Furthermore, we briefly discuss the new behaviour of texture interpolation specific to our NCA and its potential.

Finally, in this section, we introduce further capabilities of the proposed architecture, like regeneration and grafting. Clear and concise steps are presented for these experiments and the results are detailed in Section 3.

2.1 Single texture generation NCA

Starting from the architecture discussed in [3] this study aims to determine an optimal baseline for further expanding the NCA for multi-texture generation. Our experiments included choosing optimal perception kernels, an adequate neural network architecture and an appropriate loss function.

2.1.1 Architecture and Inference

A neural cellular automaton is a model that encompasses cells displayed in a structured manner. Different cellular automaton types were developed in literature, each designed for a specific use-case [10, 11, 12, 13, 14].

For the texture synthesis task, alive cells are displayed in a grid and each one corresponds to a pixel in the generated texture. The NCA is initialized with a seed state, s_0, and is guided through time using an update function applied for each cell. At each timestamp, cell states are modified, and thus the expected texture emerges after a few evolution stages. The update rule is learnt by a neural network and outputs the new state of a cell based on its previous state and those of its neighbours.

Each cell's information is stored in an n_s-dimensional state vector, where n_s is a hyperparameter of the model. The state vector for each of the cells comprises two components:

- **3 color channels**: the first 3 values of the state vector correlate with the RGB channels, assigning color to the corresponding pixel
- n_h **hidden channels**: the following values or channels, designed to facilitate cell communication ($n_h = n_s - 3$). orangeIn our experiments, we preponderantly used $n_h \in \{9, 10, 12\}$ hidden channels, specific per-experiment values are detailed in Section 3.

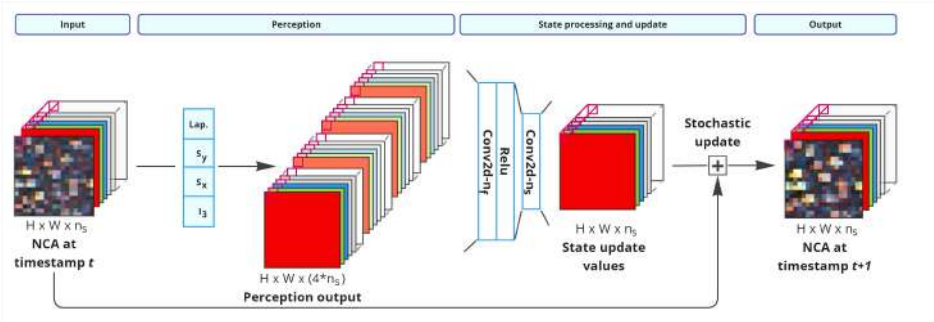


Figure 1: One NCA pass.

For single texture generation we use a NCA based on the architecture proposed in [3]. A NCA pass, applied at the transition from time step t to $t+1$, is illustrated in Figure 1 and consists of two major steps: perception, and state processing and update. The perception stage employs 4 fixed kernels: I_3 - 3×3 identity kernel, S_x - $Sobel_x$, S_y - $Sobel_y$, Lap - $Laplacian$ to convolve with each channel of the NCA state and offers the perception values for the next stage. We used the 9-point variant of the discrete Laplace operator, as recommended in [3]. The kernel values for S_x, S_y, Lap are displayed in Equation 1.

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}; S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}; Lap = \begin{bmatrix} 1 & 2 & 1 \\ 2 & -12 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad (1)$$

Different kernels have been studied and are discussed in further sections. Also, during this stage circular padding is applied to ensure the preservation of the image size and the visual tileability of the generated texture. The concatenated results provided by the 4 kernels are fed to the neural network that models the update rule as they provide each cell's state together with data from its neighbourhood.

The neural network employed for learning the update rule is displayed in Figure 2. It is important to note that all of its filter kernels are of size 1×1, meaning a cell only updates its state based on the input received by the perception, and does not interfere with the processing of other states. This simulates the per-cell, independent, state processing. The first convolutional layer employs n_f filters. In our experiments we use different values for the n_f hyperparameter. These specific values are presented in Section 3. The neural network generates the values needed to update each cell's state, resulting in an output size $h \times w \times n_s$ (h - height, w - width) similar to the input automaton state, and the necessity of using n_s filters in the second convolutional layer.

The output modification values are applied stochastically, to break symmetry and relax the expectancy of global synchronization for our self-organising system. This can also be seen as a per-cell dropout [3]. In our experiments, half

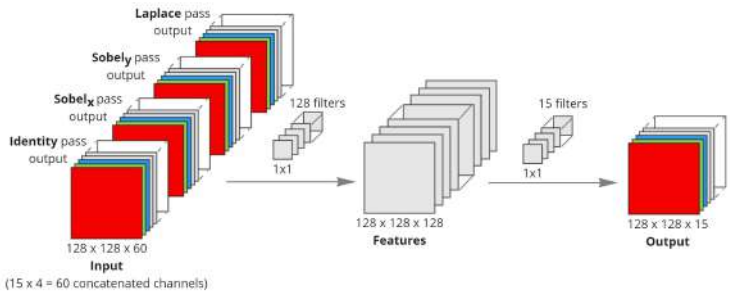


Figure 2: The NN that models the update rule based on the perception output.

the number of the total cells updates at each timestamp.

Using this architecture, we can generate textures of varying sizes. During training, we use 128×128 examples. At inference, we can generate both 128×128 textures and smaller or larger rectangular textures.

2.1.2 Training of the NCA

In the current context, training the NCA involves training the neural network that models the update rule. We initialize the NCA with a uniform state and evolve it iteratively for t steps. At timestamp $t + 1$ we measure the quality of the generated texture against the provided example and apply backpropagation to the neural network. In our experiments, t is a randomly generated number between 32 and 96, as utilized and tested in [3]. The training proccess is illustrated in Figure 3.

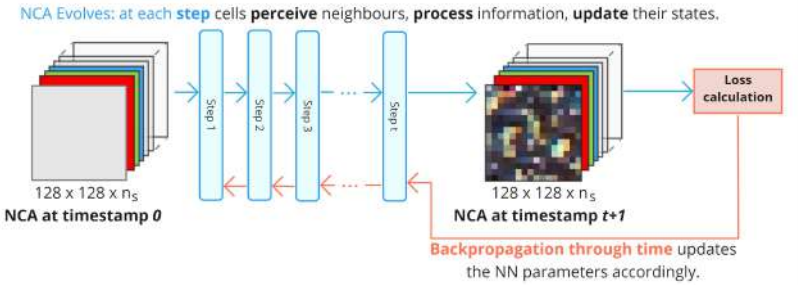


Figure 3: A training step. The NCA runs iteratively through the NN for t steps, as illustrated in Figure 1. The loss is then calculated and the the NN's parameters are updated by backpropagation through time.

Loss function

Our NCA should imitate a given example's style, not generate a pixel copy of it. Therefore, we must train our state processor, represented by the neural

network illustrated in Figure 2, accordingly. Most style transfer methods rely on the distributions of feature maps provided by a neural network whose layers are considered to capture style. We name it an *observer* or *differentiable texture discriminator* [9, 15]. We pass the RGB channels of our NCA and the example image through the observer and drive the selected feature maps distributions to match.

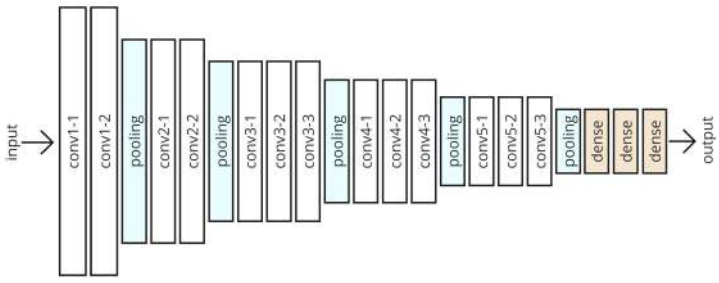


Figure 4: VGG16 architecture

The NCA uses a trained VGG16 network as the differentiable texture discriminator. The choice of VGG is deliberate, as most style transfer approaches utilize VGG variants due to superior results compared to other architectures [9, 3]. The activations provided by a selection of $L = 5$ layers from the VGG16 network (conv1-1, conv2-1, conv3-1, conv4-1, conv5-1) is used, considering the architecture as shown in Figure 4. The loss calculation process is illustrated in Figure 5.

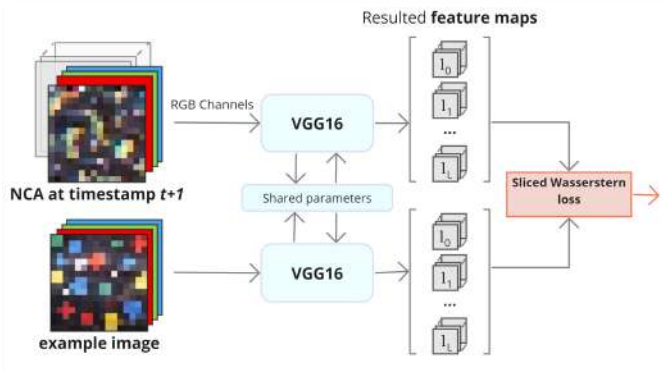


Figure 5: The loss calculation process. The feature distributions for the specified L layers are extracted by passing the state's RGB channels through VGG16. They are then matched to those of the example image using SWL.

Although most texture synthesis algorithms rely on feature distribution matching based on Gram matrices [16, 3], their limitations have been thoroughly

discussed. The recent work of [16] demonstrates the superiority of the Sliced Wasserstein Loss (SWL) in capturing the complete set of feature distributions. Therefore, we employ the SW Loss for our experiments.

We select feature maps from L layers of the observer network for the target image I, and the generated image, $\tilde{I}$. We denote the textural loss by $\mathcal{L}(I, \tilde{I}) \in \mathbb{R}^+$. For a layer l we have $M_l = h \times w$ (w - height, w - width) feature vectors: $F_m \in \mathbb{R}^{c_l}, m \in \{0, 1, \ldots, M_l - 1\}$, where c_l is the depth dimension of the feature map. The set of distributions for these vectors is noted p^l for the features obtained by passing the example image through the VGG, respectively $\tilde{p}^l$, obtained by passing the generated image through the VGG.

The SWL operates on the sets of distributions p^l and $\tilde{p}^l$, for all $l \in L$. Its value is the sum of the distances between the distributions extracted from I and $\tilde{I}$ at each layer l, as formalized by

$$\mathcal{L}_{SW}(I, \tilde{I}) = \sum_{l=1}^{L} \mathcal{L}_{SW}(p^l, \tilde{p}^l) \tag{2}$$

The distance between the distributions p^l and $\tilde{p}^l$ is calculated by approximating the optimal transport (OT) between them (note that the transport map is not optimal but the optimized distribution is proven to converge towards the target distribution [16]):

$$\mathcal{L}_{SW}(p^l, \tilde{p}^l) = \mathbb{E}_v[\mathcal{L}_{SW1D}(p^l_V, \tilde{p}^l_V)] \tag{3}$$

The approximation is done through the SW distance, as defined in Eq 3: the expectation of the one dimensional optimal transport (1D OT) distances after projecting the feature points onto random directions $V \in \mathcal{S}^{c_l}$ (here, p^l_V and $\tilde{p}^l_V$ correspond to the projections of the feature points on a direction V). Projecting over several directions leads our histogram to converge towards the target histogram [17].

We project p^l and $\tilde{p}^l$ feature vectors on a random direction using dot product with the vector V and obtain two sets of scalars. The 1D OT is calculated by sorting the scalars and applying Mean Squared Error (MSE), known also as L_2 norm, over the obtained sets:

$$\mathcal{L}_{SW1D}(S, \tilde{S}^l) = \frac{1}{|S|} \left\| sort(S) - sort(\tilde{S}) \right\|^2 \tag{4}$$

A simplified visualization of the SWL is illustrated in Figure 6 and a pseudocode for this algorithm is presented in Algorithm 1. First, we reshape the feature tensor of shape $h \times w \times c_l$ into $M_l \times c_l$. We then project these features over 32 random directions (unit vectors of dimension c_l), sort them and measure the L_2 distance. The strategy of applying this loss is portrayed in Figure 5. A detail not covered in the figure for simplicity is the overflow loss term, given in equation 5, that we add alongside the SWL to keep all state channels in the interval $[-1, 1]$.

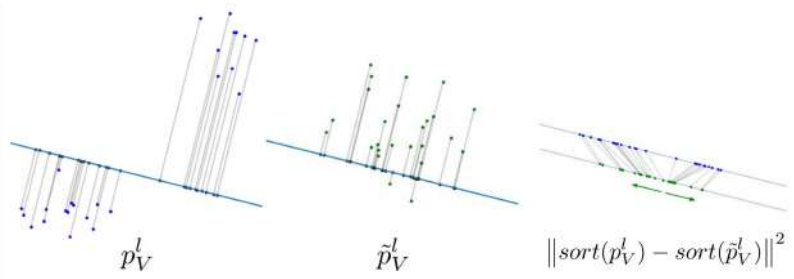


Figure 6: The Sliced Wasserstein loss visualized for a 2D example. By computing the L_2 difference between sorted lists of projections from the target distribution (p_V^l) and the obtained distribution ($\tilde{p}_V^l$) we push the NN to reach the targeted distribution. This process is done for several projection directions to capture all necessary information.

$$\mathcal{L}_{overflow}(I) = \sum_{x \in I} |x - clip_{[-1,1]}x| \tag{5}$$

Note that $clip_{[-1,1]}x$ returns a vector of the same dimension, with all values clipped in the $[-1, 1]$ interval. This stabilises training by preventing drift in latent channels and aids in post-training quantisation [2]. The complete formula for our loss is:

$$\mathcal{L}(I, \tilde{I}) = \mathcal{L}_{SW}(I_{RGB}, \tilde{I}) + \mathcal{L}_{overflow}(I) \tag{6}$$

Algorithm 1 Sliced Wasserstein Loss Computing

Input:
$source_l$ - features received from VGG's l layer for the NCA output, with (c_l, M_l) shape
$target_l$ - features received by the same layer for the example image of the same shape
Output: the SWL for the given layer - $\mathcal{L}_{SW}(p^l, \tilde{p}^l)$
function SW_LOSS($source_l, target_l$)
$proj\_n \leftarrow 32$ ▷ number of projections
$Vs \leftarrow rand\_like(c_l, proj\_n)$ ▷ random directions (normalized values)
$source\_proj \leftarrow dot(source_l, Vs)$ ▷ project source in Vs
$target\_proj \leftarrow dot(target_l, Vs)$ ▷ project target in Vs
$diff \leftarrow sort(source\_proj, axis = 1) - sort(target\_proj, axis = 1)$
return $mean(square(diff))$
end function

Pooling

We pass NCA states to the training step in batches. A *sample pool* based strategy is necessary for the long-term stability of the automaton [18, 14, 3], explicitly the behaviour of the NCA outside the few steps considered during training. If no pooling strategy is applied, the behaviour of the automaton

after the 96 steps threshold would be unstable. We employ the same strategy presented in [14, 3] meaning we construct a pool of 1024 future textures, and seed states of the NCA. Batches of 8 elements are selected from this pool, and after the training step presented in Figure 3 the new states are placed back into the pool. Also, at batch selection, the highest loss scoring element is replaced with a seed - the uniformly initialized image - to enforce seed evolution, and texture stability and avoid training on irrelevant hallucinations during the first stages of training.

2.2 Multi-texture generation

Signals are the central component of animal interaction and an essential part of animal life. Internal signals include hormonal or neural signals within the body, bioelectrical and genomic signals during development [19]. Research studying the integration of internal signals in the morphogenesis model of NCA has been conducted [11, 19, 20], but not in the context of self-organising textures.

Integrating internal signals in our NCA boils down to attributing special meaning to a few channels of the cell's hidden state. As illustrated in Figure 7, the hidden n_h channels of the hidden state are now divided into n_c communication channels and n_g genomic channels.

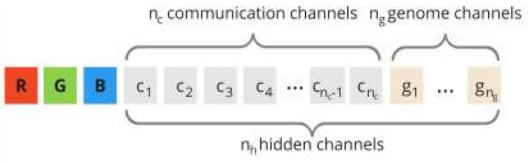


Figure 7: The state vector of a cell in a multi-texture generation NCA. It comprises n_s components: 3 RGB channels, n_h hidden channels consisting of n_c communication channels and n_g genome channels.

Our experiments include $n_g = 1$, $n_g = 2$ and, respectively, $n_g = 3$ genomic channels, corresponding to 2, 4 and 8 different textures generated by one NCA. Since we employ the overflow loss to stabilise training, encouraging all channels to hold values in the [-1, 1] interval, we needed an appropriate encoding strategy. Therefore we chose binary encodings, as genome channel values will initially be either 0 or 1. Furthermore, n_g genomic channels allow generation of 2^{n_g} textures and a texture index in its binary coding will allow (ideally) interpolation with all other textures, an advantage further detailed in Section 2.3. An example further discussed in the results section is depicted in Figure 8 where we see the correspondence between the expected textures and the genomic encoding. The genome channels are set at timestamp 0 for each cell in the automaton as follows: for a frilly texture, we set them to 000, for a stratified texture - 001 and so on. In pseudocode, this translates to the function written in Algorithm 2. In the algorithm, we initialize all cells in the automaton to be of genome g,

by setting their last n_g channels ($seed[:, :, -n_g:]$) to the binary encoding of the genome index g. All other values are 0.

Algorithm 2 Automaton genome-based initialization

Input:
h, w = height and width of cell plane (size of texture)
n_h = number of cells' state hidden channels, with n_g genome channels,
g = expected genome index
Output: One NCA state of the specified genome
function SEED_OF_GENOME(h, w, g)
$seed \leftarrow zeros(h, w, 3 + n_h)$
$g_{b_2} \leftarrow to\_base\_2(g, n_g)$ ▷ binary representation of g on n_g bits
$seed[:, :, -n_g:] \leftarrow g_{b_2}$
return *seed*
end function

After timestamp 1, we do not interfere with any channels: we do not restrict their modifications or values other than encouraging low values through the overflow loss. It is up to the automaton to *understand* and keep the information throughout evolution in order to reach the expected specimen. Discussions on how the automaton understands and preserves the genome details throughout evolution are addressed in the results and discussions section, see Section 3.

Algorithm 3 Interpolated genome initialization

Input:
n_s = number of cell's state channels, with n_g genome channels,
g_1 = first genome index
g_2 = second genome index
Output: A NCA cell state situated halfway between genomes g_1 and g_2
function INTERP_GENOME(g_1, g_2)
$g1_{b_2} \leftarrow to\_base\_2(g_1, n_g)$
$g2_{b_2} \leftarrow to\_base\_2(g_2, n_g)$
$g_{result} \leftarrow (g1_{b_2} + g2_{b_2})/2$
$seed \leftarrow zeros(n_s)$
$seed[-n_g:] = g_{result}$
return *seed*
end function

Apart from generating textures based on the given 2, 4 or 8 example images, we also study the interpolation behaviour for these NCA. Interpolation refers to creating blended textures, a smooth transition between two given examples. Meaning, if the automaton learnt textures based on Figure 8, we could also create a mossy wood (combination between the first and second images, starting with a seed with genomic channels set to, for example, g=(0,0,0.5)) or grids on the spectrum defined from the third and fourth textures. Technically, we could interpolate between any two learnt genomes: we interpolate between a genomic code $g_1 = (a_0, a_1, \ldots, a_{n_g})$ and $g_2 = (b_0, b_1, \ldots, b_{n_g})$ by selecting all

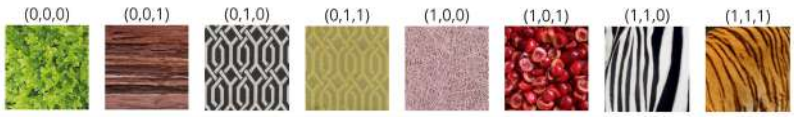


Figure 8: Examples of textures generated by a 3-genome NCA architecture. The tuple for each texture corresponds to the (g_1, g_2, g_3) channels in the seed state. This means that, for example, if we want to create a cherry texture, at timestamp 0 all cells would have the state vector with all values set to 0, except the last and second-to-last position which would be set to 1.

pairs where $a_i = b_i$ and setting $c_i = a_i$ and for all pairs where $a_i \neq b_i$ setting c_i to an intermediate value (eg. 0.25, 0.5, 0.75, etc.). The result will be $g_3 = (c_0, c_1, \ldots, c_{n_g})$. Experiments showed that interpolations, where more than one intermediate value has to be set in the genome, are more unstable and unpredictable. The algorithm for creating a genome situated at an equal distance between two learnt genomes is depicted in Algorithm 3. While interpolation behaviour has been studied for multiple texture synthesis models [21], it is specific to our architecture in the NCA realm given that other NCAs only develop one texture. Interpolation is performed during inference and does not influence the training strategy.

The optimal adapted training strategy for multi-texture generation relies on a pool with equally divided seeds for each texture. When replacing the highest-scoring state during batch selection, we replace it with a seed with the same initial genome coding. Moreover, we cycle through genome coding replacements: we don't choose the highest-scoring state, but the highest-scoring state belonging to the genome g_r, and then move on to the next g_r. Practically, we replace from the first batch the highest-scoring state belonging to the genome g_0, in the second batch we replace the state belonging to genome g_1 and so on. This ensures that more intricate textures, that are harder to learn and yield an overall higher loss during training are not always replaced, since it would lead to instability during the inference for those textures.

Algorithm 4 outlines the pooling-based training strategy for generating multiple textures. First, the pool is initialized ($init\_pool(pool\_size)$) by creating $pool\_size/n\_genomes$ seeds of each genome and placing all of them in the shared pool. For each element in the pool, we track the NCA state (x) and the index of the corresponding genome (y). This tracking is essential because the genome channels in the NCA state change throughout evolution, and we need to identify the corresponding genome at any time for loss calculation and eventual replacement. Next, we initialize the NCA parameters, particularly those of the neural network that models the update rule. The training process runs for a specified number of epochs, where, in each iteration, a batch is selected, prepared, run through the NCA for updates, and then returned to the pool. The NCA weights are adjusted based on the loss calculated for each batch. This iterative process allows the algorithm to evolve and refine the textures within the pool.

Algorithm 4 Pooling strategy based training for multi-texture generation

Input: The $pool\_size$ of nca states, $number\_of\_epochs$ and $batch\_size$
Output: The trained nca, the final pool

```
pool ← init_pool(pool_size)
nca ← init_nca_params()
for iteration ∈ range(0, number_of_epochs) do
    batch ← random pick batch_size elements from pool
                                        ▷ states and corresponding genome indices
    g_r ← iteration%n_genomes
    if exists g_r genome in batch then
        worst_{g_r} ← state_with_highest_loss_of_genome(batch, g_r)
    else
        worst_{g_r} ← state_with_highest_los(batch)
    end if
    batch[worst_{g_r}].x ← seed_of_genome(batch[worst_{g_r}].y)
    num_steps ← random(64, 90)
    for i ∈ range(0, num_steps) do
        batch.x = nca(batch.x)
    end for
    loss ← compute_loss(batch.x, target_images(batch.y))
    apply nca weights backpropagation
    pool ← put updated individuals of batch back in the pool
end for
return nca
```

2.3 Regeneration and Grafting

A well-studied property of NCAs is regeneration [14]. No adaptation of the training strategy is required for the single-texture experiments, and the automaton inherently regenerates and stabilises the pattern shortly after being damaged. However, slight alterations in the training strategy must be applied for the multi-texture architecture. Without these modifications, the NCA generates patches of different textures instead of regenerating the governing texture.

Fortunately, only a few adjustments must be made. The automaton reaches the desired behaviour using the adaptation presented in [9]. During batch sampling from the pool, besides replacing a high-scoring state we also damage the lowest-scoring state. Damaging here refers to randomizing the state vectors for cells contained in a circle of a radius of 15 to 25 pixels. The adapted training procedure is concisely shown in Algorithm 5, as an altered version of Algorithm 4.

Other experiments include grafting visualizations. Grafting is the act of joining two organisms to continue their growth together. In our case, we consider grafting two or more types of cells, belonging to different genomes, coexisting in one automaton. We do this by initializing, either at timestamp 0 or at a random timestamp, some cells with the seed values of a different genome. By doing this, we enable two textures to coexist and interact in the same automaton. This yields to interesting results, studied in further sections. Cells of differ-

Algorithm 5 Regeneration-adapted training for multi-texture architecture

Input: the $pool\_size$ of nca states, $number\_of\_epochs$ and $batch\_size$
Output: The trained nca, the final pool

```
pool ← init_pool(pool_size)
nca ← init_nca_params()
for iteration ∈ range(0, number_of_epochs) do
    batch ← random pick batch_size elements from pool
                                  ▷ states and corresponding genome indices
    g_r ← iteration%n_genomes
    replace_highest_loss_of_genome(batch, g_r)                ▷ as in Alg. 4
    x_1, x_2 ← lowest_2_losses(batch)
    damage_texture_from_batch(batch[x_1])
    damage_texture_from_batch(batch[x_2])
    nca_train_step(nca)                                       ▷ as in Alg. 4
    loss ← compute_loss(batch.x, target_images(batch.y))
    apply nca weights backpropagation
    pool ← put updated individuals of batch back in the pool
end for
return nca
```

ent genomes can be structured in multiple ways, by creating concentric circles, stripes etc. of different genome cells. An example for generating a texture where the left half belongs to one genome and the right half belongs to another genome is depicted in Algorithm 6.

Algorithm 6 Grafting inference

Input: nca = nca model with associated hyperparameters
h, w = height and width of expected texture
g_0 = expected genome index for the left half of the texture
g_1 = expected genome index for the right half of the texture
t_max = timestamp at which to extract the generated texture
Output: The grafted texture (RGB image) at timestamp t_max

```
function GRAFT(h, w, g_0, g_1, t_max)
    left_seed ← seed_of_genome(h, w/2, g_0)
    right_seed ← seed_of_genome(h, w/2, g_1)
    texture ← concat(left_seed, right_seed, dim = 1)
    for t ∈ range(0, t_max) do
        texture ← nca(texture)
    end for
    return texture_RGB
end function
```

3 Results and Discussions

In order to analyze and evaluate the impact of the different hyperparameters, like size of the first hidden layer and loss function, as well as the influence of

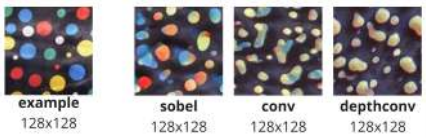


Figure 9: All generation snapshots are taken at timestamp 90. From left to right we see the texture generated with Sobel and Laplace perception kernels, 4 learnt convolution kernels, 4 learnt depthwise convolution kernels.

the genome on the generation of different texture, a series of experiments were performed. Furthermore the possibility of using the NCA for more complex tasks like interpolation of textures, regeneration of damaged textures and grafting is studied and evaluated.

The images for all experiments are selected from the Describable Textures Dataset [22] that groups of 5640 images, organized in 47 terms (categories) inspired by human perception: banded, bubbly, bumpy, frilly etc. Moreover, we selected a few textures from VisTex Database [23]. All experiments were run on T4 GPU and lasted up to 2h each.

3.1 Results of the experiments

The single texture generation experiments studied the influence of different perception filters on the quality of the generated texture. For these experiments we used a pool of 1024 images with a batch size of 8. Training was performed for 5000 epochs.

As represented in Figure 9, the $Sobel_x$, $Sobel_y$, $Laplace$ filters offer enough representation of the neighbors and lead to significantly better results than learned kernels. Of course, leaving the NCA to train longer may enrich the representation provided by learnt kernels. Still, we consider that it would not quantitatively improve the quality output, but it would only add up to the complexity of the NCA representation. Insight on the influence of image processing filters can be studied in [18], where experiments with other combinations of hardcoded filters have been conducted concerning the influence of such filters on the perception stage.

The main focus of this paper is on using a single NCA for the generation of multiple textures, by using the information in the genome channel. Furthermore, some new usages of the NCA were explored. The following experiments and results underline the significance of our work.

3.1.1 Multi-texture generation and interpolation results

This subsection details the tackled NCA architectures for multi-texture generation. Specifically, we study different state formats, the long-term stability of generated textures, and address interpolation behaviour between learnt

genomes. The NCAs are trained for 10000 epochs, with a pool size of 1024 samples and batch size of 8 images. Table 1 summarizes the performed multi-texture experiments.

No. genomes	Experiment name	n_h		n_f	Aim
		n_c	n_g		
2	G2Feasible (G2F)	9	1	96	feasibility of model
4	G4Similar (G4Sim)	9	2	128	similar (sim.) textures
	G4Different (G4Diff)	9	2	128	different (diff.) textures
	G4Structured (G4Str)	9	2	128	structured textures
8	G8Large (G8L)	9	3	128	diff. + sim. textures
	G8Medium (G8M)	6	3	70	small model
	G8SmallNoR (G8SNR)	3	3	30	xs model (no regen.)

Table 1: Multi-texture initial experiments. Sobel and Laplacian filters are used for perception. n_h = number of state's hidden channels with n_c communication channels and n_g genomic channels. n_f = number of filters for the first convolution layer of the NN.

The experiments covered various aspects in the image examples, such as similarity and regularity, and are detailed as follows:

- **Experiment G2Feasible (G2F)** tested whether a proposed genomic coding is enough for the NCA to learn to differentiate between textures. Since the results were satisfactory and the NCA learned to differentiate between the two provided textures, it opened the way for generating more textures and lead to the experiments presented below.
- **Experiment G4Similar (G4Sim)** follows the generation of 4 similar textures by a single NCA, enhancing shared feature representation but struggling with long-term stability of the genomes. The automaton learns to read the genome and differentiate the wanted textures, but, as seen in Figure 10, shows difficulties in maintaining the textures over a large number of iterations. This instability may be prevented by training the NCA for more epochs. All 4-genome experiments training lasted around 1h40min.

Figure 10: Genome stability for experiment G4Sim.

- **Experiment G4Different (G4Diff)** aims the generation of different textures by a single NCA, which means that different features must be interpreted and learnt. It successfully demonstrated that the automaton does not depend on shared features between textures for learning and gathers a deeper understanding of the genome and particularizes its behaviour accordingly. This experiment also uncovered an intriguing behaviour for the artificially generated texture regarding global communication along cells. Specifically, the NCA fails to achieve the global organization of the grid texture as provided for genome 2 (01), but it still produces a texture with the style specifics of the example (Figure 11, second top). This means that the NCA learns the difference between the genomes, but fails to understand the global organization required for the highly structured texture. Referencing the G4Sim experiment, we only changed the textures we provided as examples to the automaton.
- **Experiment G4Structured (G4Str)** followed the above observation regarding the NCA behaviour on textures with large repetitive models and trained highly structured patterns that necessitate a broader communication across cells in order to test the extent of the cell's global organisation. The experiments' results hyphened a shortcoming of our approach, as the automaton can not reach the global organisation of such textures although it does imitate the style of smaller patterns (Figure 11, third top). This limitation is tackled in Section 3.3.
- **Experiments G8Large (G8L) and G8Medium (G8M)** followed 8-texture generation on a pack of different textures, both artificially generated and real-life examples. They covered different texture categories, colors and patterns, while offering enough room for interpolation between examples. Experiment G8M was performed in order to optimize the number of parameters of the NCA used in Experiment G8L, downgrading the 10k parameter architecture to a 4270 parameter architecture with similar results, while keeping all the other training details unaltered. Training these experiments lasted approximately 1h40min.
- **Experiment G8SmallNoR (G8SNR)** followed 8-texture generation on the same texture pack as the other 8-genome experiments, with the scope of testing how small can our neural network be. Since G8M exhibited a slow regeneration process, we decided to drop the regeneration expectancy for this experiment. We therefore obtained a 1500 parameter NCA that can generate 8 different textures and that holds stability up until 500 steps. Roughly, this architecture size would be equivalent to 8 NCAs with 187 parameters that correctly generate the expected textures, a hard-to-complete task. However, our approach's disadvantage in this case would be the long-term instability of the genomes. This architecture trained for 1h20min.

In Figure 11 we visualize the images generated by our experiments, in groups. The conducted experiments were successful. The automaton learned to evolve

numerous textures according to the planted internal signals. Most training experiments were done on an unnecessarily large neural network, as seen in comparing the results between architecture G8L of 10k parameters and architecture G8M of 4k parameters with little impact on output texture quality, visible in the second and third rows of Figure 11. However, the smaller architecture's regeneration process is slowed down. For example, the G8L architecture visually regenerates the first genome in approximately 180 steps, whereas the G8M architecture restores it in approximately 420 steps. Another experiment with an even smaller architecture followed, with 3300 parameters ($n_c = 5$, $n_f = 60$), and the 8 textures was successfully developed similarly to the ones presented above, but the regeneration process lasted 700 steps and the automaton did not recover perfectly. Moreover, the genomes get corrupted around the timestamp 1000.

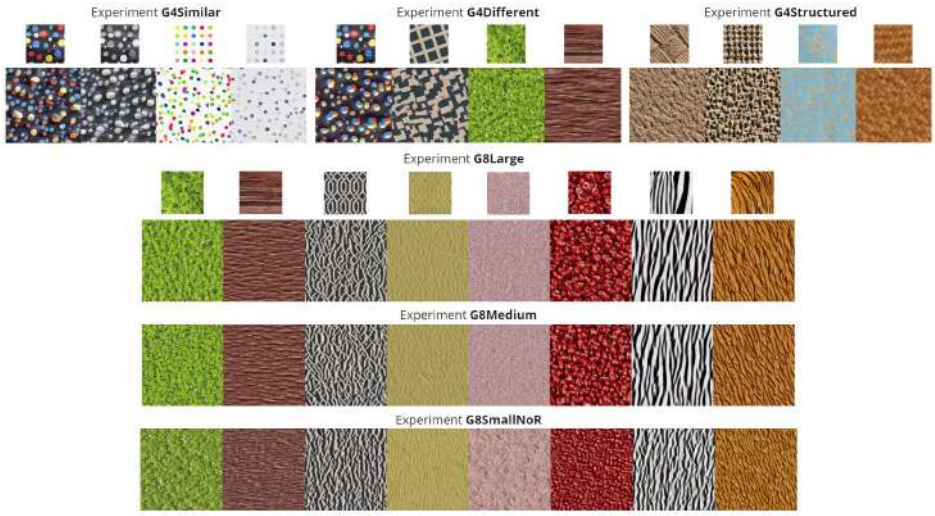


Figure 11: The results for multitexture generation, using the architectures as in 1. Images are grouped according to architectures, each genome representing a texture. For each group of experiments, the top row represents the provided examples, and the bottom illustrates the generated textures after training.

The images selected in Figure 11 are at timestamp 110. At inference, cellular automata can run an indefinite number of steps. In most of our experiments, we observed that the automaton held stability up until 6000 iterations (some even longer) when one genome would get corrupted and start evolving patches of some other genomes.

Furthermore, we test the NCA's ability to generalize learnt features by analyzing its performance on interpolation tasks. Interpolation is a niche texture generation area [24], of interest in computer graphics [25, 26]. This ability is specific to our NCA architecture, as other NCAs developed for texture synthesis

cannot generate multiple textures using one single model.

Figure 12 illustrates the interpolation between two learnt genomes, for experiment G4Sim. We aimed to achieve a blended polka-dot texture between a coloured and grayscale version of the same texture. This is an ideal scenario, as the textures are highly similar. We can observe in Figure 12 the quality of the interpolated textures, which closely resemble those that a programmer could manually create by using color dots derived from each texture. The interpolation method avoids combining multiple colors into a single dot and ensures that the dots maintain alignment with the properties of the original genomes. This observation aligns with the statement in [3] that the cells find an algorithm that generates the patterns. Moreover, specifically on these textures, this example can be extended into multiple color spectrums and enjoy the advantages of interpolation in creating new, blended, images.

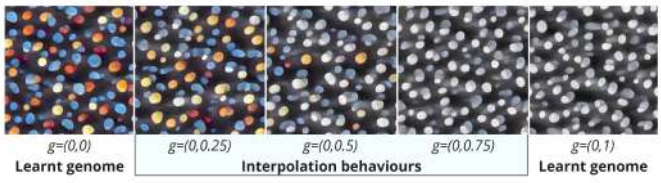


Figure 12: Interpolation of learnt genomes for G4Sim experiment.

Examples covering the interpolation of more distinctive textures are represented in Figure 13. These extend on the aforementioned ideas and demonstrate the quality and utility of interpolation, paving a new area of development for NCAs in texture generation. The loss function heavily influences the interpolation behaviour, and employing other loss functions will lead to varying results, as seen in the experiments conducted in Section 3.3 and in [21].

Figure 13: More interpolated textures, all for the 8-genome experiment G8L. Genomes initialized with: (a) $g = (0, 0, 0.5)$ (b) $g = (0, 1, 0.5)$ (c) $g = (1, 1, 0.5)$ (d) $g = (1, 0.5, 0)$, (e) $g = (0.35, 0.35, 0.35)$.

3.1.2 Regeneration and grafting results

In addition to the experiments described in the preceding sections, some experiments related to regeneration and grafting have also been performed. These

highlight the proposed architecture's extensibility while also providing a stable baseline for texture synthesis using small models.

Regeneration experiments were carried out to test whether our automaton can consistently inpaint a damaged region. This behaviour is studied in different NCA architectures [3, 14, 19] and the inpainting task is also covered by other texture synthesis architectures [21]. The outcome of our experiments emphasizes the rich representation of the genomic texture that leads to successful regeneration. All of our NCAs trained with the adapted methodology enhanced this behaviour. The NCA of experiment G8M had a slower regeneration process due to its reduced architecture.

Figure 14 displays the NCA regeneration behaviour before and after adjusting the training methodology, as presented in Algorithm 5 of Section 2.3.

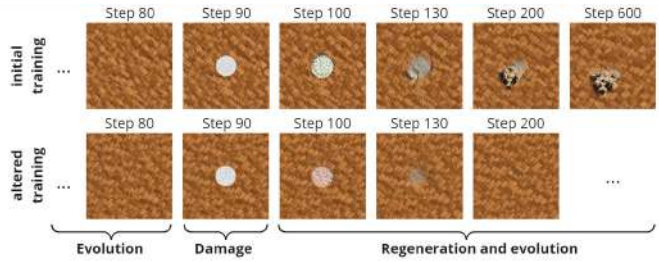


Figure 14: Regeneration before and after adjusting the training methodology, exemplified on the automaton of experiment G4Dif.

Grafting experiments highlight the rich context in which textures can be generated and combined. As discussed, interpolation tasks involve blending between existing textures to create a new one, combining features of both in a smooth transition. On the other hand, grafting combines distinct textures onto a single surface and often requires careful alignment and blending for a cohesive appearance. Recent implementations of grafting techniques in this context include utilizing compatible neural cellular automata instances. For example, [27] defines compatible NCA instances as instances where their weights at training are initialized with those from a common trained ancestor. These instances provide the base for texture grafting, alongside interpolation at the border between the two textures provided by passing one image through both NCA instances and masking them to create the final result. While our architecture allows for this grafting type, we see the benefit of being able to obtain grafting based just on the genome channel with a single automaton.

We *graft* textures by either initializing, at timestamp 0, one NCA with different genomes in the patches we want, or running a NCA on one genome, selecting a patch at a given timestamp and transferring that patch over to another NCA, with different genomes. Of course, given that the combined automaton will continue to evolve iteratively, the patches will modify shape, location and area over time. If this is an unwanted behaviour, we could create the illusion of grafting by running 2 or more instances of the NCA, each with an expected genome, and

layering the generated textures over each other by masking unwanted patches for each.

In Figure 15 a visual example of the initialization with different genomes for grafting is presented. In Figure 15 (a) the yellow color represents genome $g_1 = (0, 0, 1)$, while the blue color represents genome $g_0 = (0, 0, 0)$. The shades between yellow and blue represent genomes for which the last channel has intermediate values in the range of $[0, 1]$, enabling a smooth transition between the two textures. Figure Figure 15 (b) illustrates the comparison of the initialization mask with the results of the evolution of the nca at timestamp 30. The results of this initialization on the evolution of the nca at timestamp 110 can be seen in Figure 15 (c).

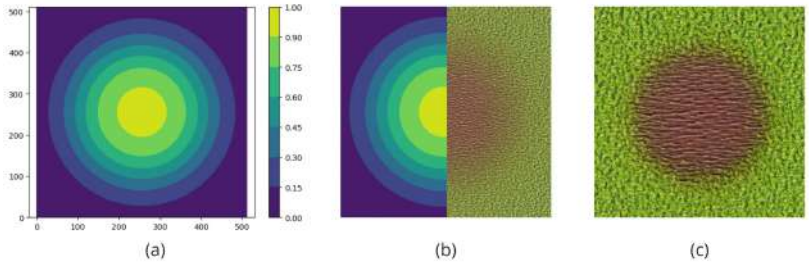


Figure 15: Grafting the G8L nca. Initialization example with genomes $g_0 = (0, 0, 0)$, $g_1 = (0, 0, 1)$. (a) Initialisation of the last genome channel, (b) Comparison of the channel initialization with the evolution of the nca at timestamp 30, (c) The state of the nca at timespamp 110.

Figure 16 illustrates a four textures generated through grafting techniques. We observe that the automaton can develop patches of specific genomes. Moreover, we see a communion where the two or more genomes collide, forming a consistent boundary transition area of cells, whether visualized as a barrier (as seen in (b), (d)) or a smooth transition ((a), (c)). Nevertheless, the transition behavior between genomes remains consistent across the intersection line.

In the light of all the described experiments, two main questions arose. Firstly we were intrigued by the role of the generation and stability of the desired texture. Thus we studied if the automaton preserves the genome information in the texture cell during evolution and by this learns to discern between the different textures. The second question was about the influence of the loss function on improving the generation of highly-structured textures that require broad-image communication. The results obtained using two different loss functions were compared. In the following these two aspects are presented in more detail.

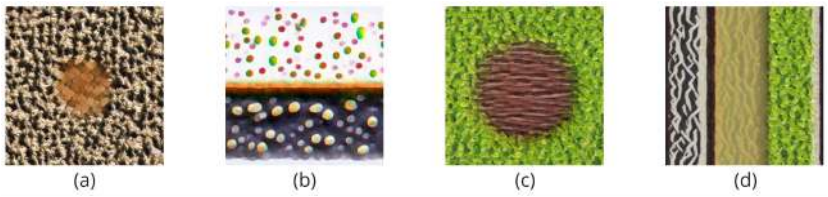


Figure 16: Grafted textures, from left to right, for NCAs of experiment (a) G4Structured: illustrates a NCA formed of cells of genome 2 with a disc of genome 1 situated in the center, (b) G4Similar: the top half is formed out of genome 3 cells, whereas the bottom half is formed out of genome 1 cells, (c) G8Large: cells situated similarly to (a), of genome 1 and 2, (d) G8Large: cells situated in three equally distributed vertical stripes of genomes 4, 3 and 1.

3.2 Preservation of the genome

Given the diversity of the generated textures in our experiments, it is essential to investigate whether the automaton preserves and generates textures based solely on the perception stage and vicinity properties, or if it also maintains genome information to ensure stability.

To investigate this issue, we based our approach on experiment G2F, where the two genomes correspond to the same texture in both color and grayscale. The automaton is more unstable given the similarity of the textures, but we cannot attribute this to the vicinity similarity, given the different colours. Therefore, we create two similar textures and test whether the automaton can learn the difference between them and remain stable over time. We selected the `dotted_0201` texture [22] (Figure 17 left) and created a similar one by deleting the blue polka-dots (Figure 17 right). Note that other colours were not modified, only the blue dots were deleted.

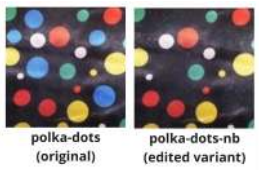


Figure 17: The 2 textures used for this experiment, the left one corresponds to the genome 0, the right one - genome 1.

We name this experiment **G2Preserve (G2Psv)**. The state vector of the NCA for these two textures has $n_s = 13$ values, representing the 3 RGB channels, $n_c = 9$ communication channels and $n_g = 1$ genome channel. The genome channel is set to 0 for the first texture and to 1 for the one without blue dots. We employ $n_f = 128$ filters for the first convolutional layer and train the NCA for 10000 epochs. The training step took 1h30min.

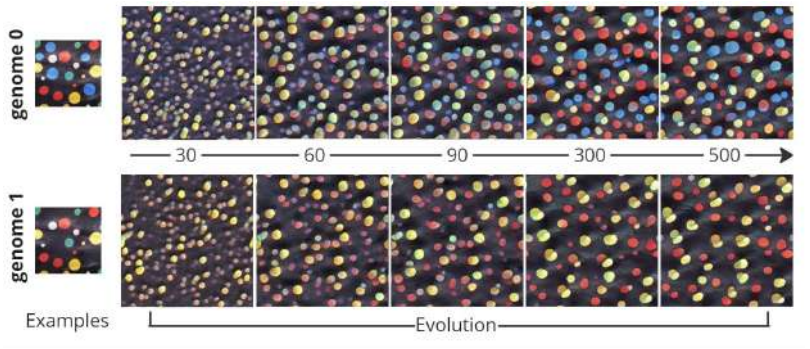


Figure 18: The evolution of the automaton based on the selected genome, in the first row - genome 0, in the second row - genome 1. The examples on the first column of size 128×128 are provided for comparison to the generation of the textures of size 256×256.

Figure 18 displays the texture generation results for each of the two genomes after different timestamps. It can be observed that the automaton has learned to differentiate between the two textures despite their extreme similarity, suggesting that the automaton considers the genome throughout evolution.

We also remark an intriguing, unexpected behaviour. Monitoring the evolution of the NCA, we notice in the second texture (with genome 1) the emergence of small blew dots at evolution step 60. While not yet prominent, it would have likely continued evolving them into dots akin to those seen in the texture generated by genome 0, had the automaton not considered the genome (see this transformation of blue dots throughout genome 0 generation in Figure 18 steps 60-90). However, for genome 1 the formed dots are dimmed and at step 500 no such dots are in formation, supporting the idea that the automaton does keep the genome information throughout evolution.

To further support this hypothesis, we provide two figures constructed in similar manner. Figure 19 illustrates the generated textures alongside the genome channel values for two inferences of this experiment, one with cells' belonging to genome 0 (top two rows) and one with cells' belonging to genome 1 (bottom rows). The genome channel is visualized through the colormap corresponding to the colorbar represented on the right of the image, where black corresponds to a genome channel value of -1, and yellow to a genome channel of 1. We observe that the genome channel values are as expected up until around timestamp 60, for genome 0, the channel's values are close to 0 and for genome 1 the channel's values are close to 1. Although the values converge at later timestamps, we pose that the automaton has assigned certain genomic values to each texture style specifics and uses them, alongside the other channels, to maintain texture stability over time. To support this statement, we direct the attention towards the genome channel for the genome 0 NCA run, at timestamp 500 (last column of row 1 in Figure 19). The lower values, approaching -1, visualized as black and

dark-purple dots correspond directly to blue dots in the RGB correspondent of the NCA state (row 0, last column), meaning, the automaton has attributed low genome values to texture features specific to the provided example for genome 0, and higher values otherwise, as the common features represent both genome 0 and 1. For the genome 1 generation (third row) we see higher values overall, no deep purple spots in the genome channel at timestamp 500. The lower genome values correspond to the blue, faded dots and areas that are prone to the generation of such dots in the texture; values are later corrected by the self-organising system.

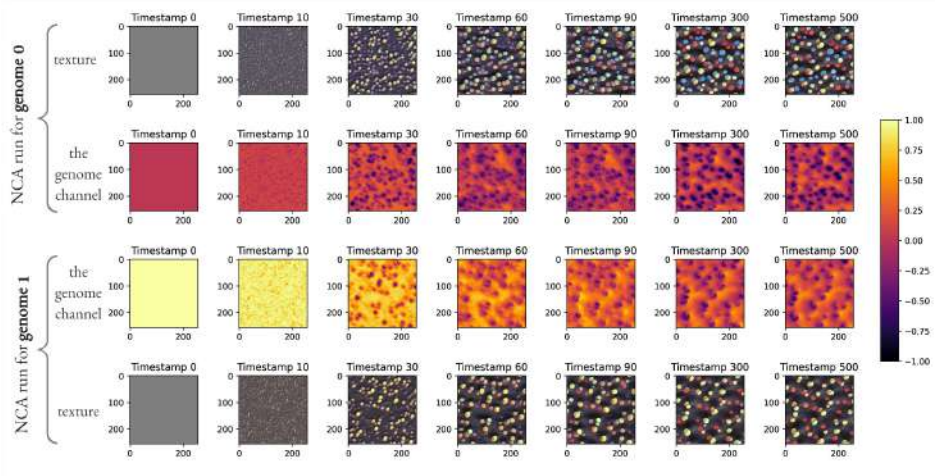


Figure 19: Analysis of the genome channel for the G2Psv NCA. The first and bottom rows illustrate the generated textures, whereas the second and third rows represent the corresponding genome channels.

We also provide the last genome channel analysis for textures generated by experiment G2Diff. The textures representing genome 00 and 01 have no similar features, as the ones discussed in the aforementioned example, resulting in a clearer divide between the values representing the last genome channel. The last genome channel values for the top rows keep values around 0, as the values for the generation of genome 01 keep higher values throughout evolution.

Furthermore, upon comparing the textures generated with automaton G2Psv for genome 0 with those generated by genome 0 for the G4Sim NCA, we remark an uptick in the occurrence of red, orange, and yellow dots, accompanied by a decrease of blue and green dots. We also observe throughout training that the G2Psv automaton learned to only generate the edited (polka-dots-nb) texture up until the 2000 epoch. Before and at this timestamp, the automaton generates the edited version of the texture no matter what genome we set for in the seed. Only after this epoch, the automaton learns to generate the blue dots corresponding to genome 0. This highlights the complex interactions between the textures during training and the NCAs tendency to generalize learnt behaviour.

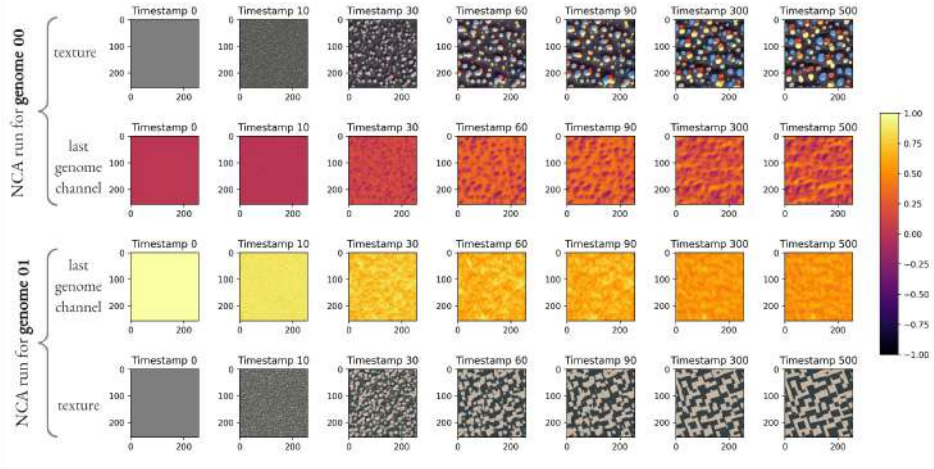


Figure 20: Analysis of the genome channel for G4Diff NCA.

In these experiments, the NCA has learnt to evolve red, orange and yellow dots faster through the texture provided for genome 1, consequently influencing the generation process for genome 0.

3.3 Loss function exploration

Utilizing the SW loss function, the automaton performs strongly on both near-regular and irregular structures. However, it exhibits weaker performance on textures that necessitate broader communication across the automaton, such as images that contain large repetitive patterns. Examples of such given and generated textures are depicted in Figure 21. It's noteworthy that this behavior is intentional and expected in many cases, as it allows for the capture of fine details while disregarding potential irregularities in the given examples, without straining the texture. The NCA does capture the style of the given image, but does not reach a similar global state. Nevertheless, for the showcased instances, we would prefer the NCA to prioritize learning the knitted, knotted, checkered or tiling structures over focusing solely on the finer details of the examples.

For comparison, we employ the loss function termed OTT Loss proposed by [2], specialized for regular patterns. We conducted a 4-genome experiment on the regular patterns of Figure 21 using the architecture details as presented in Section 2.2 for experiment G4Sim, using our SW Loss and OTT Loss respectively. A further comparison of the performance of the two losses is obtained by the 8-genome experiment with 4270 parameters, utilizing the G8M architecture, and the outcomes can be visualized in Figure 22. Both experiments highlight the limitations of each loss: SWL produces favorable results on most textures, while OTT Loss excels in generating structured patterns. Examining the generation of the fibrous texture (Figure 22 g_4), it is notable that SWL captures

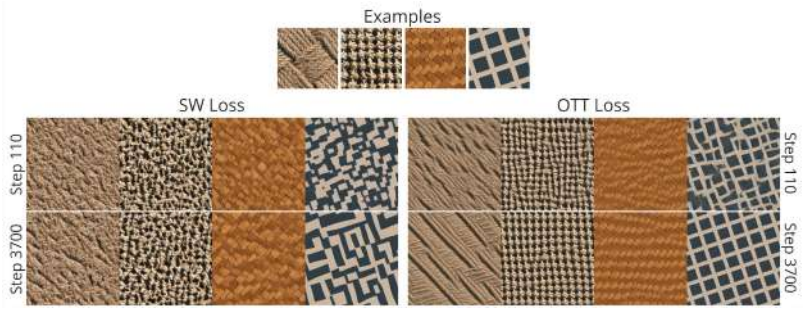


Figure 21: Generation of fabric textures (extracted from [23]) and a grid texture (extracted from [22]) using the SW Loss and OTT Loss.

Figure 22: Based on the examples of the top row, generated textures (size 256×256) using the same architecture and parameters, but utilizing SWL Loss and, respectively, OTT Loss during training.

a better understanding of the texture, while OTT Loss generates small dots against a pink background. The interlaced texture (Figure 22 g_3), characterized by similar colours for foreground lines and background is less accurately captured by the OTT Loss, as are the properties of the frilly texture (Figure 22 g_0) and pitted texture (Figure 22 g_5).

Overall, SWL captures more details, as illustrated in Figure 22, given that both trainings rely on 4270 parameters. However, the OTT Loss addresses the aforementioned shortcoming of the SWL in enhancing broader communication along the cells, as depicted in Figure 21. Regeneration is slowed down using the OTT Loss (from 410 steps for regenerating the first genome to 1210), and interpolation leads to images as displayed in Figure 23 (for comparison with the textures displayed in Figure 13).

We conclude that the OTT loss covers the shortcoming of the SW Loss for the NCA, which does not accurately reproduce textures with regular, relatively large patterns. This makes OTT loss the preferred one int the case of such texture (ideally artificial [2]) pattern generation. For a broader approach, we

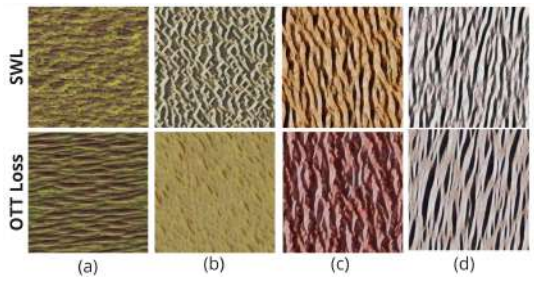


Figure 23: Interpolation behaviour comparison of the NCA trained on the same methodologies with 4270 parameters but with SWL and, respectively, OTT Loss. Genomes for interpolation are considered as in Figure 13.

consider that SWL covers most cases of the presented examples. A hybrid approach may also lead to better results for both regular and irregular patterns and is one of the things to consider in future experiments.

4 Conclusions

Neural cellular automata are an active research field with many promising future opportunities. Self-organising structures are both studied from the software point of view and hardware implementations are emerging. Cell division, regeneration, and grafting offer promising prospects in the context of physics, robotics and swarm robotics, artificial intelligence and biology, offering a captivating approach to studying and understanding dynamic, self-organising systems. Real-time robust synthesis of high-quality texture pictures can be achieved using the lightweight NCA. More significantly, it exhibits an amazing zero-shot generalization capability to several post-training adjustments, including local coordinate transformation, speed control, and resizing.

This study aimed at increasing the usability of neural cellular automata in the context of texture generation, pinpointing and providing a solution for the limitation of using for each texture a dedicated automaton. For this scope, we applied the idea of providing model context through internal signals, previously used in NCAs trained for growing models from one cell [19]. By employing a similar approach, we developed a novel architecture that enriches one NCA to generate multiple textures. We improved the training methodology to enhance the generation and stability of all textures, by cycling through genomes in the genome replacement stage of batch preparation. Also, we provided qualitative and quantitative analysis regarding the extent to which our automaton preserves and uses the defined genome channels throughout evolution, supporting our statement that the genome channels are indeed maintained and further used in the automaton's development. In essence, we trained the automaton to express the genomically coded signals accordingly, thus having the self-organising system form and behave conclusively with its genome coding. In order to achieve

satisfying results on most types of textures, we employed an efficient loss function, Sliced Wasserstein Loss, supporting the statement that the SWL better captures style than former Gram-based solutions, utilized in the base inspiration for our implementation [3].

Moreover, we extended the study of neural cellular automata in the context of texture interpolation, an area previously inaccessible due to the restricted ability of only generating one texture per automaton. Interpolation is natural to our automaton, as it can derive at inference the styling for an intermediate texture between two learnt examples. We studied this behaviour and analyzed its interactions with the used loss function. Also, we examined grafting techniques using a single instance of a NCA as a host for cells belonging to different genomes, as opposed to former solutions where multiple instances were used to test this behaviour. Visual results were displayed and discussed.

To conclude, this study treated neural cellular automata as a model of morphogenesis and utilized genomic coding to make it exhibit the wanted stylistic properties of textures based on given examples. We encourage the use of NCA in software solutions by providing an extensible yet compact and easy-to-train architecture for texture synthesis. The proposed model was also studied in adjacent research contexts, such as regeneration, grafting and interpolation, and exhibits promising abilities, encouraging future developments of embedding expected behaviour into the NCA model. Furthermore, given the interest in extending NCAs to 3D textures, our study can also contribute to offering new opportunities for improvement and generalization in this area.

References

[1] Li-Yi Wei, Sylvain Lefebvre, Vivek Kwatra, and Greg Turk. State of the Art in Example-based Texture Synthesis. In *Eurographics 2009, State of the Art Report, EG-STAR*, pages 93–117, Munich, Germany, March 2009. Eurographics Association.

[2] Alexander Mordvintsev and Eyvind Niklasson. μnca: Texture generation with ultra-compact neural cellular automata, 2021.

[3] Eyvind Niklasson, Alexander Mordvintsev, Ettore Randazzo, and Michael Levin. Self-organising textures. *Distill*, 2021. https://distill.pub/selforg/2021/textures.

[4] Leon Gatys, Alexander S Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[5] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of

Proceedings of Machine Learning Research, pages 1349–1357, New York, New York, USA, 20–22 Jun 2016. PMLR.

[6] W. Xian, P. Sangkloy, V. Agrawal, A. Raj, J. Lu, C. Fang, F. Yu, and J. Hays. Texturegan: Controlling deep image synthesis with texture patches. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8456–8465, Los Alamitos, CA, USA, jun 2018. IEEE Computer Society.

[7] Ehsan Pajouheshgar, Yitao Xu, Tong Zhang, and Sabine Süsstrunk. Dynca: Real-time dynamic texture synthesis using neural cellular automata. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[8] Yitao Xu. 3d texture synthesis using graph neural cellular automata, 2023.

[9] Alexander Mordvintsev, Nicola Pezzotti, Ludwig Schubert, and Chris Olah. Differentiable image parameterizations. *Distill*, 2018. https://distill.pub/2018/differentiable-parameterizations.

[10] E. F. Castejon C. M. Almeida, J. M. Gleriani and B. S. Soares-Filho. Using neural networks and cellular automata for modelling intra-urban land-use dynamics. *International Journal of Geographical Information Science*, 22(9):943–963, 2008.

[11] *Adversarial Takeover of Neural Cellular Automata*, volume ALIFE 2022: The 2022 Conference on Artificial Life of *Artificial Life Conference Proceedings*, 07 2022.

[12] Sam Earle, Justin Snider, Matthew C. Fontaine, Stefanos Nikolaidis, and Julian Togelius. Illuminating diverse neural cellular automata for level generation. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '22, page 68–76, New York, NY, USA, 2022. Association for Computing Machinery.

[13] Kazuya Horibe, Kathryn Walker, and Sebastian Risi. *Regenerating Soft Robots Through Neural Cellular Automata*, pages 36–50. 03 2021.

[14] Alexander Mordvintsev, Ettore Randazzo, Eyvind Niklasson, and Michael Levin. Growing neural cellular automata. *Distill*, 2020.

[15] Alexander Mordvintsev and Ettore Randazzo. Texture generation with neural cellular automata. *CoRR*, abs/2105.07299, 2021.

[16] Eric Heitz, Kenneth Vanhoey, Thomas Chambon, and Laurent Belcour. A sliced wasserstein loss for neural texture synthesis. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021.

[17] F. Pitie, A.C. Kokaram, and R. Dahyot. N-dimensional probability density function transfer and its application to color transfer. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 2, pages 1434–1439 Vol. 2, 2005.

[18] Sorana Catrina, Mirela Catrina, Alexandra Băicoianu, and Ioana Cristina Plajer. Learning about growing neural cellular automata. *IEEE Access*, 12:45740–45751, 2024.

[19] James Stovold. Neural Cellular Automata Can Respond to Signals. volume ALIFE 2023: Ghost in the Machine: Proceedings of the 2023 Artificial Life Conference of *Artificial Life Conference Proceedings*, page 5, 07 2023.

[20] *Gradient Climbing Neural Cellular Automata*, volume ALIFE 2022: The 2022 Conference on Artificial Life of *Artificial Life Conference Proceedings*, 07 2022.

[21] Antoine Houdard, Arthur Leclaire, Nicolas Papadakis, and Julien Rabin. A generative model for texture synthesis based on optimal transport between feature distributions. *J. Math. Imaging Vis.*, 65(1):4–28, jun 2022.

[22] *Describing Textures in the Wild*, volume Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), 2014.

[23] MIT Vision and Modeling group. Vistex database.

[24] Jonathan Vacher, Aida Davila, Adam Kohn, and Ruben Coen-Cagli. Texture interpolation for probing visual perception. volume 33, 12 2020.

[25] Z. Bar-Joseph, R. El-Yaniv, D. Lischinski, and M. Werman. Texture mixing and texture movie synthesis using statistical learning. *IEEE Transactions on Visualization and Computer Graphics*, 7(2):120–135, 2001.

[26] Roland Ruiters, Ruwen Schnabel, and Reinhard Klein. Patch-based texture interpolation. *Computer Graphics Forum*, 29(4):1421–1429, 2010.

[27] Ehsan Pajouheshgar, Yitao Xu, Alexander Mordvintsev, Eyvind Niklasson, Tong Zhang, and Sabine Süsstrunk. Mesh neural cellular automata, 2024.

Engineering Applications of Artificial Intelligence 133 (2024) 108380

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

Research Paper

Fractal interpolation in the context of prediction accuracy optimization

Alexandra Băicoianu [a], Cristina Gabriela Gavrilă [a], Cristina Maria Păcurar [a,*], Victor Dan Păcurar [b]

[a] Faculty of Mathematics and Computer Science, Transilvania University of Braşov, 50 Iuliu Maniu Blvd., Braşov, Romania
[b] Faculty of Silviculture and Forest Engineering, Transilvania University of Braşov, 1 Şirul Beethoven Street, Braşov, Romania

ARTICLE INFO

Keywords:
Machine learning
Fractal interpolation
LSTM
Synthetic data
Meteorological data
Optimization

ABSTRACT

This paper focuses on the hypothesis of optimizing time series predictions using fractal interpolation techniques. In general, the accuracy of machine learning model predictions is closely related to the quality and quantitative aspects of the data used, following the principle of *garbage-in, garbage-out*. In order to quantitatively and qualitatively augment datasets, one of the most prevalent concerns of data scientists is to generate synthetic data, which should follow as closely as possible the actual pattern of the original data.

This study proposes three different data augmentation strategies based on fractal interpolation, namely the *Closest Hurst Strategy*, *Closest Values Strategy* and *Formula Strategy*. To validate the strategies, we used four public datasets from the literature, as well as a private dataset obtained from meteorological records in the city of Braşov, Romania. The prediction results obtained with the LSTM model using the presented interpolation strategies showed a significant accuracy improvement compared to the raw datasets, thus providing a possible answer to practical problems in the field of remote sensing and sensor sensitivity. Moreover, our methodologies answer some optimization-related open questions for the fractal interpolation step using *Optuna* framework.

1. Introduction

Developing successful Artificial Intelligence (AI) and machine learning (ML) models requires access to immense amounts of high-quality data, as it is widely acknowledged that the performance of most ML models depends on the quantity and diversity of data. However, collecting the necessary amount of labelled training data can be cost-prohibitive. Thus, developing various strategies to improve the quantity and quality of data is of utmost importance.

Our research focuses on the use of interpolation to enhance the quality of the predictions of ML models. The interpolation technique that we adopt is fractal interpolation which provides interpolants that are not necessarily differentiable functions at every point. Since differentiability implies smoothness and a continuous behaviour, interpolating data using functions with this property tends to oversimplify or smooth out some of the irregular and rough patterns that are specific to real-world data, especially at smaller scales. This smoothing effect can lead to a loss of crucial information, making the interpolation less suitable for fitting real-world data. Furthermore, fractal interpolation allows interpolants that can capture the inherent roughness and self-similar structures often found in real-world data. These interpolants can better replicate the complexity and irregularity present in natural phenomena.

On one hand, we develop three different strategies for the pre-processing step of data, which all use fractal interpolation. Our first strategy follows a similar approach to the one used by Raubitzek and Neubauer (2021). However, we managed to solve a series of issues, such as answering the question of optimal choice of the vertical scaling factors involved in fractal interpolation. Moreover, we present a detailed scheme of all the steps involved, which makes our research highly reproducible, and through *Optuna* framework we optimize the prediction model presented. The other two strategies, Closest Values Strategy and Formula Strategy are new approaches that have not been considered in literature before as far as we know.

On the other hand, besides testing our strategies with an ML model fed with public datasets, we also provide examples using real meteorologic data to put our research in the existing research context. This opens a new gate for researchers in the field to obtain much-needed data either when sensors break, or when finer data are needed, based on data recorded at larger time intervals. The response time of a temperature monitoring sensor is producer-dependent and defines how fast the sensor can adapt to temperature changes in a defined period of time, thus influencing the frequency of data logging. The sampling rate, which determines the time resolution of data, depends on the sensors' response time (generally correlated with the price of the device). Developing better interpolation techniques could be very useful for enabling time resolution enhancement, and making it possible, for example, to integrate in the predictive models input data (with high time resolution) subsets obtained from the raw data recorded by sensor-loggers installed in the area for climatological purposes (less expensive

* Corresponding author.
E-mail address: cristina.pacurar@unitbv.ro (C.M. Păcurar).

https://doi.org/10.1016/j.engappai.2024.108380
Received 28 May 2022; Received in revised form 9 October 2023; Accepted 1 April 2024
Available online 11 April 2024

devices, with a typical sample rate of 1 h, but with a much better spatial coverage). This matter highly sustains the utility of the present study, which aims to find new and improved techniques for data interpolation, namely for modifying data time resolution.

Generating synthetic data, or data augmentation for time-series data, has been an important research issue for many researchers. Among utilization of data augmentation, we mention augmenting sparse datasets (Forestier et al., 2017), generating controllable datasets (Kang et al., 2019), moving block bootstrap (Bergmeir et al., 2016), a.o. For a comprehensive review on time series augmentation for deep learning see Wen et al. (2021). Among applications of data augmentation, we also mention time series classification, Fawaz et al. (2018), Guennec et al. (2016), Iwana and Uchida (2021) and Kamycki et al. (2020) or improving the accuracy of forecasting (Bandara et al., 2021, Lee and Kim, 2020 and Raubitzek and Neubauer, 2021). Moreover, it is noticeable that research on time-series data augmentation proved interpolation to be a robust method (Oh et al., 2020).

Classical interpolation methods are often used in prediction machine learning techniques (Bélisle et al., 2015; Jia and Ma, 2017, Meijering, 2002, Wu et al., 2020 and Yadav and Ray, 2021). Interpolation techniques have proven to be an essential and effective tool in reconstructing incomplete datasets (Chai et al., 2021).

Raubitzek and Neubauer have recently introduced fractal interpolation in data preprocessing for machine learning (Raubitzek and Neubauer, 2021). However, our approach addressed several challenges, including determining the optimal selection of vertical scaling factors in fractal interpolation. Moreover, we optimize the presented prediction model using the Optuna framework. Two strategies that we use, namely the Closest Values Strategy and the Formula Strategy, represent novel approaches that have not been used before.

Fractal interpolation is a method for generating interpolation points between a given set of data $\Delta = \{(x_i, y_i), i \in \{0, 1, 2, \dots, N\}\}$, where N is a natural number. The main difference between fractal interpolation and other types of interpolation techniques is the outlook of the result of interpolation, which is a continuous function that is not differentiable everywhere. Thus, fractal interpolation is more relevant for fitting real-world data. Fractal interpolation has applications in a vast range of areas, such as computer graphics (Manousopoulos et al., 2008), image compression (Bouboulis et al., 2006 and May, 1996), reconstruction of satellite images (Chen et al., 2011), single-image super-resolution procedure (Zhang et al., 2018), reconstruction of fingerprint shape (Bajahzar and Guedri, 2019), signal processing (Navascués, 2010 and Zhai et al., 2011), reconstruction of epidemic curves (Păcurar and Necula, 2020) and others.

Research combining machine learning and fractal analysis features has been performed before in combination with Support-Vector Machine (see Ni et al., 2011 that focuses on the enhancement of stock trend prediction accuracy by combining a fractal feature selection method with a support vector machine, demonstrating its superiority compared to five other commonly used feature selection methods, and Wang et al., 2019 where the stock price indexes are forecasted, offering improved accuracy compared to three other commonly used models) or Time-Delayed Neural Network (see Yakuwa et al., 2003 where there is shown an improved short-term prediction accuracy compared to a back propagation-type forward neural network).

2. Materials

2.1. Datasets

This section presents the experimental datasets used in the current research study. Their properties sustain the significance of interpolation techniques in the prediction process, but also serve as validation for the methodology that we will propose with respect to the quantitative aspect of the data.

2.1.1. Meteorological data

The data set was provided by the Forest Meteorology-Climatology Laboratory, from the Faculty of Silviculture and Forest Engineering, Transilvania University of Brasov, more precisely it was extracted from the database recorded by the automatic weather station HOBO®RX3000 (research-grade), deployed at the Sânpetru Education and Research Base, located about 10 km north-east of Braşov City Centre (45.71°N, 25.65°W).

The temperature and relative humidity values were measured and recorded every 10 min by an S-THB smart sensor (Hoboware, produced by Onset Computer Corporation). This device is designed to operate in a range from −40 °C to 75 °C, with an accuracy of ±0.21 °C (from 0° to 50 °C, thus in the temperature range of the three autumn months considered in this study) and a resolution of 0.02 °C at 25 °C.

The sensor response time is 5 min (in air moving 1 m/s), consequently, the time resolution of temperature data measurements (10 min) was adequately established. For the data logger programming an important element is the sampling time interval, which depends on the sensor response time (a shorter sampling interval is not acceptable). This issue highly sustains the utility of the present study, which aims to find new, improved techniques for data interpolation, namely for modifying data time resolution. For studying the regional mountain climate, the Forest Meteorology-Climatology Laboratory operates a dense network of temperature and relative humidity data loggers (HOBO Pro v2 Temp/RH logger) deployed in Postavaru Mountains (on different elevations, aspect, wind exposure etc.) with similar accuracy and resolution, but with a response time of 40 min, which forces the sampling interval to be higher, being consequently set at 1 h (suitable for climatological studies but with the time resolution enhancement useful for other applications).

For this study, the temperature data were formatted with two decimals, as considered adequate for developing and testing the interpolation technique. For meteorological applications, the temperature data resulting from direct measurements should be rounded to one decimal (corresponding to readings at the ordinary meteorological thermometer).

The data set is composed of 13 105 temperature entries recorded between 1 September 2021, 0:00:00 and 30 November 2021, 23:50:00. The file is in a *.csv* format with a size of 385 kB.

2.1.2. Additional public datasets

In view of the research by Raubitzek and Neubauer (2021), we consider four additional public datasets: Shampoo sales with 36 data points, (Kaggle, 2022), Airline passengers with 144 data points, (Kaggle, 2022), Annual wheat yields in Austria with 57 data points (FaostatDocs, 2022), and Annual maize yields in Austria with 58 data points (FaostatDocs, 2022).

The consistent differences between our study and the research from Raubitzek and Neubauer (2021) are convincingly outlined by using the same datasets. Moreover, comparing the methods on the same datasets highlights the progress in this direction of research that our study provides.

2.2. Prerequisites

2.2.1. Fractal interpolation

Fractal interpolation was introduced by Barnsley (2012) and it has since been intensively studied and applied.

To interpolate the data set $\Delta = \{(x_i, y_i), i \in \{0, 1, 2, \dots, N\}\}$ consider the equations

$$\begin{aligned} a_i &= \frac{x_i - x_{i-1}}{x_N - x_0} \\ c_i &= \frac{x_N x_{i-1} - x_0 x_i}{x_N - x_0} \\ d_i &= \frac{y_i - y_{i-1}}{x_N - x_0} - s_i \frac{y_N - y_0}{x_N - x_0} \\ e_i &= \frac{x_i y_{i-1} - x_0 y_i}{x_N - x_0} - s_i \frac{x_N y_0 - x_0 y_n}{x_N - x_0}, \end{aligned} \tag{1}$$

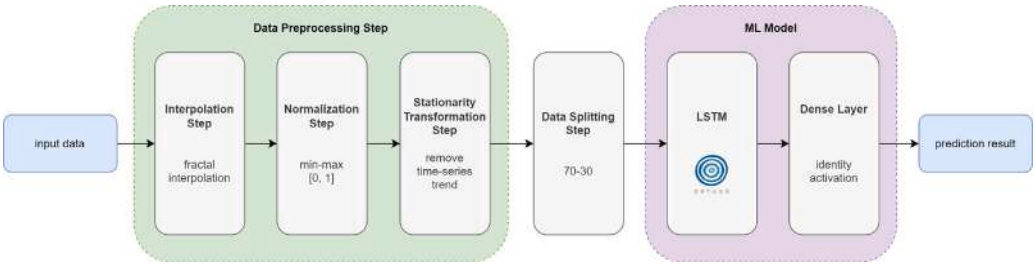


Fig. 1. Methodology outline.

where $s_i \in (-1, 1)$ is called the vertical scaling factor.

Let the family of functions $f_i : [x_0, x_N] \times Y \to [x_0, x_N] \times Y$ defined as

$$f_i \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a_i & 0 \\ d_i & s_i \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} c_i \\ e_i \end{pmatrix}, \tag{2}$$

for every $i \in \{1, \dots, N\}$.

Given a metric space (X, d), the pair $((X, d), (f_i)_{i\in\{1,\dots,N\}})$ is called an iterated function system (IFS, for short) if:

(i) (X, d) is a complete space;
(ii) the functions f_i are continuous, for every $i \in \{1, \dots, N\}$.

The concept of IFS is a notion due to Hutchinson (1981). For an IFS, the fractal operator $F_S : \mathcal{P}(X) \to \mathcal{P}(X)$ is defined as $F_S(K) = \cup_{i\in\{1,\dots,N\}} f_i(K)$, for every $K \in \mathcal{P}(X)$, where $\mathcal{P}(X)$ represents the set of all subsets of X.

Taking $X = [x_0, x_N]$ with the Euclidean metric and the functions f_i defined in Eq. (2), we obtain an IFS.

The fixed point of the fractal operator associated with an iterated function system composed of the functions $(f_i)_{i\in\{1,\dots,N\}}$ is an interpolation function for the system of data Δ called the fractal interpolation function.

We construct the fractal interpolation part based on the above formulas and the code provided by Barnsley in Chapter IV of *Fractals everywhere* (Barnsley, 2012).

2.2.2. Optuna framework

Optuna is an open-source hyperparameter optimization framework that provides multiple state-of-the-art algorithms for sampling hyperparameters ranging from grid sampling strategies to genetic algorithms approaches (Optuna-ReadTheDocs, 2023b).

The main steps for using *Optuna* are as follows:

- Define an objective function to be optimized.
- Create an optimization study *optuna_study*, which will determine the best parameters by running several trials. A trial can be defined as a single execution of the objective function.
- Use one or multiple *suggest* API function calls for the parameters that are subject to optimization inside a trial.

The default hyperparameter sampler is TPESampler which implements the Tree-structured Parzen Estimator algorithm (Optuna-ReadTheDocs, 2023a). The algorithm starts by running the objective function on randomly sampled hyperparameter values from the given domain. After a number of observations, the results are divided into two groups depending on whether they fall below or above a certain quantile of the observed values of the objective function, thus separating the best hyperparameter values from the others. In every iteration, the two groups are updated and a Gaussian Mixture Model (GMM) is fitted to each group, resulting in two densities, $l(x)$ for the best hyperparameters values and $g(x)$ for the remaining ones, where x is the value of the hyperparameter. The algorithm will select the value that maximizes the ratio $\frac{l(x)}{g(x)}$.

3. Method and procedures

This section presents the methodology of the present study. We emphasize the steps which are of utmost importance for the final results in Fig. 1. Furthermore, in the following sections, we will explore each block included in the diagram and highlight its role in the whole process.

It is noteworthy that the main contribution of this paper is concentrated on the interpolation step, which is placed in a time series prediction pipeline. The main goal is to evaluate how the data augmentation step, through fractal interpolation, can have an impact on the quality/accuracy of the prediction.

The field of modelling weather and climate is getting increasingly popular, so choosing a suitable learning machine model becomes a challenge. LSTM models are particularly suited for predicting climate change because they can recall and utilize past data to inform future forecasts. Seasonality and patterns in climate data are frequently visible and can last many years. Based on past data, LSTM models may successfully identify these patterns and produce precise forecasts. The capacity of LSTM models to manage missing data is one of their key characteristics. Due to several issues, including sensor malfunctions and data gathering gaps, climate data is frequently unreliable. While LSTM models can make predictions even with insufficient data, traditional statistical methods have difficulty handling missing data.

LSTM models can also be trained to adjust to changing situations. Since climate change is dynamic, conventional statistical models frequently have difficulty adjusting to new patterns and trends. The ability to train LSTM models on an ongoing stream of data, on the other hand, enables them to adjust and produce precise forecasts even as the climate changes.

For all these reasons exposed, in this research, we chose an LSTM model to explore the predictions on the data used.

3.1. Data preprocessing step

Data preprocessing is an essential step in the development of successful ML models. This technique requires some data preparation, including cleaning the data and transforming the data such that their quality is enhanced. Incomplete, inconsistent, or inaccurate data that contain errors or outliers can be eliminated in this preprocessing step. There are numerous preprocessing techniques (for a comprehensive book on data preprocessing see Garcia et al., 2014) that produce quality data that lead to high-quality patterns.

Our preprocessing step includes transformations of data (interpolation, normalization, and stationarity) to obtain the most suitable data for applying ML algorithms.

3.1.1. Interpolation step

Real-world data are often noisy, with incorrect or missing values. Interpolation is a method of creating new data points within the range of known data points, so it is a technique for filling in missing values. The interpolation should be used where there is a trend observed in the input data and the requirement is to fill the missing value along with the same trend.

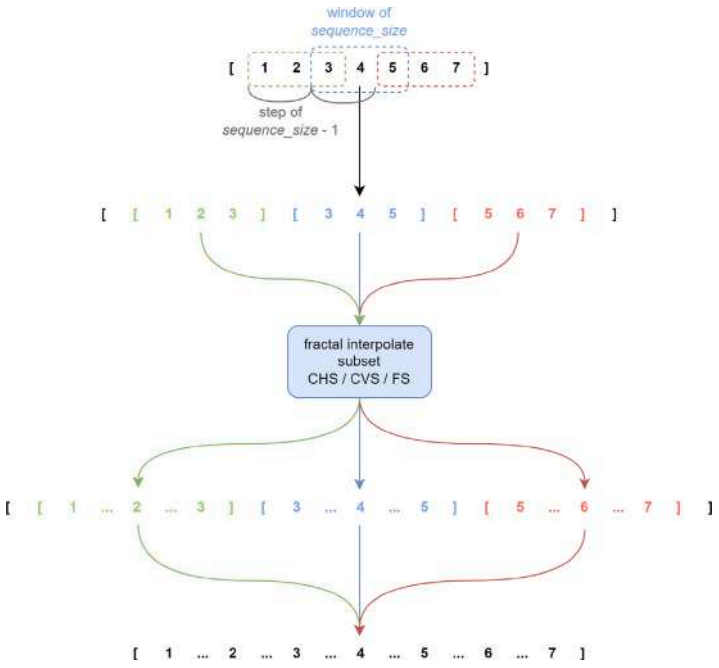


Fig. 2. Interpolation flow diagram for *sequence_size* 3.

We present the complete and detailed steps required for the interpolation step, applied to the datasets considered.

Substep 1: Divide the time series into m subsets of size *sequence_size*

This step consists of splitting the given sequence into m subsets of length *sequence_size* so that the last value in subset i is the first value in the subset $i+1$.

As regards the way we implement this substep, there are some remarks that are worth mentioning:

Remark 1. If the algorithm is used in strict mode, then the dataset is divided into m sequences with equal dimensions; otherwise, the last subset might have a dimension between 3 and *sequence_size* - 1 (at least 3 because 2 points cannot generate new intermediate points in the interpolation process).

Remark 2. At the end of the interpolation, the subsets need to be reunited into a unique list that contains the initial points, chronologically and without repetitions.

Substep 2: The proposed interpolation strategies

In this section, we present three different strategies for the steps specific to interpolation, namely *Closest Hurst Strategy (CHS)*, *Closest Values Strategy (CVS)* and *Formula Strategy (FS)*.

We choose different strategies, firstly to obtain validation for the results from Raubitzek and Neubauer (2021), and most importantly, to enhance the results and obtain improved techniques. We test our methods for both the public datasets Maize (Annual maize yields in Austria), Shampoo Sales, Airline Passengers, Wheat (Annual wheat yields in Austria), see details in Section 2.1, as well as the original data set, Weather described in Section 2.1.1. As regards the latter, for all strategies we use the data corresponding to the first week of entries (1 September 2021–8 September 2021) and we chose the data recorded every hour, to better outline the significance of interpolation.

Fig. 2 describes the general flow of the interpolation step. While the diagram is constructed considering a *sequence_size* of 3, note that a particular *sequence_size* was used for implementing the proposed methodologies. Specific details are given in the next sections for each of the defined strategies.

The interpolation step is presented in Algorithm 1, and it is applicable for all the next proposed strategies. The description of the parameters for the FRACTAL_INTERPOLATION procedure are:

- *subset*: subset created from original data as described in **Substep 1**.
- s_i: a vector of vertical scaling factors which dictate how jagged (and fractal) will be the aspect of the generated data, in the sense that, as its name states, it scales the points vertically.
- *n_interpolation*: the number of distinct interpolation points to be generated between every 2 points of the original data

Algorithm 1 Pseudocode for Fractal Interpolation Computation

1: **procedure** FRACTAL_INTERPOLATION(*subset*, s_i, *n_interpolation* = 17)
2: Compute the interpolation factors a_i, c_i, d_i și e_i based on Equation (1)
3: Generate interpolation points based on Equation (2) until between every 2 points from the *subset* there are *n_interpolation* distinct interpolation points

I. Closest Hurst Strategy (CHS)

The first Strategy is similar to the one employed by Raubitzek and Neubauer (2021). We will refer to it as Closest Hurst Strategy (CHS).

For each subset resulting from Interpolation Substep 1 with *sequence_size* 10, the Algorithm 2 is performed. Note that the parameters have the same signification as previously described.

Results and analysis for Closest Hurst Strategy

We present the results obtained for the considered datasets based on CHS. Firstly, we show the outcome for the public datasets, with the parameter $s_i \in [-1, 1]$ (Figs. 3–6).

However, following tests, we found that the most appropriate vertical scaling factor must be chosen between $s_i \in [0, 0.2]$. In this case, the differences between the points obtained and the initial points are limited, and the initial outlook of the graphic defined by the initial points is conserved. We show in Figs. 7–10 the results obtained in this case.

For our private data set, Weather, we obtain the results presented in Figs. 11 and 12.

Algorithm 2 Pseudocode of Closest Hurst Strategy

```
procedure CLOSEST_HURST_STRATEGY(subset, n_interpolation = 17)
    Compute the initial_hurst, the initial Hurst exponent
    Generate s_i ∈ [−1, 1] a vector with the same value on all positions, representing the constant vertical scaling factor for the current subset
    for k ← 1, 15 do
        interpolated_subset ← FRACTAL_INTERPOLATION(subset, s_i, n_interpolation)
        Compute the h_new, the Hurst exponent for the interpolated_subset
        if k = 1 then
            h_old ← h_new
            interpolated_result ← interpolated_subset
        else
            if abs(h_new - initial_hurst) < abs(h_old - initial_hurst) then
                h_old ← h_new
                interpolated_result ← interpolated_subset
            else
                Generate a new s_i ∈ [−1, 1]
    return interpolated_result
```

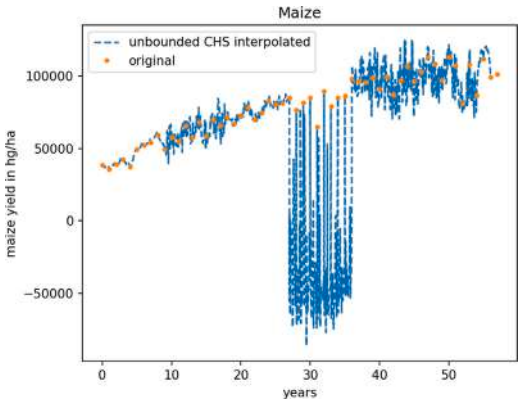


Fig. 3. Maize data set, CHS.

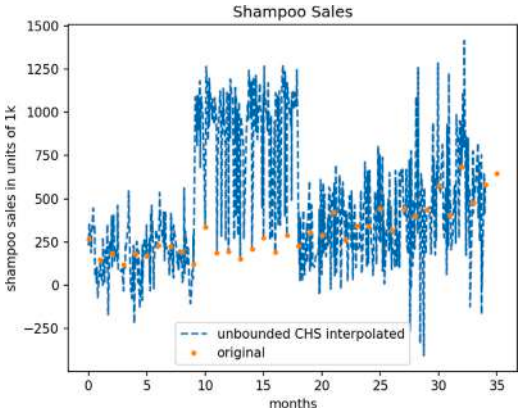


Fig. 4. Shampoo Sales data set, CHS.

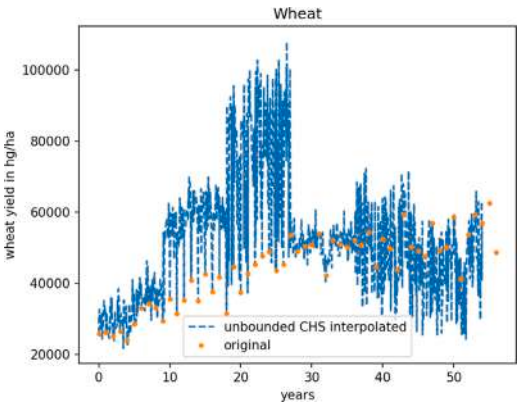


Fig. 5. Wheat data set, CHS.

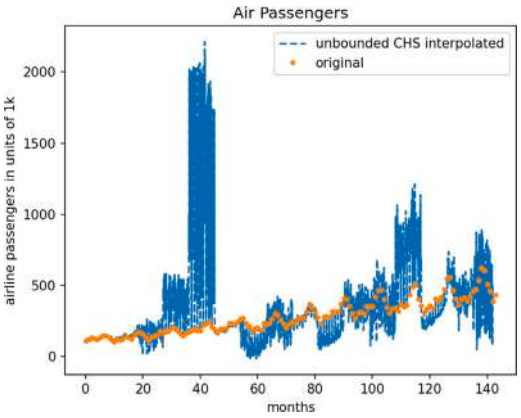


Fig. 6. Air Passengers data set, CHS.

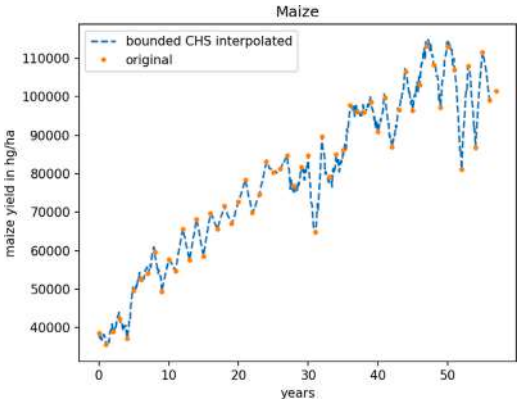


Fig. 7. Maize data set, CHS.

Although the stop condition ensures that the Hurst exponent for the interpolated data is close enough to the initial Hurst value, this does not guarantee the persistence of other properties of the data. This motivates us to define new strategies that ensure the preservation of certain properties of the data through interpolation.

II. Optimized procedure - Closest Values Strategy (CVS)

For this type of strategy, we propose the *Optuna* framework, described in Section 2.2.2.

Thus, in the current CVS strategy, for each data subset obtained using *sequence_size* 10, the steps from Algorithm 3 are performed. Observe that Algorithm 3 defines two procedures, the first one is the objective function that *Optuna* will minimize over the course of 15 trials, having the new parameter *optuna_trial*, and the second one is the main implementation of the strategy. Additionally, procedure LINEAR_INTERPOLATION constructs a linear interpolation of the *subset* considering each interpolation point generated by the FRACTAL_INTERPOLATION procedure, and RMSE represents the Root Mean Square Error.

Results and analysis for Closest Values Strategy

In Figs. 13–17 there are shown the results of the proposed strategy for all five datasets based on CVS. The parameter s_i was optimized by *Optuna* with a possible range set in the interval $[-1, 1]$.

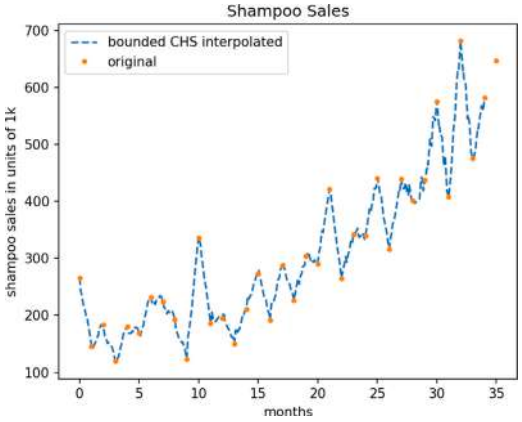


Fig. 8. Shampoo Sales data set, CHS.

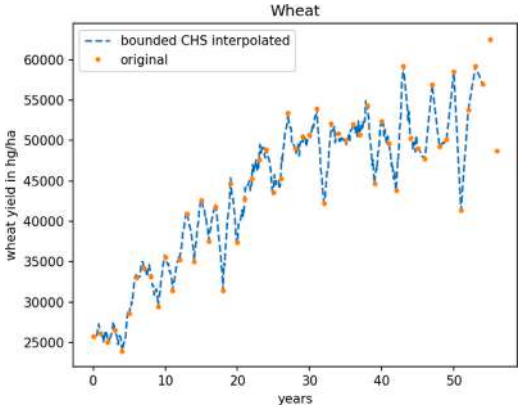


Fig. 9. Wheat data set.

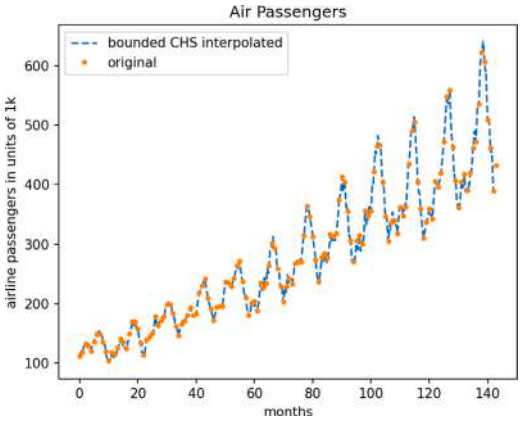


Fig. 10. Air Passengers data set.

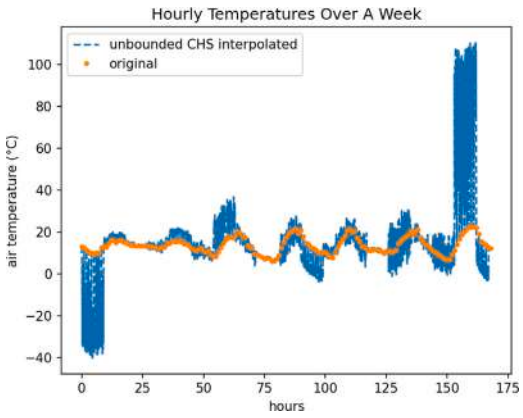


Fig. 11. Weather data set with $s_i \in [-1, 1]$.

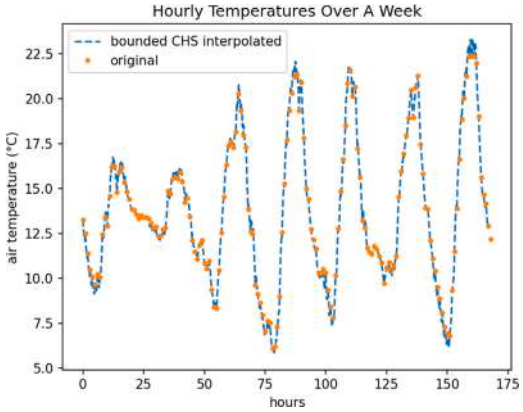


Fig. 12. Weather data set with $s_i \in [0, 0.2]$.

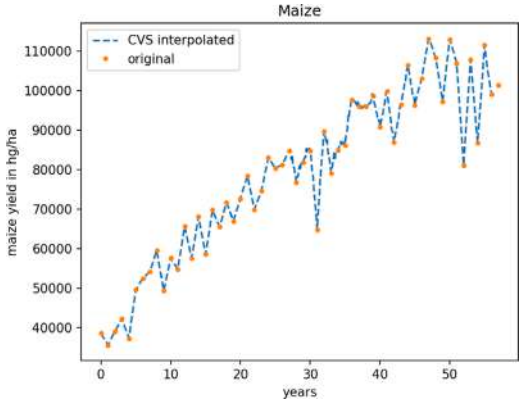


Fig. 13. Maize data set, CVS.

Algorithm 3 Pseudocode of Closest Values Strategy

procedure CLOSEST_VALUES_STRATEGY_OBJECTIVE(*optuna_trial*, *subset*, *n_interpolation* = 17)
  Generate $s_i \in [-1, 1]$, a vector with the same value on all positions, representing the constant vertical scaling factor for the current *subset* in the current *optuna_trial* using *suggest* API
  interpolated_subset ← FRACTAL_INTERPOLATION(*subset*, s_i, *n_interpolation*)
  linear_interpolated_subset ← LINEAR_INTERPOLATION(*subset*)
  return RMSE(*interpolated_subset*, *linear_interpolated_subset*)
▷
procedure CLOSEST_VALUES_STRATEGY(*subset*, *n_interpolation* = 17)
  Create *optuna_study*, a study with direction 'minimize', the objective function CLOSEST_VALUES_STRATEGY_OBJECTIVE() and 15 trials
  s_i ← best trial parameter of *optuna_study*
  return FRACTAL_INTERPOLATION(*subset*, s_i, *n_interpolation*)

It is noticeable that the graphics obtained using CVS resemble the results from CHS with $s_i \in [0, 0.2]$. This is because the RMSE is minimum for the parameter s_i close to the interval $[0, 0.2]$. This can be observed in Fig. 18 where the evolution of the parameter s_i is presented with respect to the objective function described in the CVS context. This result validates our choice of the parameter s_i in the case of CHS.

III. Optimized strategy - Formula Strategy (FS)

For this method, we shall follow a different direction, that is to use a formula to optimize the parameter s_i. Although there is no way to determine a general optimal vertical scaling factor, there are various approaches to optimize this parameter. We follow the ideas from Mazel

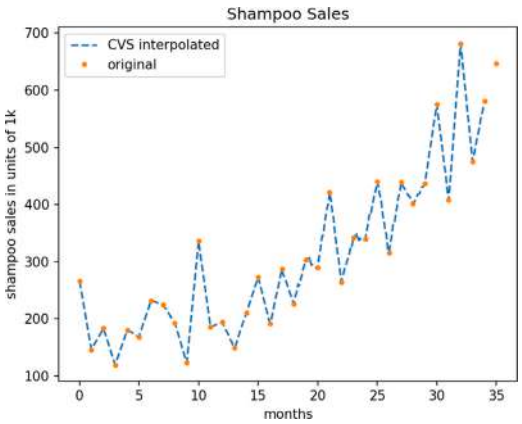


Fig. 14. Shampoo Sales data set, CVS.

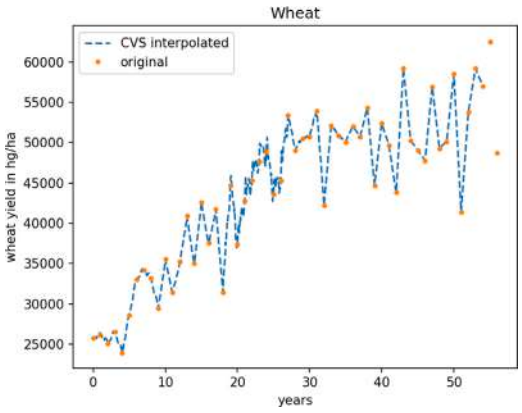


Fig. 15. Wheat data set, CVS.

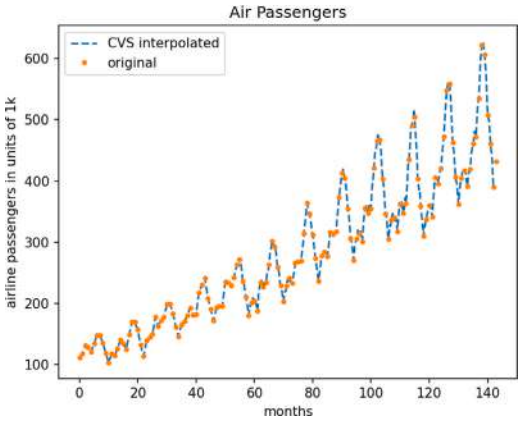


Fig. 16. Air Passengers data set, CVS.

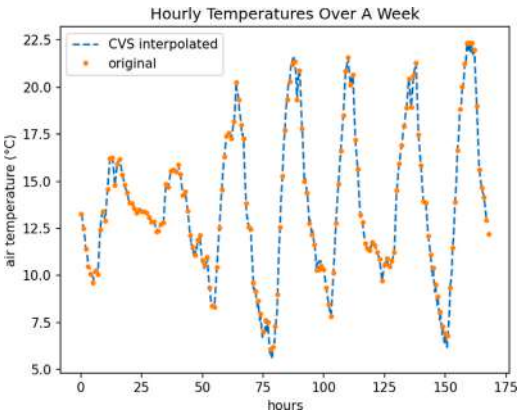


Fig. 17. Weather data set, CVS.

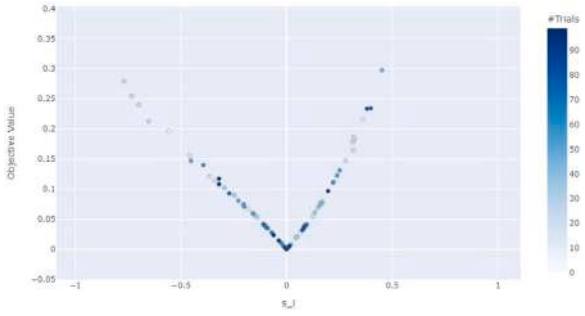


Fig. 18. Evolution of parameter s_i with *Optuna*.

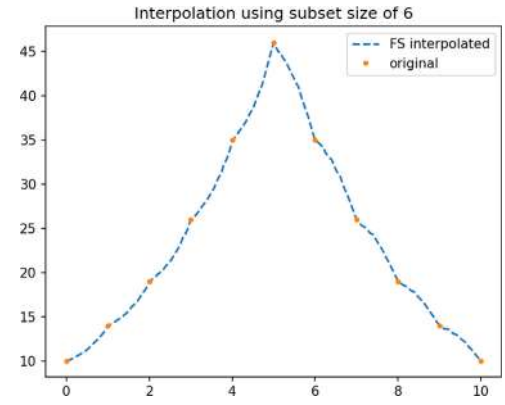


Fig. 19. Interpolation of Γ with *sequence_size* 6.

and Hayes (1992), Manousopoulos et al. (2011) and Gowrisankar et al. (2022).

For the data set $\Delta = \{(x_i, y_i), i \in \{0, 1, 2, \dots, N\}\}$, we choose the parameter s_i as follows:

$$s_i = \frac{y_i - y_{i-1}}{\sqrt{(y_N - y_0)^2 + (y_i - y_{i-1})^2}}, \tag{3}$$

for each $i \in \{1, 2, \dots, N\}$.

However, since the denominator in Eq. (3) becomes closer to 0 when the first and the last data in the subset are too close (the line determined by the start point and ending point of the subset is parallel to the Ox axis), then s_i becomes irrelevantly big, inducing an unwanted variation in the data. Thus, we avoid this by optimizing the current strategy. The optimization is achieved by modifying the *sequence_size* parameter such that the difference between the two ends does not tend to zero.

To exemplify the dependence of the interpolated data on the *sequence_size* parameter chosen, let us consider a trial data set

$$\Gamma = \{(1, 10), (2, 14), (3, 19), (4, 26), (5, 35),$$
$$(6, 46), (7, 35), (8, 26), (9, 19), (10, 14), (11, 10)\}.$$

In Figs. 19 and 20, there are presented the results of interpolation with FS for two different values for *sequence_size*, 6 and 11 respectively.

Therefore, in the context of FS, the initial step is determining the optimal value of the *sequence_size*. For this, we used the procedures defined in Algorithm 4. Note that the optimization is computed for the entire dataset using *Optuna* with 50 trials, as opposed to previous

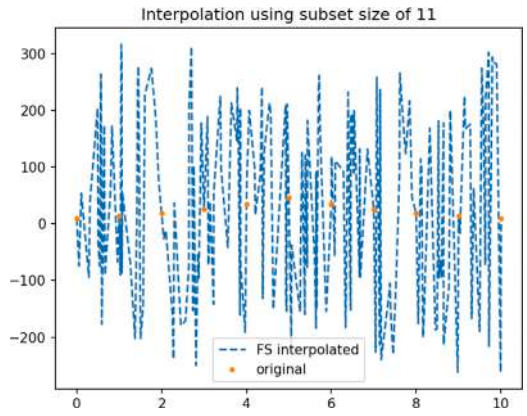


Fig. 20. Interpolation of Γ with *sequence_size* 11.

optimization strategies where the optimization was done at the subset level.

To determine the optimal values for *sequence_size*, we consider the search interval $[4, \text{length(dataset)}-3]$ and allow the fractal interpolation algorithm to work in the non-strict mode, see Remark 1, to minimize data loss.

Algorithm 4 Pseudocode of *sequence_size* optimization

procedure OPTIMIZE_SUBSET_LENGTH_OBJECTIVE(*optuna_trial*, *dataset*, *n_interpolation* = 17)
  Generate subset length *sequence_size* ∈ [4, length(*dataset*) − 3] in the current *optuna_trial* using *suggest* API
  Split *dataset* into *subsets* of length *sequence_size* as described in **Substep 1** from Section 3.1.1
  $total\_RMSE \leftarrow 0$
  for each *subset* **in** *subsets* **do**
    $interpolated\_subset \leftarrow$ FORMULA_STRATEGY(*subset*, *n_interpolation*)
    $linear\_interpolated\_subset \leftarrow$ LINEAR_INTERPOLATION(*subset*)
    $total\_RMSE \leftarrow total\_RMSE + \text{RMSE}(interpolated\_subset, linear\_interpolated\_subset)$
  return $total\_RMSE$
▷
procedure OPTIMIZE_SUBSET_LENGTH(*dataset*, *n_interpolation* = 17)
  Create *optuna_study*, a study with direction 'minimize', the objective function OPTIMIZE_SUBSET_LENGTH_OBJECTIVE() and 50 trials
  sequence_size ← best trial parameter of *optuna_study*
  return *sequence_size*

With regards to FS, for each subset of data, the procedure from Algorithm 5 is executed.

Algorithm 5 Pseudocode of Formula Strategy

procedure FORMULA_STRATEGY(*subset*, *n_interpolation* = 17)
  Compute s_i based on Equation (3)
  return FRACTAL_INTERPOLATION(*subset*, s_i, *n_interpolation*)

Let us note that for the procedure FORMULA_STRATEGY, s_i is not constant for the entire subset (as is the case for CHS and CVS), but it varies according to each interval defined by two consecutive points in the subset.

One of the main advantages of this strategy is that once the optimal dimension of the subset is found, the repetitive process required to execute the optimization routine for the previous two strategies is no longer required.

The optimal value of *sequence_size* for each data set is presented in Table 1.

Table 1
Optimal *sequence_size* values for FS interpolation.

Data set	*sequence_size*
Maize	29
Shampoo Sales	10
Wheat	54
Air Passengers	141
Weather	6

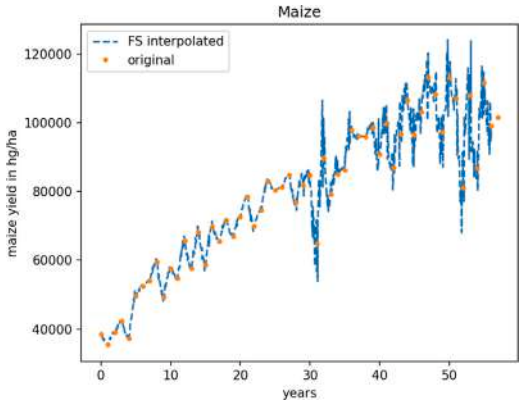


Fig. 21. Maize data set, FS.

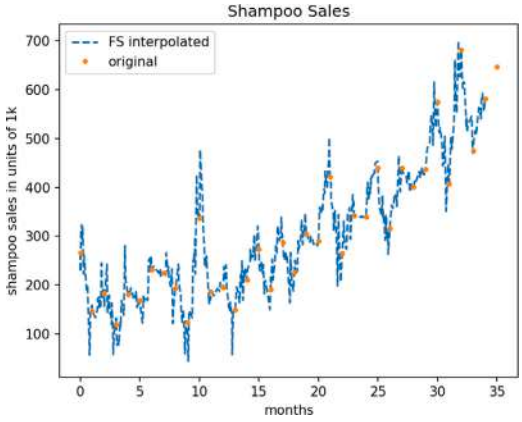


Fig. 22. Shampoo Sales data set, FS.

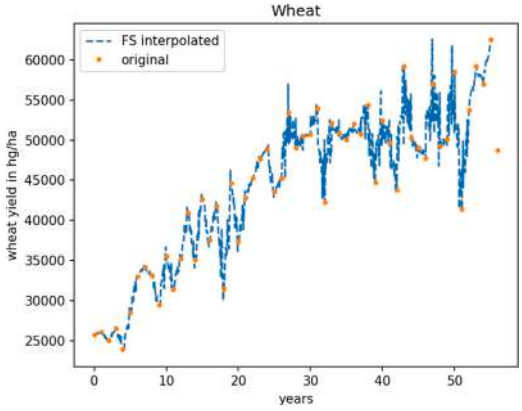


Fig. 23. Wheat data set, FS.

Results and analysis for formula strategy

Applying formula (3) using the optimal *sequence_size* values from Table 1, we obtain the interpolation results presented in Figs. 21–25.

Comparison of methods and results

We proposed three strategies for the interpolation step, each with a different approach. To obtain a better understanding of the differences

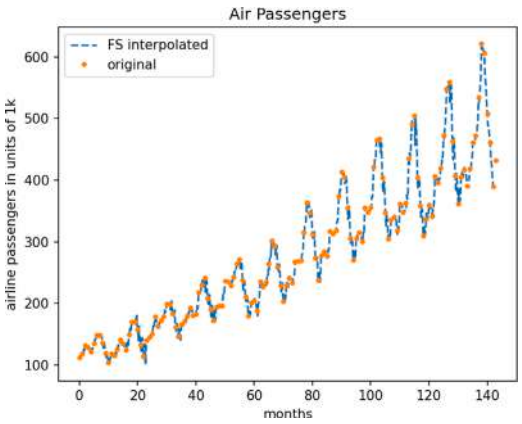


Fig. 24. Air Passengers data set, FS.

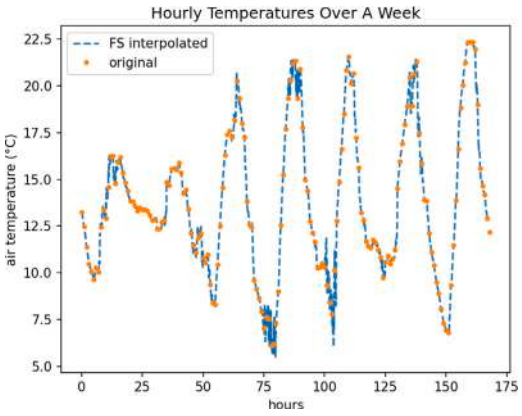


Fig. 25. Weather data set, FS.

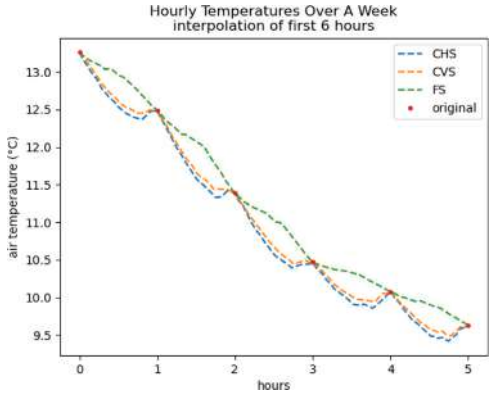


Fig. 26. First 6 points.

between the three methods, we depict in Figs. 26 and 27 the interpolation results for our Weather data set provided by the three strategies on the same graphic.

We can observe that CHS (with $s_i \in [0, 0.2]$) and CVS approaches provide similar results, while the FS approach determines slightly higher variations.

To emphasize the comparison, let us use the original Weather data set. We extract hourly data and use the three strategies for fractal interpolation with a number of five interpolation points (*n_interpolation* = 5) to simulate 10-min data.

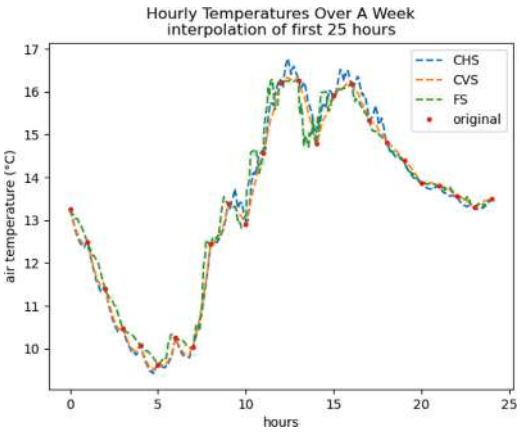


Fig. 27. First 25 points.

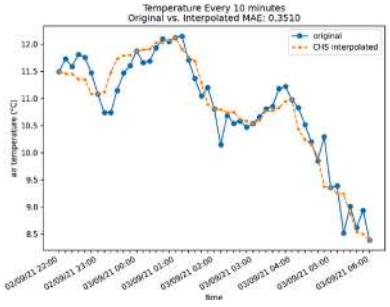


Fig. 28. MAE, CHS.

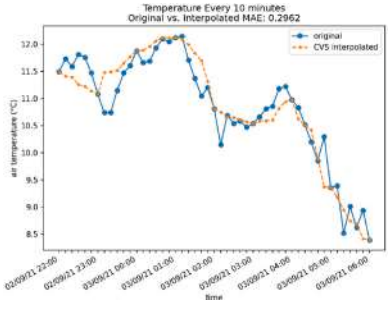


Fig. 29. MAE, CVS.

In Figs. 28–30 there are presented the results for data recorded between 02/09/21, 22:00 and 03/09/21, 06:00 compared to the original data for all three strategies. To obtain a better understanding of the differences between the three strategies, we computed the Mean Absolute Error (MAE) for each data set, which provides us with the mean difference, in degrees, between the real and the interpolated data.

As regards the Weather data set, for CHS we obtain $MAE = 0.3510$, for CVS we have $MAE = 0.2962$ and for FS we get $MAE = 0.4997$. Thus, we can observe that the least MAE is obtained for CVS, thus, this strategy is the optimal one, followed by CHS and FS.

3.1.2. Normalization step

Usually, the normalization step's objective is to convert an attribute's values to a better range. There are more strategies to normalize data. We chose to scale the data using a method similar to the *min–max* method, which performs a linear transform of the data in a given interval.

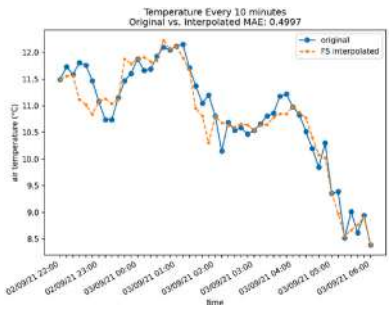


Fig. 30. MAE, FS.

More precisely, to normalize data $x' \in [0, 1]$ we use the formula

$$x' = \frac{x - \min x}{\max x - \min x}.$$

For data $x'' \in [-1, 1]$ the formula becomes

$$x'' = 2\frac{x - \min x}{\max x - \min x} - 1. \tag{4}$$

In general, for data $x''' \in [a, b]$, where $a, b \in \mathbb{R}$ the formula becomes

$$x''' = (b - a)\frac{x - \min x}{\max x - \min x} + a.$$

The purpose of normalization is to transform data in a way that they are either dimensionless and/or have similar distributions. Normalization is an essential step in data preprocessing in any ML algorithm and model fitting. Propagated errors must not have high values, especially in the case of recurrent neural networks (as is LSTM). Moreover, data normalization allows comparison of the results over data with a different configuration, thus reducing biased prioritization of some features over others, which can be caused by providing data with different features that have wide-scale differences to the model.

3.1.3. Stationarity in time series analysis

Stationarity is an important concept in the field of time series analysis with tremendous influence on how the data is perceived and predicted. It indicates whether statistical properties such as mean, variance, and autocorrelation of a time series change over time.

To determine whether a data set must be transformed, we use the Augmented Dicky-Fuller test which uses the coefficient that defines the unit root, p-value. If the p-value obtained is below 0.05, then the current data set is stationary.

For maintaining data stationarity, several transformations can be applied to eliminate trends and seasonality of a data set. In Table 2 there are presented the p-values obtained after the three types of transformations used for eliminating the trend: Log Transformation, Square Root Transformation and Linear Regression Transformation.

Since for the hourly temperatures over a week, the data set is already stationary and no transformation was required as can be seen in Table 2, we manually created a data set composed of the daily maximum temperature recorded.

Daily temperature extremes are more informative than the mean value, indicating the range over this time interval, which is highly important for interpreting its decisive influence on various processes. The minimum temperature generally occurs at dawn, at the end of the negative net radiation interval, when atmospheric status in the boundary layer is rather stable. The situation is completely different as concerns the maximum daily temperature, which arises at midday, after the heating peak, which also induces a considerable enhancement of thermal turbulence, causing increased air temperature variability. Consequently, the daily maximum temperature was chosen for this study, considering that fractal interpolation and a machine learning approach could significantly improve its assessment.

As a result of the different dimensions of the initial and interpolated data set, the p-values obtained are different. For comparison, we also perform linear interpolation. We can notice from Table 2 that regardless of the interpolation strategy employed, the p-values are close to the value obtained for linear interpolation.

We highlight in Table 2 the transforms that are maintained for each strategy.

As seasonality is regarded, a frequent step is the differentiation of data. However, since fractal interpolation produces functions that are continuous, but are not necessarily everywhere differentiable, this step cannot be considered.

Moreover, the LSTM model has a sufficient complexity to process non-stationary data, as is not the case for other statistic models like Autoregressive Integrated Moving Average (ARIMA) which depends closely on data stationarity. For the LSTM model, data normalization is more significant.

3.2. Data splitting step

For all datasets, 70% of them are retained, while the remaining 30% are allocated to the test data set, in chronological order.

For the Weather data set, the division of the data set is presented in Table 3.

The current division presented in Table 3 is relevant for the Weather data set and does not alter the predictions as in the Brasov mountain region, also including the neighbouring depressionary area, the three autumn months (September, October, November) are generally characterized by a more stable weather regime as compared with the symmetrical springtime interval. There are some sudden cold intervals, especially at the end of September or the beginning of October, but after this more dynamic period, the weather pattern shows prolonged intervals of fine weather, favourable for mountain tourism, which is also encountered towards the end of November in many years.

3.3. Model description

The selected prediction model uses an LSTM layer, followed by one dense layer with dimension 1 since the datasets have only one feature (see Fig. 31).

For each data set, the number of hidden layers in LSTM was optimized using *Optuna*. For the Weather data set, the obtained values are presented in Table 4.

LSTM is configured with the default parameters. The metric used for measuring the loss in the training step is Mean Squared Error (MSE) and for the general evaluation of the model we used RMSE.

3.3.1. Specific input structure of LSTM

To be able to feed the data to the LSTM layer, we must transform the data set into a supervised learning format. This is achieved by sliding a window of size *input_data_points* over the whole data set using a step of 1. The obtained subsets will represent the inputs for the network. In the case of Univariate Time Series prediction, the output for subset i $[data_i, data_{\text{i+input\_data\_points-1}}]$ will be $data_{\text{i+input\_data\_points}}$.

The LSTM layer implementation in *Keras - Tensorflow Keras 2.4.1*, expects the input data to be in the format of [*batch_size*, *input_data_points*, *features*]. In our case, we used a batch size of 1.

Special attention is needed such that the window size *input_data_points* considered must be less than 30% of the entire dataset. Otherwise, using the testing set to evaluate the model's performance will not be possible, as for a single data point prediction, *input_data_points* entries are required in the supervised format.

Table 2
p-values obtained for the performed transformations.

Data set	Interpolation strategy	None	Log	Square	Linear regression
Shampoo Sales	None	1.0000	0.9983	0.9991	**0.0000**
	Linear	0.7860	0.8076	0.7829	0.0655
	CHS	0.9655	0.8245	0.9221	**0.1022**
	CVS	0.9755	0.9353	0.9678	**0.0839**
	FS	0.8744	0.0423	0.7658	**0.0054**
Air Passengers	None	0.9919	**0.4224**	0.9181	0.4415
	Linear	0.0934	0.2104	0.1522	0.0001
	CHS	0.1250	0.2869	0.2118	**0.0002**
	CVS	0.1334	0.2248	0.1919	**0.0001**
	FS	0.0919	0.2018	0.1442	**0.0001**
Wheat	None	0.0607	**0.0018**	0.0122	0.9983
	Linear	0.4275	0.3043	0.3704	0.8529
	CHS	0.4840	**0.2979**	0.3924	0.9303
	CVS	0.4997	**0.3335**	0.4207	0.9110
	FS	0.6144	**0.4094**	0.5235	0.8590
Maize	None	0.2370	**0.0207**	0.1073	0.9986
	Linear	0.4905	0.1421	0.3132	0.9008
	CHS	0.4047	**0.0497**	0.1863	0.9751
	CVS	0.4196	**0.0820**	0.2131	0.9798
	FS	0.4110	**0.0965**	0.2446	0.9493
Hourly Temperatures Over a Week	None	**0.0000**	–	–	–
	Linear	0.0000	–	–	–
	CHS	**0.0000**	–	–	–
	CVS	**0.0002**	–	–	–
	FS	**0.0000**	–	–	–
Max Daily Temperature	None	0.1210	0.8297	**0.1160**	0.8669
	Linear	0.0343	0.0772	0.0510	0.4679
	CHS	**0.1374**	0.1942	0.1504	0.7768
	CVS	0.2860	**0.2697**	0.3451	0.8406
	FS	0.0714	**0.0453**	0.0655	0.5549

Table 3
Splitting of the weather data set.

Data set	Train data	Test data
Hourly Temperature	01/09/21, 00:00–05/09/21, 21:00	05/09/21, 21:00–08/09/21, 00:00
Daily Maximum Temperature	01/09/21–02/11/21	02/11/21–30/11/21

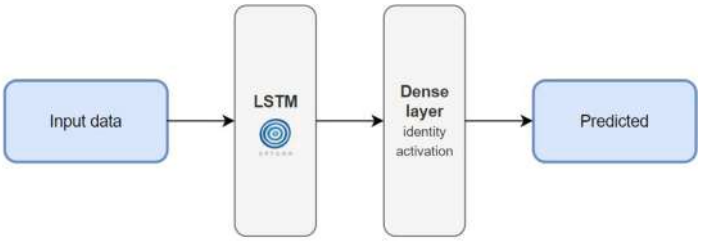


Fig. 31. Neural network structure.

Table 4
Number of hidden layers.

Interpolation strategy	Hidden layers
None	48
CHS	39
CVS	28
FS	10

3.3.2. Model optimization

Optimization plays an important role in a machine learning solution and the step of tuning a chosen model is critical to the model's performance and accuracy. Hyperparameter tuning intends to find the hyperparameters of a given machine learning algorithm that guarantee the best performance as measured on a validation set. Saving time, but also eliminating the chance of overfitting or underfitting influenced the interest in hyperparameter tuning research. Thus, entire branches of machine learning and deep learning theory have been dedicated to the optimization of models.

In our study, we use *Optuna - Optuna 2.10.0* in two key points of our implementation:

1. optimizing the vertical scaling factor s_i for the interpolation step;
2. fine-tuning LSTM network hyperparameters.

For the LSTM network model, we considered the following hyperparameters for optimization:

- *units*: the number of hidden layers used for the LSTM layer, which defines the complexity of the model. There is no formula or rule to determine this number, so it is a natural decision to pick this hyperparameter for the optimization process search space. Our choice is the interval $[2, 64]$;
- *input_data_points* (timesteps): given the data structure expected by the LSTM network, this represents the number of consecutive

Table 5
Model tuning and hyperparameters optimization.

Data set	Linear regression	Hidden layers	Input data points	Epochs	Learning rate	Train RMSE	Test RMSE
Shampoo Sales	None	62	8	80	0.02	0.1717	0.4997
	CHS	48	16	15	0.04	0.0859	0.0951
	CVS	41	28	10	0.01	0.0607	**0.0749**
	FS	17	1	8	0.005	0.1218	0.1264
Air Passengers	None	56	14	65	0.007	0.0947	0.1348
	CHS	33	97	8	0.03	0.0285	0.0655
	CVS	62	51	5	0.02	0.0288	**0.0444**
	FS	29	93	7	0.03	0.0416	0.0619
Wheat	None	13	5	125	0.005	0.1480	0.2410
	CHS	8	93	12	0.03	0.0559	0.0653
	CVS	11	98	10	0.01	0.0431	**0.0552**
	FS	51	99	12	0.01	0.0656	0.0932
Maize	None	52	3	150	0.05	0.1327	0.1766
	CHS	32	82	15	0.01	0.0299	0.0304
	CVS	39	1	10	0.002	0.0187	**0.0273**
	FS	27	20	20	0.007	0.0920	0.0991
Hourly Temperatures Over a Week	None	48	8	60	0.008	0.1111	0.1383
	CHS	39	22	7	0.005	0.0325	0.0414
	CVS	28	34	15	0.001	0.3446	**0.0389**
	FS	10	17	14	0.005	0.0391	0.0462
Daily Max Temperatures	None	43	5	175	0.02	0.1132	0.4944
	CHS	47	84	12	0.03	0.0321	**0.0523**
	CVS	49	11	20	0.005	0.0290	0.0692
	FS	50	27	15	0.005	0.0526	0.1351

input values considered for an output value in the supervised learning format. Since this is highly dependent on the data set size, but also on the frequency of the desired prediction, we considered the interval $[1, min(x, 30 * \text{length(dataset)}/100 - 1)]$, where $x = 15$ for non-interpolated datasets and $x = 100$ for the interpolated ones. This limit was necessary for the optimization process because a large window size produces fewer predictions which misleadingly leads to smaller cumulated errors, however, the purpose of the model is to come up with predictions relying on as few known values as possible.

- *learning_rate*: finding the right value for the learning rate can significantly improve the accuracy of the model. A learning rate too big might lead to poor performance since the algorithm makes leaps too large while searching for the optimal value, whereas a small learning rate slows down the execution time of the training process. For the search space, we chose the interval $[1e-3, 1e-1]$.
- *epochs*: although not specific to Recurrent neural networks (RNNs), the number of epochs to train a model is an important key factor. The more epochs, the better for the model, but too many epochs can often lead to overfitting. Each model corresponding to a non-interpolated data set was trained for 150 epochs, while the interpolated ones were limited to 25. The final epoch number was hand-picked by observing the evolution of the loss at each epoch

As described in Section 2.2.2, the *suggest* APIs from *Optuna* framework are used to define the search space for each selected hyperparameter. Each study ran 50 trials since no significant improvement in the overall score was observed after that.

The objective function defined will create, compile, fit and evaluate the model using the training data set 5 times for each set of considered hyperparameters. This is done to ensure that the optimized hyperparameters produce a more stable model. After the optimization, we obtained the configurations from Table 5 with the corresponding scores. Mean results are again computed for 5 individual runs using the same configuration.

We mention the fact that all optimizations were executed using the free environment offered by Google Colab, having a GPU-accelerated runtime, making this accessible to everybody.

We notice that for the majority of the datasets, the best results are obtained via CVS. However, the other two strategies are also of considerable importance. For all datasets studied, the values obtained for CHS and CVS are rather close, while the results obtained for FS are higher. Even so, FS is a viable strategy for its easy way of usage. Moreover, it is worth noticing that all results obtained with the three strategies are at least 50% better than the results obtained without any interpolation strategy.

4. Results and discussions

We present the graphics of the predictions obtained via the LSTM model for our Weather data set, for the three proposed strategies and the initial data which is not interpolated. The results emphasize the advantages brought to the predictions.

For the manufactured data set which contains the daily maximum temperatures, the results are presented in Figs. 33–35. It can be observed that for the initial data, the LSTM does not perform well enough. However, for the interpolated datasets, the results are visibly better, with the best result being obtained for CHS. All the strategies provide better results and this is motivated by the fact that more training data provides better ML solutions. Thus, the importance of the proposed preprocessing strategies is supported and emphasized.

The results for both the maximum daily temperature (considered for example purposes) and the hourly data set with 17 interpolation points, see Figs. 37–39, prove that fractal interpolation brings a significant upgrade to the model as it improves visibly the results of the predictions when compared to the results obtained for the initial datasets (results that are presented in Figs. 32 and 36). Even though 17 interpolation points provide good theoretical results, we should also practically test the predictions by considering the hourly data set with 5 interpolation points to simulate the 10-min data recorded by the sensor.

Thus, we consider the entries from the Weather data set from 01/09/21, 00:00 to 08/09/21, 00:00. For this data set, we extract the hourly temperatures data and use fractal interpolation with *n_interpolation* = 5 according to the three strategies proposed (CHS, CVS, FS) to simulate 10-min data.

We can observe from Table 6 that while the hourly data without any interpolation provides visibly worse results (as expected) than the predictions with initial 10-min data, the case is the opposite when interpolating the hourly data set to simulate 10-min data with either of the three strategies. Thus, it can be observed that extracting hourly data

Table 6
Comparison between interpolated 10-min data and original data predictions.

Data set	Linear regression	Hidden layers	Input data points	Epochs	Learning rate	Train RMSE	Test RMSE
Hourly Temperatures Over a Week	None	48	8	60	0.008	0.1111	0.1383
	CHS	30	94	15	0.03	0.0415	**0.0462**
	CVS	61	94	10	0.02	0.0443	0.0474
	FS	34	99	17	0.016	0.0488	0.0494
Temperature Every 10 min	None	55	32	25	0.016	0.0523	0.0541

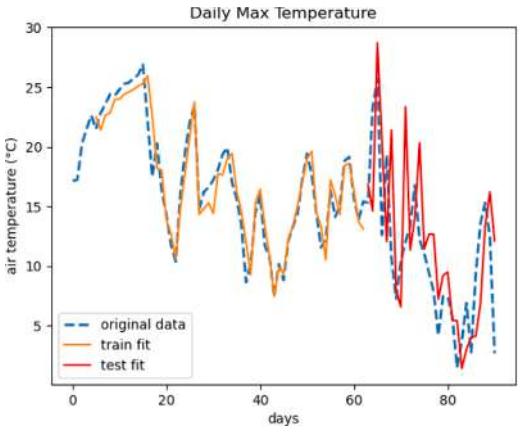


Fig. 32. Prediction for daily maximum data without interpolation.

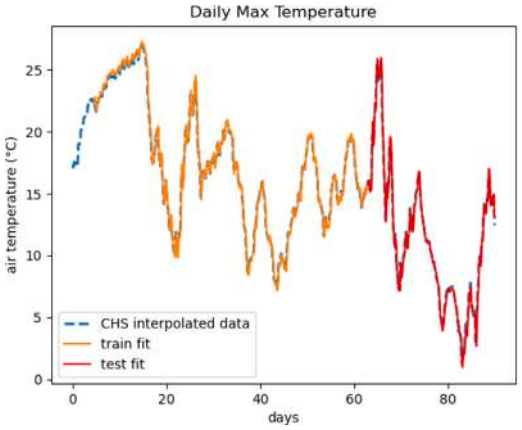


Fig. 33. Prediction for daily maximum data with CHS.

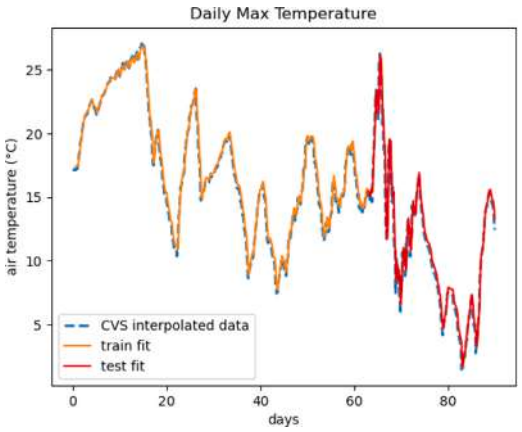


Fig. 34. Prediction for daily maximum data with CVS.

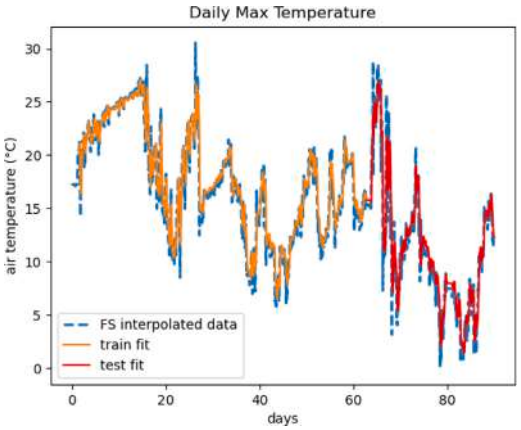


Fig. 35. Prediction for daily maximum data with FS.

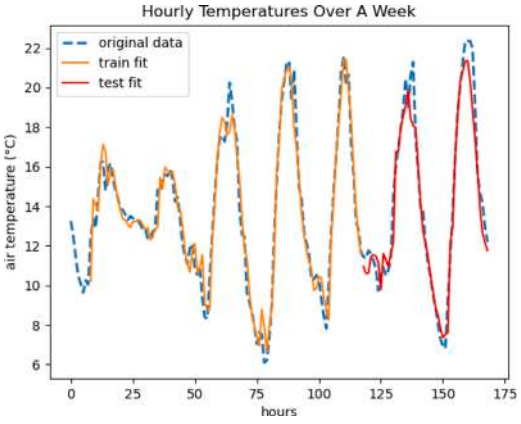


Fig. 36. Prediction for hourly data without interpolation.

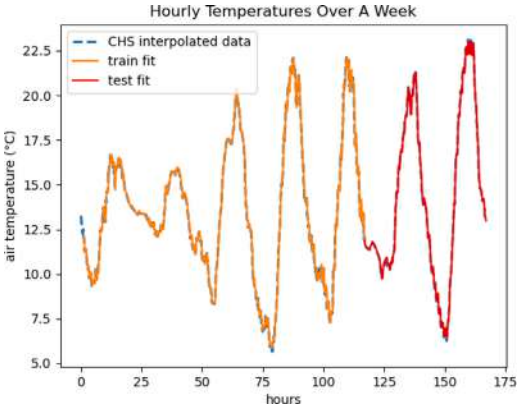


Fig. 37. Prediction for hourly data without CHS.

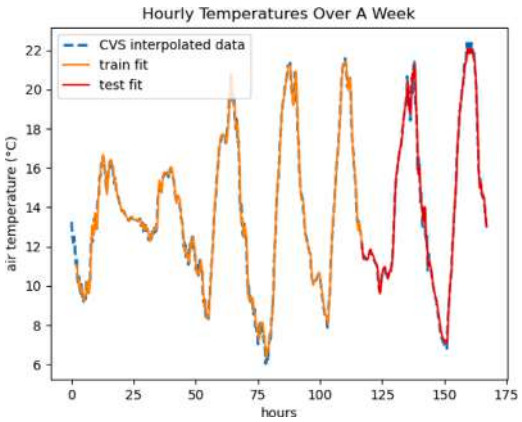


Fig. 38. Prediction for hourly data with CVS.

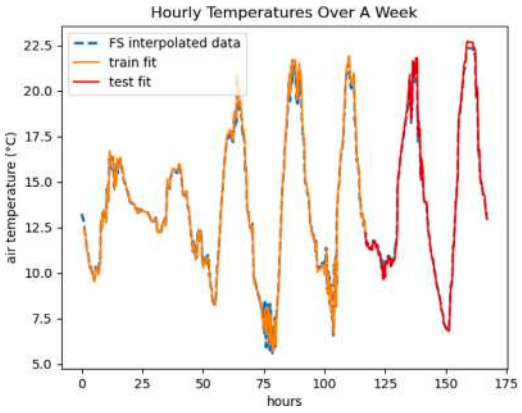


Fig. 39. Prediction for hourly data with FS.

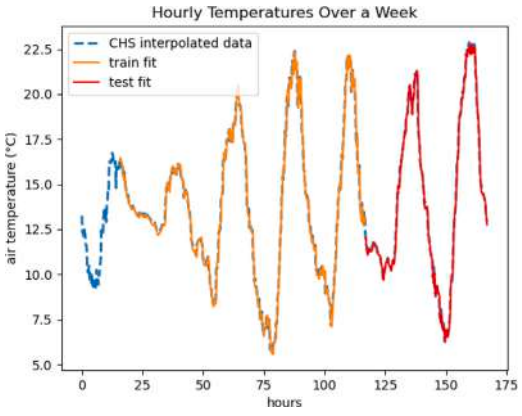


Fig. 40. Prediction for hourly data set interpolated with CHS, n_interpolation = 5.

and constructing artificial fractal interpolation points for 10-min data is a better strategy for prediction. The result is surprising and requires further testing on various weather datasets, but the current results obtained for the considered data set and the consistent differences between the test RMSE results, see Table 6, allow us to be optimistic about the performances of our strategies on real-world data (especially meteorological datasets) (see Figs. 40 and 41).

Artificial Neural Networks (ANN) and machine learning have proven to be efficient tools for predicting meteorological data. These could be significant for predicting data for smaller local areas since some meteorological processes are too small-scale or too complex to be explicitly included in classical numerical weather prediction models.

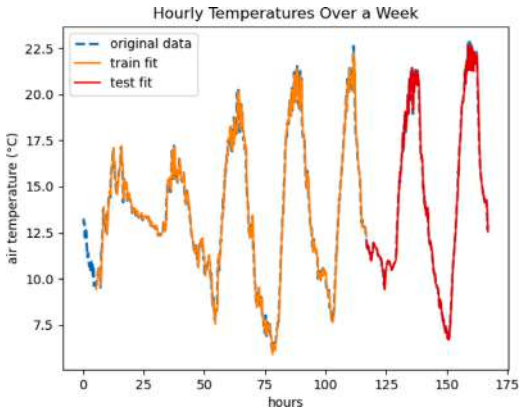


Fig. 41. Prediction using the original 10-min entry data set.

Thus, our study proposes a recent idea of using fractal interpolation tools for preprocessing data before feeding the data to an ML algorithm, in our case, an LSTM algorithm.

Without any doubt, this approach has some limitations, and there are opportunities to improve this study.

In order to provide a comprehensive coverage of the interpolation step, the study proposes three techniques. The persistence of certain aspects of the data is not guaranteed by the stop condition from the CHS method, although it assures that the Hurst exponent for the interpolated data is sufficiently near to the original Hurst value. This inspired us to develop novel interpolation techniques guaranteeing the preservation of specific data features. The final approach, namely FS, similarly focuses on maintaining these characteristics while also improving in terms of time complexity.

Furthermore, future experiments could be completed in order to assess the performance of the proposed strategies in relation to outliers, but also further testing on various weather datasets from different regions needs to be performed. In addition, although the LSTM modelling obtained is sufficient for the selected datasets, a more complex model can be developed to achieve better accuracy results.

5. Conclusions

The results obtained in this study confirm the relevance and extend the applications of fractal interpolation as a time series augmentation technique. The three proposed strategies involve optimizing the vertical scaling factor and the size of the fractal interpolation subset to generate relevant data in the context of the prediction optimization problem. Depending on the source domain and data pattern, we were able to identify the appropriate interpolation strategy in order to improve predictions. As a result, for all considered datasets, the current approach improved accuracy prediction results between 50% and 89% over the base case where the raw data was used.

By using the meteorological dataset of temperatures recorded in Braşov, we were able to show that the three proposed strategies can also be used independently of a prediction model in order to obtain data simulating a higher sampling rate than the maximum capacity of the sensor, with an average error of maximum ± 0.49. Numerical weather prediction models such as those described in Malardel (2019), based on fluid dynamics and thermodynamic equations, could also benefit from this data enrichment. Although they do not outperform solutions based on artificial neural networks, these models can perform well for short time intervals (up to 5 days) when sufficient data are available.

We have highlighted the need to use machine learning modelling in this study. In addition, it is possible to go even further with the results obtained so that a relevant prediction on meteorological data also implies a focus on process optimization in precision agriculture,

early detection of extreme weather phenomena, and local and global environmental understanding.

Our outcomes extend the current results existing in the literature and contribute significantly to research dedicated to data augmentation and data preprocessing, as well as enhancing machine learning prediction models. Moreover, our results provide a significant answer to the question of refining data prediction based on data recorded at larger intervals.

CRediT authorship contribution statement

Alexandra Băicoianu: Conceptualization, Methodology, Software, Validation, Formal analysis, Writing – original draft, Writing – review & editing. **Cristina Gabriela Gavrilă:** Conceptualization, Methodology, Software, Validation, Formal analysis, Visualization, Writing – original draft, Writing – review & editing. **Cristina Maria Păcurar:** Conceptualization, Methodology, Validation, Writing - Original Draft,Writing – review & editing, Visualization. **Victor Dan Păcurar:** Conceptualization, Validation, Resources, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

Bajahzar, A., Guedri, H., 2019. Reconstruction of fingerprint shape using fractal interpolation. Int. J. Adv. Comput. Sci. Appl. 10 (5), http://dx.doi.org/10.14569/IJACSA.2019.0100514.

Bandara, K., Hewamalage, H., Liu, Y., Kang, Y., Bergmeir, C., 2021. Improving the accuracy of global forecasting models using time series data augmentation. Pattern Recognit. 120, http://dx.doi.org/10.1016/j.patcog.2021.108148.

Barnsley, M., 2012. Fractals Everywhere, third ed. Dover Publications.

Bélisle, E., Huang, Z., Le Digabel, S., Gheribi, A.E., 2015. Evaluation of machine learning interpolation techniques for prediction of physical properties. Comput. Mater. Sci. 98, 170–177. http://dx.doi.org/10.1016/j.commatsci.2014.10.032, URL: https://www.sciencedirect.com/science/article/pii/S092702561400706X.

Bergmeir, C., Hyndman, R.J., Benítez, J.M., 2016. Bagging exponential smoothing methods using STL decomposition and Box–Cox transformation. Int. J. Forecast. 32 (2), 303–312. http://dx.doi.org/10.1016/j.ijforecast.2015, URL: https://ideas.repec.org/a/eee/intfor/v32y2016i2p303-312.html.

Bouboulis, P., Dalla, L., Drakopulos, V., 2006. Image compression using recurrent bivariate fractal interpolation surfaces. Int. J. Bifurcation Chaos 16 (07), 2063–2071. http://dx.doi.org/10.1142/S0218127406015908, arXiv:https://doi.org/10.1142/S0218127406015908.

Chai, X., Tang, G., Wang, S., Lin, K., Peng, R., 2021. Deep learning for irregularly and regularly missing 3-D data reconstruction. IEEE Trans. Geosci. Remote Sens. 59, 6244–6265, URL: https://api.semanticscholar.org/CorpusID:261340870.

Chen, C.-J., Cheng, S.-C., Huang, Y.M., 2011. The reconstruction of satellite images based on fractal interpolation. Fractals 19 (03), 347–354. http://dx.doi.org/10.1142/S0218348X11005385, arXiv:https://doi.org/10.1142/S0218348X11005385.

FaostatDocs, 2022. Food and agriculture data. URL: http://www.fao.org/faostat/. (accessed 7 March 2022).

Fawaz, H.I., Forestier, G., Weber, J., Idoumghar, L., Muller, P.A., 2018. Data augmentation using synthetic data for time series classification with deep residual networks. In: AALTD'18 Workshop in ECML/PKDD.

Forestier, G., Petitjean, F., Dau, H.A., Webb, G.I., Keogh, E., 2017. Generating synthetic time series to augment sparse datasets. In: 2017 IEEE International Conference on Data Mining. ICDM, pp. 865–870. http://dx.doi.org/10.1109/ICDM.2017.106.

Garcia, S., Luengo, J., Herrera, F., 2014. Data Preprocessing in Data Mining. In: Intelligent Systems Reference Library, Springer.

Gowrisankar, A., Priyanka, T.M.C., Banerjee, S., 2022. Omicron: a mysterious variant of concern. Eur. Phys. J. Plus 137, 100.

Guennec, A.L., Malinowski, S., Tavenard, R., 2016. Data augmentation for time series classification using convolutional neural networks. URL: https://api.semanticscholar.org/CorpusID:3907864.

Hutchinson, J., 1981. Fractals and self-similarity. Indiana Univ. Math. J. 30, 713–747.

Iwana, B.K., Uchida, S., 2021. An empirical survey of data augmentation for time series classification with neural networks. PLOS ONE 16 (7), 1–32. http://dx.doi.org/10.1371/journal.pone.0254841.

Jia, Y., Ma, J., 2017. What can machine learning do for seismic data processing? An interpolation application. Geophysics 82, URL: https://api.semanticscholar.org/CorpusID:67218941.

Kaggle, 2022. High-quality public datasets. URL: https://www.kaggle.com/. (accessed 7 March 2022).

Kamycki, K., Kapuscinski, T., Oszust, M., 2020. Data augmentation with suboptimal warping for time-series classification. Sensors 20 (1), http://dx.doi.org/10.3390/s20010098, URL: https://www.mdpi.com/1424-8220/20/1/98.

Kang, Y., Hyndman, R.J., Li, F., 2019. GRATIS: GeneRAting TIme Series with diverse and controllable characteristics. Stat. Anal. Data Min.: ASA Data Sci. J. 13, 354–376, URL: https://api.semanticscholar.org/CorpusID:71146933.

Lee, S.W., Kim, H.Y., 2020. Stock market forecasting with super-high dimensional time-series data using ConvLSTM, trend sampling, and specialized data augmentation. Expert Syst. Appl. 161, 113704, URL: https://api.semanticscholar.org/CorpusID:225041816.

Malardel, S., 2019. Weather forecasting models. URL: https://www.encyclopedie-environnement.org/en/air-en/weather-forecasting-models/. (accessed 19 April 2022).

Manousopoulos, P., Drakopoulos, V., Theoharis, T., 2008. Curve fitting by fractal interpolation. In: Gavrilova, M.L., Tan, C.J.K. (Eds.), Transactions on Computational Science I. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 85–103. http://dx.doi.org/10.1007/978-3-540-79299-4_4.

Manousopoulos, P., Drakopoulos, V., Theoharis, T., 2011. Parameter identification of 1D recurrent fractal interpolation functions with applications to imaging and signal processing. J. Math. Imaging Vision 40 (2).

May, M., 1996. Fractal image compression. Am. Sci. 84 (5).

Mazel, D., Hayes, M., 1992. Using iterated function systems to model discrete sequences. IEEE Trans. Signal Process. 40 (7), 1724–1734. http://dx.doi.org/10.1109/78.143444.

Meijering, E., 2002. A chronology of interpolation: from ancient astronomy to modern signal and image processing. Proc. IEEE 90 (3), 319–342. http://dx.doi.org/10.1109/5.993400.

Navascués, M., 2010. Reconstruction of sampled signals with fractal functions. Acta Appl. Math. 110.

Ni, L.-P., Ni, Z.-W., Gao, Y.-Z., 2011. Stock trend prediction based on fractal feature selection and support vector machine. Expert Syst. Appl. 38 (5), 5569–5576. http://dx.doi.org/10.1016/j.eswa.2010.10.079, URL: https://www.sciencedirect.com/science/article/pii/S0957417410012236.

Oh, C., Han, S., Jeong, J., 2020. Time-series data augmentation based on interpolation. In: FNC/MobiSPC. URL: https://api.semanticscholar.org/CorpusID:222112073.

Optuna-ReadTheDocs, 2023a. Optuna samplers - TPESampler docs. URL: https://optuna.readthedocs.io/en/stable/reference/samplers/generated/optuna.samplers.TPESampler.html#optuna.samplers.TPESampler. (accessed 2 October 2023).

Optuna-ReadTheDocs, 2023b. Optuna samplers docs. URL: https://optuna.readthedocs.io/en/stable/reference/samplers/index.html. (accessed 2 October 2023).

Păcurar, C.-M., Necula, B.-R., 2020. An analysis of COVID-19 spread based on fractal interpolation and fractal dimension. Chaos Solitons Fractals 139, 110073. http://dx.doi.org/10.1016/j.chaos.2020.110073, URL: https://www.sciencedirect.com/science/article/pii/S0960077920304707.

Raubitzek, S., Neubauer, T., 2021. A fractal interpolation approach to improve neural network predictions for difficult time series data. Expert Syst. Appl. 169, 114474. http://dx.doi.org/10.1016/j.eswa.2020.114474, URL: https://www.sciencedirect.com/science/article/pii/S0957417420311234.

Wang, H.-Y., Li, H., Shen, J.-Y., 2019. A novel hybrid fractal interpolation-Svm model for forecasting stock price indexes. Fractals 27 (4), http://dx.doi.org/10.1142/S0218348X19500555, 1950055.

Wen, Q., Sun, L., Yang, F., Song, X., Gao, J., Wang, X., Xu, H., 2021. Time series data augmentation for deep learning: A survey. In: 30th International Joint Conference on Artificial Intelligence. IJCAI 2021.

Wu, Y., et al., 2020. Using linear interpolation to reduce the training samples for regression based visible light positioning system. IEEE Photonics J. 12 (2), 1–5.

Yadav, O.P., Ray, S., 2021. A novel method of preprocessing and modeling ECG signals with Lagrange–Chebyshev interpolating polynomials. Int. J. Syst. Assur. Eng. Manag. 12 (3), 377–390. http://dx.doi.org/10.1007/s13198-021-01077-, URL: https://ideas.repec.org/a/spr/ijsaem/v12y2021i3d10.1007_s13198-021-01077-z.html.

Yakuwa, F., Dote, Y., Yoneyama, M., Uzurabashi, S., 2003. Novel time series analysis and prediction of stock trading using fractal theory and time delayed neural network. In: SMC'03 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme - System Security and Assurance (Cat. No.03CH37483). Vol. 1, pp. 134–141. http://dx.doi.org/10.1109/ICSMC.2003.1243804.

Zhai, M.-Y., Fernández-Martínez, J.L., Rector, J.W., 2011. A new fractal interpolation algorithm and its applications to self-affine signal reconstruction. Fractals 19 (03), 355–365. http://dx.doi.org/10.1142/S0218348X11005427, arXiv:https://doi.org/10.1142/S0218348X11005427.

Zhang, Y., Fan, Q., Bao, F., Liu, Y., Zhang, C., 2018. Single-image super-resolution based on rational fractal interpolation. IEEE Trans. Image Process. 27 (8), 3782–3797. http://dx.doi.org/10.1109/TIP.2018.2826139.

Multisource Remote Sensing Data Visualization Using Machine Learning

Ioana Cristina Plajer, Alexandra Băicoianu, Luciana Majercsik, and Mihai Ivanovici, *Member, IEEE*

***Abstract*— With the availability of several remotely sensed data sources, the problem of efficiently visualizing the information from multisource data for improved Earth observation becomes an intriguing and challenging subject. Multispectral (MS) and hyperspectral (HS) images encompass a wealth of spectral data that standard RGB monitors cannot replicate directly. Thus, it is important to elaborate methods for accurately representing this information on conventional displays. These images, with tens to hundreds of spectral bands, contain relevant data about specific wavelengths that RGB channels cannot capture. Traditional visualization methods often use only a limited amount of the available spectral information, resulting in a significant loss of information. However, recent advances in artificial intelligence models have provided superior visualization techniques. These artificial Intelligence (AI)-based methods allow for more realistic and visually appealing representations, which are important for the information interpretation and direct identification of areas of interest. The main goal of our study is to process aggregated datasets from various sources using a fully connected neural network (FCNN), while considering visualization as a secondary objective. Given that our data come from a variety of sources, a significant emphasis in our study was placed on the preprocessing stage. In order to achieve a consistent visualization across datasets from different sources, proper preprocessing by standardization or normalization procedures is essential. Our research comprises numerous experiments to demonstrate the effectiveness of the proposed technique for image visualization.**

***Index Terms*— Multisource, multispectral (MS) and hyperspectral (HS) images, neural network, normalization, remote sensing, standardization, visualization.**

I. Introduction

THE unique spectral properties of different types of materials determine variations in the way they absorb light, thus giving each material a distinct spectral fingerprint. Multispectral (MS) and hyperspectral (HS) sensors are capable of capturing a rich spectral information that can cover hundreds of spectral bands, allowing them to detect even the smallest changes in the reflectance or radiance of objects. The resulting images produced by MS and HS sensors include additional spectral information about the chemical composition of objects. This additional information makes them extremely useful in a series of applications in areas, such as agriculture [1], [2], forestry [3], [4], environmental monitoring and ecology [5], [6], object detection [7], land cover classification [8], [9], as well as military and industrial fields.

In this article, machine learning techniques allow for notable progress in a large range of applications of remote sensing. Examples include a Siamese Transformer Network designed for HS image target detection [10] and some techniques for HS image denoising and anomaly detection [11], [12], [13], [14]. These advances highlight the growing ability of machine learning strategies to refine the understanding and interpretation of remote sensing data.

The first step in MS and HS interpretation is the visualization of the data in a comprehensive manner for human users. MS and HS images contain more spectral information than standard RGB channels can display, providing important data about specific wavelengths, beyond the RGB capabilities.

A realistically displayed spectral satellite image enables direct human interpretation and identification of areas of interest, but accurate visualization is challenging because of the need to compress information from numerous bands into three while preserving essential spatial and spectral detail.

Several visualization techniques have been proposed in the scientific literature in order to generate realistic RGB images from spectral images. These include band selection [15], [16], independent component analysis (ICA) [17], and principal component analysis (PCA) [18], [19] based methods, linear and nonlinear methods [20], and, relatively recent, machine learning approaches [21], [22].

While the classical method based on band selection uses a very small part of the available information, disregarding the information available in the other unused bands, better results of visualization with artificial intelligence (AI) models have already been reported in recent research papers [23]. Such methods, like the proposed one, allow for a more realistic rendering of colors and, at the same time, producing more appealing images (vivid colors, good contrast, and large dynamic range) while showing the color as close to the real colors of the scene, as they would have been perceived by a human observer.

Manuscript received 23 January 2024; accepted 25 February 2024. Date of publication 4 March 2024; date of current version 15 March 2024. This work was supported by the Romanian Excellence Center on Artificial Intelligence on Earth Observation Data for Agriculture (AI4AGRI) Project titled "Romanian Excellence Center on Artificial Intelligence on Earth Observation Data for Agriculture" funded by European Union's Horizon Europe Research and Innovation Program under Grant 101079136. *(Corresponding author: Ioana Cristina Plajer.)*

Ioana Cristina Plajer, Alexandra Băicoianu, and Luciana Majercsik are with the Department of Mathematics and Computer Science, Faculty of Mathematics and Computer Science, Transilvania University of Braşov, 500024 Braşov, Romania (e-mail: ioana.plajer@unitbv.ro; a.baicoianu@unitbv.ro; luciana.carabaneanu@unitbv.ro).

Mihai Ivanovici is with the Department of Electronics and Computers, Faculty of Electrical Engineering and Computer Science, Transilvania University of Braşov, 500024 Braşov, Romania (e-mail: mihai.ivanovici@unitbv.ro).

Digital Object Identifier 10.1109/TGRS.2024.3372639

An accurate visualization of satellite images can facilitate a direct human interpretation for spotting pixels or areas affected by various degradation (e.g., in agriculture, spotting areas affected by diseases of the agricultural crops). Moreover, satellite images rendered in a more realistic manner can be used as a first step to verify the accuracy of different approaches, such as classification, which can be initially appreciated visually.

A major challenge in addressing different tasks on MS and HS images is the variability of the image data (spectral range and spatial resolution) when obtained by different sensors. Thus the primary goal of our research, presented in this paper, is a method of processing such aggregated datasets, obtained from various sources, using a fully connected neural network (FCNN). Visualization is merely a secondary goal, as a test case specifically for this fusion.

Although research in the area of visualization is still limited, as already described in [24] and [23], a neural network trained on a set of MS images can be used successfully to visualize spectral images acquired by different sensors. Challenges in this field include limited labeled training data and spectral variability between different sensors, as they capture information at different spectral wavelengths. The spectral variability can make it difficult for a neural network to extract significant and meaningful features in a coherent manner, especially if the training data does not cover the full range of spectral variation.

This article explores different approaches to normalization and standardization, respectively, of the data from different datasets and describes experiments with different preprocessing approaches. Our FCNN methodology efficiently handles both MS and HS images. It also addresses the challenges of band selection and spectral variability, making it a versatile and robust approach for different data settings. Furthermore, the architecture of the network and training hyperparameters are presented in detail.

The main advantages of our method lie in the generalization of the approach and adaptability to data of different spectral and spatial resolutions. Furthermore, by combining publicly-available datasets with appropriately labeled data for network training, we ensure a principled and accessible foundation for model learning.

Our proposed method also has the advantage of not requiring further image postprocessing of the obtained RGB image, which leads to a more polished and refined output straight out of the model.

In the case of FCNN processing, information aggregation is quite beneficial, especially when public data are limited. Such aggregation increases the methodology's efficacy despite related limitations. The visualization results demonstrate potential even when dealing with datasets containing fewer bands, indicating the versatility and robustness of our methodology across various data configurations. Such a methodology eliminates any doubt regarding the selection criteria of the effective bands used.

This article is organized as follows. Section II describes different MS and HS datasets used in our experiments, pointing out the challenges of working with multisource data. Section III presents the data preprocessing steps and the proposed FCNN model architecture, together with the model training process. The visual results of our experiments for different strategies used are presented and discussed in Section IV, and their comparative quality assessment is outlined in Section V, showing the potential of the new approach proposed in Section IV. Finally, the conclusions of this article and some limitations of our study, which became apparent after the completion of the research are emphasized in Section VI, together with future directions for our research.

II. Materials

A. Hyperspectral and Multispectral Imaging

MS and HS images are two types of remote sensing data that capture information about the Earth's surface by sensing electromagnetic radiation. As mentioned in the introduction, they are widely used in various fields, including agriculture and environmental monitoring. The key distinction between MS and HS images lies in their spectral resolution. HS images have a far higher spectral resolution, catching data in several narrow and contiguous bands, while MS images only capture data in a small number of discrete bands. This higher spectral resolution allows for a more detailed characterization of surface materials based on their spectral signatures. MS and HS images offer valuable insights into the Earth's surface, and the choice between them depends on the specific application requirements and the level of spectral and spatial detail needed for analysis.

Differences in sensor types used to capture MS and HS images can introduce specific challenges and variations in the data. Different sensors may have varying spectral band configurations, capturing data in different wavelength ranges or bandwidths. Multiple sources and sensor variety can result in inconsistency when comparing or combining data from different sensors. It is essential to carefully account for these differences to ensure accurate and meaningful analysis. Also, sensors used in MS and HS imaging require calibration to ensure the accuracy and reliability of the captured data. However, calibration procedures can vary between sensors and platforms. Inconsistencies in calibration methods can lead to differences in the radiometric or spectral accuracy of the acquired data. Proper calibration and validation techniques must address these issues and establish reliable and consistent datasets. Each sensor has its characteristics and limitations, such as spectral response functions, signal-to-noise ratios, and dynamic ranges. These characteristics can also affect the quality and reliability of the captured data. In spectral imaging, sensors can have different spatial resolutions, affecting the level of detail captured in the images. Combining or comparing red multisource sensor data with different spatial resolutions can introduce challenges, as the spatial variability may need to align better.

In addition, researchers and analysts should be aware of the sensor-specific characteristics and limitations and consider them when interpreting and comparing data from different sources. Standardization efforts are also conducted to facilitate cross-sensor compatibility and consistency. While multisource sensor differences can introduce challenges and variations in the data, proper preprocessing and understanding of the

sensor characteristics can be addressed to ensure accurate and meaningful MS and HS imagery analyses.

B. Description of the Datasets

The availability of MS and HS imagery with high-resolution spectral information has redefined our understanding of environmental and ecosystem phenomena. These datasets are increasingly accessible for both scientific and practical purposes, and their analysis represents an initial step toward gaining a deeper comprehension of the aforementioned phenomena. Several datasets are recognized and used in this article.

This study selected the CAVE and UGR datasets for the model's training due to their well-established reputation in the literature. They offer a diverse range of colors, and the images were captured under various environmental conditions. Moreover, a corresponding RGB image is available for each image in these datasets, facilitating a comprehensive analysis.

For the testing phase, three HS images were used, two of them provided by the relatively recent PRISMA satellite and one acquired by the Reflective Optics System Imaging Spectrometer (ROSIS-3) over Pavia University, Pavia, Italy.

1) CAVE Dataset: The CAVE dataset [25] comprises 32 MS images captured indoors under controlled illumination conditions. Each image has a resolution of 512 × 512 pixels in the spatial domain. The spectral range is between 400 and 700 nm, with a sampling interval of 10 nm, resulting in a total of 31 spectral bands. A corresponding RGB image with the same spatial resolution is provided for each MS image. This dataset does not contain any natural scenes.

2) UGR Dataset: The UGR dataset [26] contains 14 outdoor images of urban scenes. Most of these images possess a spatial resolution of 1000 × 900 pixels. The spectral range spans from 400 to 1000 nm, with a sampling interval of 10 nm, resulting in 61 spectral bands, out of which 31 fall within the visible spectrum. In addition, each MS image is accompanied by a corresponding RGB image, sharing the same spatial resolution.

As can be observed from the description, the CAVE and the UGR MS images have in common the first 31 wavelengths from the visual domain. This makes them suitable for using the same FCNN with 31 input neurons. From UGR images, we thus considered only the spectral bands in the visual range, as these are relevant for visualization.

3) PRISMA Images: The PRISMA images used for coloring in this study are obtained from the PRISMA HS satellite operated by the Italian Space Agency (ASI). Two specific images were captured, one on October 18, 2022, in the northern region of Brasov county, Romania, and the other on March 24, 2023. The satellite's HS sensors are capable of capturing images within a wide wavelength range of 239 spectral bands, spanning from 400 to 2500 nm. Among these bands, 66 falls within the visible-near infrared range (400–1010 nm), while 173 bands reside in the short-wave infrared range (920–2500 nm). The spectral sampling interval for the satellite's images is less than 12 nm. Regarding spatial characteristics, the images possess a resolution of 1000 × 1000 pixels, with a ground sample distance of 30 m [27].

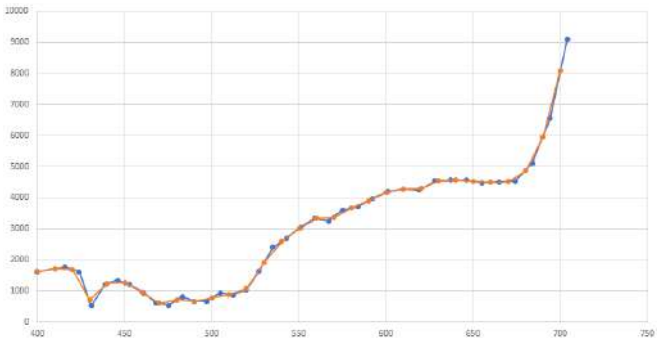


Fig. 1. Original values of one HS pixel from the first PRISMA image (blue) and interpolated values (orange).

The spectral bands used for the experiments are in the visible domain, covering the range from 406 to 713 nm, with an approximate sampling interval of 8 nm.

4) Pavia University Dataset: The Pavia University dataset was made available by the Telecommunications and Remote Sensing Laboratory, Pavia University, in 2001. This dataset was acquired using the ROSIS sensor during a flight campaign conducted over Pavia. The dataset comprises 610 × 610 pixels and covers a wavelength range from 430 to 860 nm with 115 spectral bands. It has a spatial resolution of 1.3 m and a spectral resolution of approximately 4 nm. However, some samples within the image do not contain any useful information and must be eliminated before analysis. Once the broken bands are removed, the 103 remaining bands can be used further in the investigation [28].

In order to test the PRISMA and the Pavia University images on the FCNN trained on the CAVE or the UGR dataset, it is necessary to adapt these images to the network's input layer. As this input is calibrated to receive pixels with the spectral signature of CAVE images, each spectral pixel from the test image has to be mapped on the wavelengths of a CAVE image pixel. This has been done in this study by linear interpolation. For each wavelength of a test image (PRISMA or Pavia University), the value of a CAVE image channel is interpolated from the values of the two neighboring channels of the test image.

Fig. 1 represents the original HS pixel from a PRISMA image together with the interpolated values of this pixel.

It can be seen from Fig. 1 that by linearly interpolating the PRISMA HS image to fit the bands of the CAVE dataset, the changes in the data profile are negligible, thus justifying this approach.

C. Prerequisites

In the context of machine learning experiments, standardization, and normalization are preprocessing techniques used to transform input data into a specific range or distribution. These techniques are commonly applied to improve the performance and convergence of machine learning models [29].

Standardization (z-score normalization or feature scaling) transforms the dataset's features into zero mean and unit variance. It entails dividing each data point by the standard deviation after taking the mean value of the feature out of each data point.

The formula for standardization is shown in the following equation:

$$z = \frac{x - \mu}{\sigma} \tag{1}$$

where z is the standardized value, x is the original value, μ is the mean of the feature, and σ is the standard deviation of the feature.

Standardization ensures that each feature has a similar scale and range, which is beneficial for algorithms that assume normally distributed data or when features have different scales. It centers the data around 0, with a standard deviation of 1.

Normalization (min–max scaling) transforms the dataset's features to a standard range—typically between 0 and 1. Still, the range depends on the specific normalization technique used and the requirements of the dataset. It involves subtracting the minimum value of the feature from each data point and then dividing it by the range.

The formula for normalization is shown in the following equation:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{2}$$

where x' is the normalized value, x is the original value, $\min(x)$ is the minimum value of the feature, and $\max(x)$ is the maximum value of the feature.

Normalization preserves the shape of the distribution and is suitable when the absolute values of the features are not important but rather their relative values or ratios.

Both standardization and normalization help improve the performance of machine learning models by ensuring that features are on a similar scale and avoiding the dominance of certain features due to their larger magnitudes. It is important to note that the impact of standardization and normalization on performance can vary depending on the dataset and the specific machine learning algorithm used. Therefore, it is advisable to experiment with both techniques and evaluate their impact on the model's performance before finalizing the preprocessing approach.

III. Methods and Procedures

The objective of this Section is to discuss some elements crucial to the success of the tests conducted for this study. For the proposed visualization to perform, it is of great importance to comprehend the preprocessing steps and the model employed.

A. Data Preprocessing Step

The preprocessing stage is essential to data preparation in machine learning and data analysis. Its main objective is to convert raw data into a format suitable for additional analysis or modeling.

Standardization and normalization are two of the many methods included in preprocessing. They seek to improve the performance and efficiency of various algorithms by bringing the data within a predictable and manageable range. Depending on the particular dataset, the type of issue, and the algorithms being applied, these preprocessing methods can change. It is advisable to test several transformation strategies and assess their effects on the model's performance to choose the best approach for a particular task.

In this study, various data preprocessing and transformation variations were tried precisely to study the feasible strategies and related results comprehensively. Thus, two strategies were highlighted in this article.

1) A strategy involving preprocessing on each file, concatenation of data from all the input files, shuffle, and separation by train subset and test subset ((3/4) and (1/4), respectively, of all existing pixels). We will further call this strategy *individual preprocessing*.
2) A strategy involving concatenation without any preprocessing beforehand, shuffle, splitting into train and test (with the exact percentages as the previous strategy), and application on the training subset of one of the proposed preprocessing methodologies (standardization or normalization). After training, our learning algorithm has learned to deal with the data in scaled form, so we have to normalize/standardize our test data with the normalizing/standardizing parameters used for training data. We will further call this strategy *global preprocessing*.

From a technical perspective, for both strategies, *StandardScaler()* [30] and *MinMaxScaler()* [31] have been used for standardization/normalization, respectively.

B. Proposed AI Model Architecture

The problem of consistent spectral image visualization is of significant importance, as it enables the users to visually interpret and understand the acquired data. For sensors like those of Landsat or Sentinel, with a small number of spectral bands overall and three bands in the visual range, the visualization problem is straightforward, as the three bans for red, green, and blue generate the corresponding RGB images. With the upcome of multisource modern MS and HS sensors with a wide range of spectral bands in the visual range, the problem of correct and accurate visualization gets more complex. Generally, classical methods like the ones mentioned in the introduction, often result in low-quality images and usually need adjustments.

Starting from the ideas in [23], an FCNNwas constructed and trained on a set of MS images, as mentioned in [24], in order to visualize spectral images acquired by different sensors. Below, we provide a detailed description of the network architecture and the hyperparameters utilized during training.

1) Model Description: Consistently mapping n-dimensional spectral pixel on a tridimensional RGB pixel can be considered a regression problem, and thus, an FCNN is an appropriate model for learning this mapping. The model, as illustrated in Fig. 2, consists of five layers: an input, an output layer, and three hidden layers.

The input layer consists of 31 neurons, as the number of channels of the CAVE images is 31, all in the visual range, and the number of spectral bands in the visual range of the UGR images is also 31. The number of output neurons is 3, one for each RGB color.

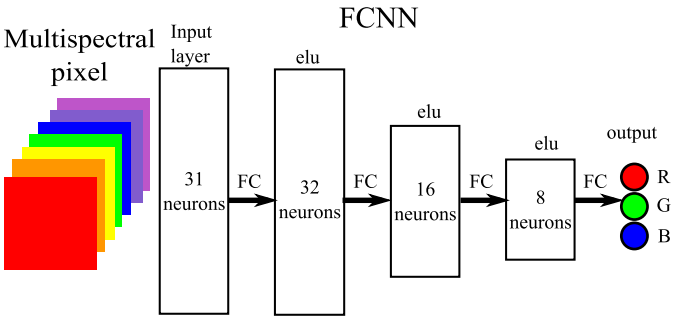


Fig. 2. Model pipeline.

In the hidden layer, the number of neurons was chosen to decrease from the size of the input toward the size of the output by a factor of 2. From this consideration and taking into account the computational advantage of data structured as the power of 2, in the first hidden layer, we used 32 neurons, in the second 16, and in the third 8. The activation function on the hidden layers is the exponential linear unit (ELU) [32], as we did not want to discard negative values.

Assessing machine learning models' performance is crucial for evaluating their effectiveness and suitability for specific assignments. Two commonly used loss functions are mean squared error (MSE) and mean absolute error (MAE). As MSE is very sensitive to outliers, while MAE reduces their impact almost completely, we chose the HuberLoss function, which combines both errors in an balanced way [33]. The Huber loss is defined as follows:

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2, & |y - f(x)| \le \delta \\ \delta\left(|y - f(x)| - \frac{1}{2}\delta\right), & |y - f(x)| > \delta \end{cases} \tag{3}$$

where y is the true value, $f(x)$ is the predicted value, and δ is a parameter that determines the threshold for which, in the loss function, the MSE is replaced by MAE.

For the data points where the distance between the label and the predicted value is small, less than the threshold δ, the function behaves like MSE. This makes it more robust to noise. However, for data points with larger differences, it switches to a linear loss. This makes it less sensitive to outliers and helps in providing a more stable estimation. If the outliers comprise 20%–30% of the data, the MAE will ignore them entirely. However, Huber loss will create a balance if the outliers are significant, and therefore, it is a useful choice when dealing with datasets that may contain outliers or noise.

The selection of δ plays a crucial role. In our case, the value $\delta = 10.0$ parameter for the HuberLoss was selected empirically. We considered that a difference of 10 between two values (of a color channel) is acceptable for using the MSE, similar to just noticeable difference (JND) corresponding to $\Delta E = 3$ in CIELab.

2) Model Training: After substantial testing, 150 epochs for training were determined to be adequate for accurate color mapping. The number of pixels in a batch was 2048 to shorten training time. In the backpropagation step, a common value for the learning rate $\alpha = 0.005$ and the Adam optimizer provided by the Pytorch library was used.

The model was trained on Intel[1] Core[2] i7-7700 CPU at 3.60-GHz server with eigh CPUs, and the duration of the training on 150 epochs on the UGR dataset was approximately 2 h.

The set used for training was preprocessed, as discussed in Section IV-A2, by concatenating all the pixels from all the images, shuffling, partitioning them into train and test sets, and standardizing over the train set.

The training algorithm can be described as follows.

Model Training Steps

1) Load pixels from all the images of the dataset (CAVE or UGR), shuffle the pixels, and partition into train and test (75% versus 25%).
2) Preprocess data by standardization using scikit-learn StandardScaler [30].
3) Split set in random batches of 2048 pixels.
4) Train the model using HuberLoss.

In order to enhance clarity and facilitate a deeper understanding of the proposed methodology, pseudocode has been provided to describe key steps in detail in Algorithm 1.

Algorithm 1 Pseudocode for the Training of the FCNN

Require: $path\_dataset$, $model$, $lr = 0.005$, $epochs = 150$
$(data, labels) \leftarrow load\_pixels(path\_dataset)$
$shuffle(data, labels)$
$nr\_train\_data \leftarrow (\frac{3}{4}) * len(data)$
$nr\_test\_data \leftarrow len(data) - nr\_train\_data$
$train\_set \leftarrow (data.head(nr\_train\_data), labels.head(nr\_train\_data))$
$test\_set \leftarrow (data.tail(nr\_test\_data), labels.tail(nr\_test\_data))$
Initialize the StandardScaler and Standardize training data
Apply Scaler on test data
$batch\_size \leftarrow 2048$
Initialize weights of the *model*
$optimizer \leftarrow AdamOptimizer(lr, weight\_decay = 0.0008)$
$best\_loss \leftarrow maxValue$
$loss\_function \leftarrow HuberLoss(10.0)$
$train\_loss\_vec \leftarrow []$
$test\_loss\_vec \leftarrow []$
for $epoch \in (1, epochs)$ **do**
  $train\_loss \leftarrow 0$
  for $batch, (x, y) \in train\_set$ **do**
    $y\_pred = model(x)$
    $loss \leftarrow loss\_function(y, y\_pred)$
    Backpropagation Step - backprop(loss)
    $train\_loss \leftarrow train\_loss + loss.item()$
  end for
  $train\_loss \leftarrow train\_loss/(batch\_size)$
  $train\_loss\_vec.append(train\_loss)$
  if $train\_loss < best\_loss$ **then**
    Save current weights
    $best\_loss \leftarrow train\_loss$
  else
  end if ▷ Now evaluate on test set
  $test\_loss \leftarrow 0$
  for $batch, (x, y) \in test\_set$ **do**
    $y\_pred \leftarrow model(x)$
    $loss \leftarrow loss\_function(y, y\_pred)$
    $test\_loss \leftarrow test\_loss + loss.item()$
  end for
  $test\_loss \leftarrow test\_loss/(batch\_size)$
  $test\_loss\_vec.append(test\_loss)$
end for
Save current weights
$plot(train\_loss\_vec, test\_loss\_vec)$

The graphical representation of the loss decay on the train and the test sets for the CAVE and UGR datasets are represented in Fig. 3. It can be seen from this figure, that

[1]Registered trademark.
[2]Trademarked.

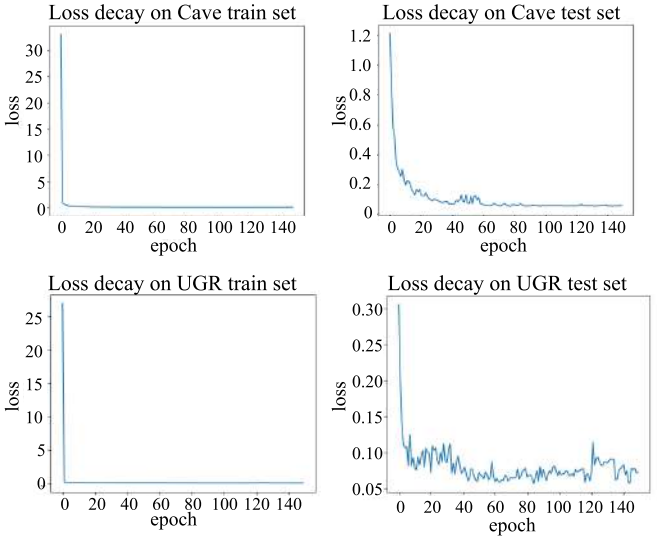


Fig. 3. Loss decay on the train and test sets when training the model with the CAVE dataset and UGR dataset, respectively.

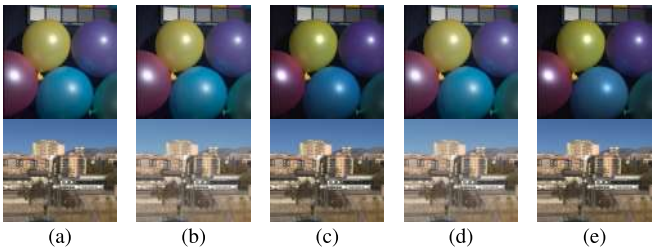


Fig. 4. Visualization results of the experiments described in Section IV-A, in which global preprocessing was done on the datasets. (a) One sample of CAVE and UGR datasets, respectively. The other subfigures show the results of the FCNN trained on (b) CAVE and (c) UGR, respectively, with global normalization and on (d) CAVE and (e) UGR, respectively, with global standardization of the training set.

on the train set, the loss decays very steeply during the first epochs and then gets stabilized, while to achieve similar results on the test set, the network needs more training.

IV. Results for Different Strategies

A. Results With Global Preprocessing

In these experiments, the general strategy is that after splitting into the train and test subsets, normalization or standardization was performed on the entire training set, and then, the results were applied to the test set. For inference, the chosen method is applied to the image being colored, using the parameters from the train.

1) Global Preprocessing With Normalization: Fig. 4(b) presented the results of the FCNN trained on the CAVE dataset, and Fig. 4(c) presented the results when training the FCNN on the UGR dataset, on one sample image, see Fig. 4(a), from each of these datasets. The data was preprocessed by min–max normalization on the respective training set using the MinMax scaler.

The results of the FCNN trained on the CAVE and UGR Datasets on the Pavia University image are presented in Fig. 6(a) and (b). The results of the model on one of the images acquired by the PRISMA Satellite in both treating

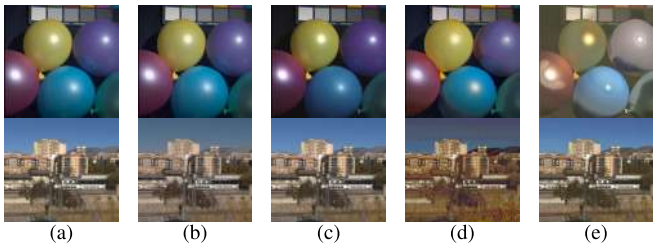


Fig. 5. Visualization results of experiments described in Section IV-B, in which preprocessing was done on each image of the datasets separately. (a) One sample of CAVE and UGR datasets, respectively. The other subfigures show the results of the FCNN trained on (b) CAVE and (c) UGR, respectively, with normalization and on (d) CAVE and (e) UGR, respectively, with standardization on each image of the training set.

scenarios are presented in Fig. 7(a) and (b). The Pavia University and the PRISMA spectral images were interpolated, to match the input spectral bands and were normalized using the parameters obtained on the respective training sets.

2) Global Preprocessing With Standardization: The results of the FCNN trained separately on the CAVE and the UGR datasets, respectively, on one sample image of each of these datasets [Fig. 4(a)] are presented in Fig. 4(d) for the CAVE and in Fig. 4(e) for the UGR trained network. The standardization in each case was done on the corresponding training dataset using the standard scaler.

The results of this method on the Pavia University image can be seen from Fig. 6(c) and (d), and the results on PRISMA HS image are presented in Fig. 7(c) and (d).

B. Results With Individual Preprocessing

In these experiments, before concatenating all the images and separating them into train and test subsets, each image is taken and brought into the same range of values by normalizing/standardizing each one according to its own values.

1) Individual Preprocessing by Normalization: Each image is considered separately, and then normalized according to the min/max in the image, thus transforming all values into the range [0, 1]. All the images are then concatenated and the resulting pixel set is split into train and test subsets, respectively. At inference, an image is normalized and colored according to its associated min/max values.

Fig. 5 presented the results of the coloring using the FCNN trained on CAVE and UGR datasets, respectively, normalizing each image before concatenating with the others.

Fig. 6(e) and (f) presented the coloring results on Pavia University and Fig. 7(e) and (f) those on the PRISMA image, with the CAVE, respectively UGR-trained FCNN, normalizing each image with respect to its own min–max values.

2) Individual Preprocessing by Standardization: In this approach, each image is considered separately, and then standardized according to the mean/average deviation in the image, so all values are brought into the standard range $[-1, 1]$. The processed images are then concatenated and the resulting pixel set is split into train and test subsets, respectively. At inference, an image is standardized and colored according to the distribution of values in the image.

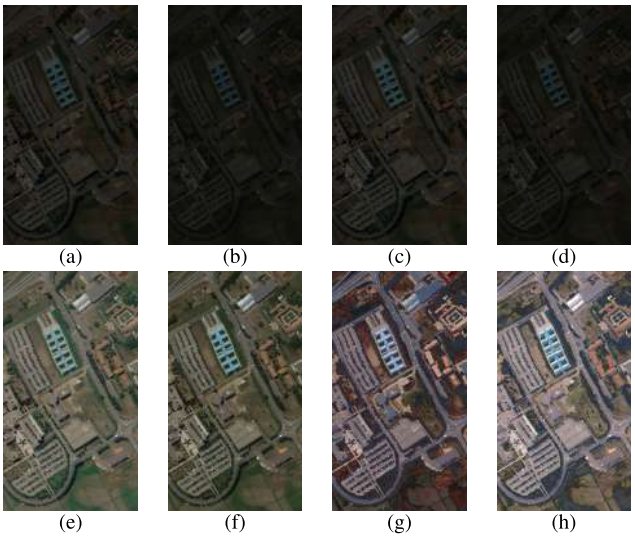


Fig. 6. Visualization of the results on the Pavia University MS image using the FCNN trained in the described scenarios on (a) CAVE and (b) UGR, respectively, with global normalization and using for inference the same scaler as for the training set; on (c) CAVE and (d) UGR, respectively, with global standardization and using for inference the same scaler as for the training set; on (e) CAVE and (f) UGR, respectively, with normalization of each image with respect to its own min–max values; and on (g) CAVE and (h) UGR, respectively, with standardization of each image with respect to its own mean and standard deviation.

Fig. 5 presented the results of the coloring using the FCNN trained on CAVE and UGR datasets, respectively, standardizing each image before concatenating with the others, on samples of these datasets.

Fig. 6(g) and (h) presented the coloring results on Pavia University and PRISMA using the CAVE- and UGR-trained FCNN, respectively, normalizing each image with respect to its own mean/std values. The visualization results of the FCNN on a PRISMA HS image are presented in Fig. 7(g) and (h).

C. Different Approach

The experiments interpreted from the previous figures show that the best coloring results on the CAVE and UGR datasets were obtained when normalizing or standardizing the training set (global preprocessing) and using the respective parameters on the test set. This strategy follows the standard practice in machine learning flows. On the other hand, the results were not acceptable on the Pavia University and the PRISMA images. This is not surprising, as these images are bound to.

Fig. 8 illustrated the distributions on the green channels (650 nm) for the CAVE Fig. 8(a) and the UGR Fig. 8(b) datasets, together with those of Pavia University Fig. 8(c) and the first PRISMA image Fig. 8(d), both calculated after interpolation. We selected the green channel for illustration, but the observation is valid for any other 31 channels of the spectral images considered.

As we aimed to obtain a model that should be able to visualize images from different acquiring sources with different distributions and spectral signatures, we tried following a nonstandard approach. We considered the FCNN model, with global preprocessing by standardization in the training stage. After training, to visualize Pavia University and PRISMA images, these were standardized concerning their own mean and standard deviation to obtain for each channel the mean of 0 and the standard deviation of 1.

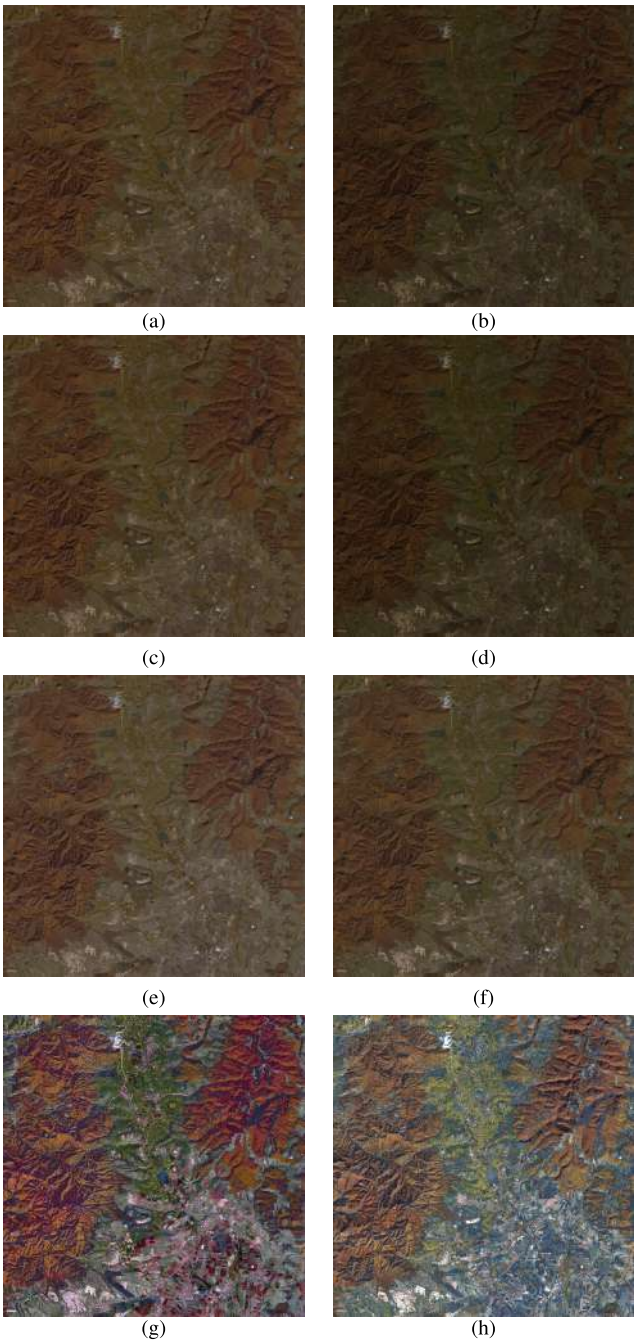


Fig. 7. Visualization of the first PRISMA HS image using the FCNN trained in the described scenarios on (a) CAVE and (b) UGR, respectively, with global normalization and using for inference the same scaler as for the training set; on (c) CAVE and (d) UGR, respectively, with global standardization and using for inference the same scaler as for the training set; on (e) CAVE and (f) UGR, respectively, with normalization of each image with respect to its own min–max values; and on (g) CAVE and (h) UGR, respectively, with standardization of each image with respect to its own mean and standard deviation.

The following algorithm can express the inference step.

Visualization Algorithm

1) Interpolate image to fit CAVE spectral range.
2) Load the pixels of the interpolated image.
3) Standardize the pixels with PyTorch StandardScaler relative to their mean and variance.

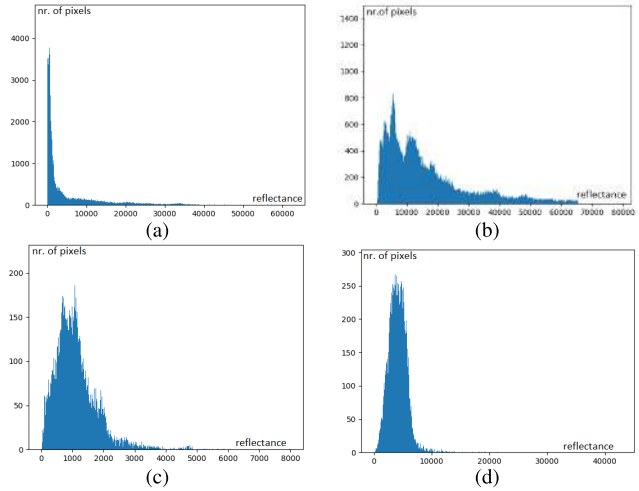


Fig. 8. Distributions on the green channel (considered at 550 nm) for CAVE and UGR datasets and for Pavia University and PRISMA spectral images. (a) Distribution for CAVE. (b) Distribution for UGR. (c) Distribution for Pavia University. (d) Distribution for PRISMA.

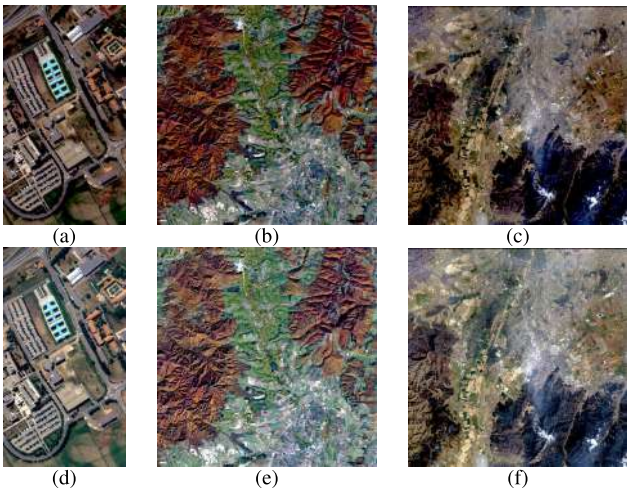


Fig. 9. Visualization of the results on Pavia University (a), first PRISMA image (b) and second PRISMA image (c) when standardizing those images according to their own distribution using the CAVE trained FCNN. Visualization of the results on Pavia University (d), first PRISMA image (e) and second PRISMA image (f) when standardizing those images according to their own distribution using the UGR trained FCNN.

4) Use the model to predict the corresponding (R, G, B) triplet for each interpolated image pixel.
5) Construct the RGB image with respect to the original size of the input image.

To enhance clarity and promote a thorough understanding of the proposed methodology, we have incorporated pseudocode of Algorithm 2 to provide detailed descriptions of inference step.

Algorithm 2 Pseudocode for Visualization using the FCNN

Require: $path\_ms\_image$, $model$, $input\_freq$, $target\_freq$
1: $image \leftarrow load(path\_ms\_image)$
2: $interpolated \leftarrow$ Interpolate$(image, input_freq, target_freq)$
3: Scale $interpolated$ with StandardScaler
4: $predicted \leftarrow model(interpolated)$
5: Make RGB image from $predicted$

The results of this approach are presented as follows: for the Pavia University image in Fig. 9(a) (CAVE-trained FCNN) and 9(d) (UGR-trained FCNN), for the first PRISMA image in Fig. 9(b) (CAVE-trained FCNN) and 9(e) and for the second PRISMA image in Fig. 9(c) and (f), respectively.

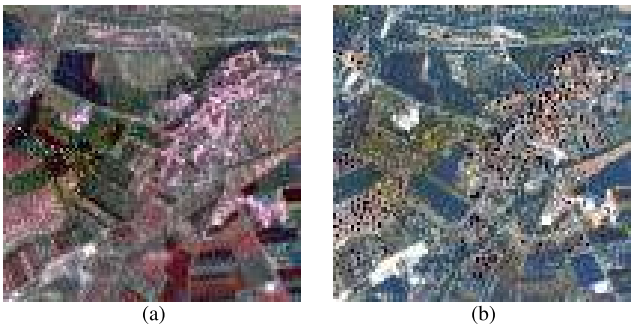


Fig. 10. Illustrations of the artifacts in the PRISMA image in a region of interest. (a) Region from the image in Fig. 7(g). (b) Region from image in Fig. 7(h).

V. Discussion and Comparisons

Various experiments have been carried out in this study, all aiming to better visualize the considered spectral images. And since we want to determine which of the discussed approaches is better, a variety of data from different types of sensors were used.

All the results to be commented on are presented from Figs. 4–13. We will comment on each experiment, in the same order in which they were performed.

Fig. 4 shows the results of the experiments in the case of global preprocessing by normalization over the whole training set. As can be noticed, visually, the results are quite similar in terms of colors. However, on closer inspection, it can be seen that there are also pairs with better contrast, in the sense that the coloring of an UGR image works better with a network trained with UGR images and similarly for CAVE images.

Fig. 5 shows the results of the experiments when each image is normalized/standardized according to its own values. As can be seen, normalization works better than standardization, which makes sense because during the preprocessing by normalization each feature undergoes a transformation to fit within a new range, while preserving its original relationships with the other features in the data, meaning that all the relational properties within the data remain intact [34].

Following these tests, this type of preprocessing with normalization seems to be a consistent option. The alternative with standardization is not plausible because this transformation step tends to change the relationships between colors, which leads to artifacts, especially for images that are not part of the training dataset, see Fig. 10(a) and (b).

Figs. 6 and 7 represent the results of tests performed on images obtained from different types of sensors and having different characteristics than the images in the training sets. To use the network on images such as Pavia University image or the images acquired by PRISMA, a mandatory step was the interpolation one, which is necessary to map the spectral bands of those images on the input with 31 channels of the model.

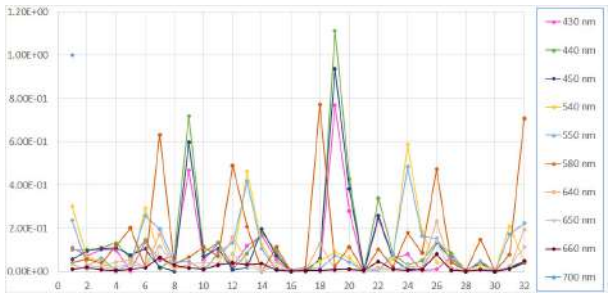


Fig. 11. Selection of neurons with highest absolute values on the first layer.

Regardless of the training dataset, CAVE or UGR, if we normalize or standardize over the whole training set and at the inference step we use the scaler/properties from the training stage, the results for the images taken from other sensors are very bad in terms of brightness and contrast, on Pavia even worse than on PRISMA [see Fig. 6(d)—standardization on UGR versus Fig. 7(d)].

Normalization on each image yields good results on Pavia [Fig. 6(e) and (f)] and acceptable on PRISMA [Fig. 7(e) and (f)]. In contrast, standardization on each image generates significant artifacts, see Figs. 6(g) and 7(g) for the standardization on each image in CAVE, and Figs. 6(h) and 7(h) for the standardization on each image in UGR. These artifacts can be seen more clearly on a selected region in PRISMA image, as illustrated in Fig. 10.

In Fig. 9, we presented different experiments for visualization of spectral images by means of a FCNN. As could be seen from all previous figures compared with Fig. 9, the best results on images that were acquired by other sensors than those used for the images in the training set and with another spectral signature, were obtained by using global preprocessing in the training step, but standardizing these images according to their own mean and standard deviation at inference.

We also compared our results with some conventional methods of spectral image visualization namely band selection and XYZ space [35] as well as two other methods: decolorization-based HSI visualization [36] and multichannel pulse-coupled neural network (MPCNN)-based HSI visualization [37]. Figs. 12 and 13 displayed the results of the visualization of the first PRISMA image and the second PRISMA image, respectively, using these methods.

Moreover, we studied if our method favors certain wavelengths with respect to others, as is the case for band selection. By plotting the weights between the input and the first hidden layer, we found out, that all the bands contribute in a balanced manner to the final results. Fig. 11 displayed a selection of these weights. Each color represents the weights of one input neuron to the neurons on the next layer, while each input neuron corresponds to one wavelength.

In conclusion, the results of our tests are better than the results of conventional methods, at least on PRISMA, our use case, regardless of how the preprocessing stage is performed. Given the fact that the standardization for other types of images is done relative to their mean and variance, we believe that a network that has been trained on images with a good distribution of light and contrast will tend to produce images

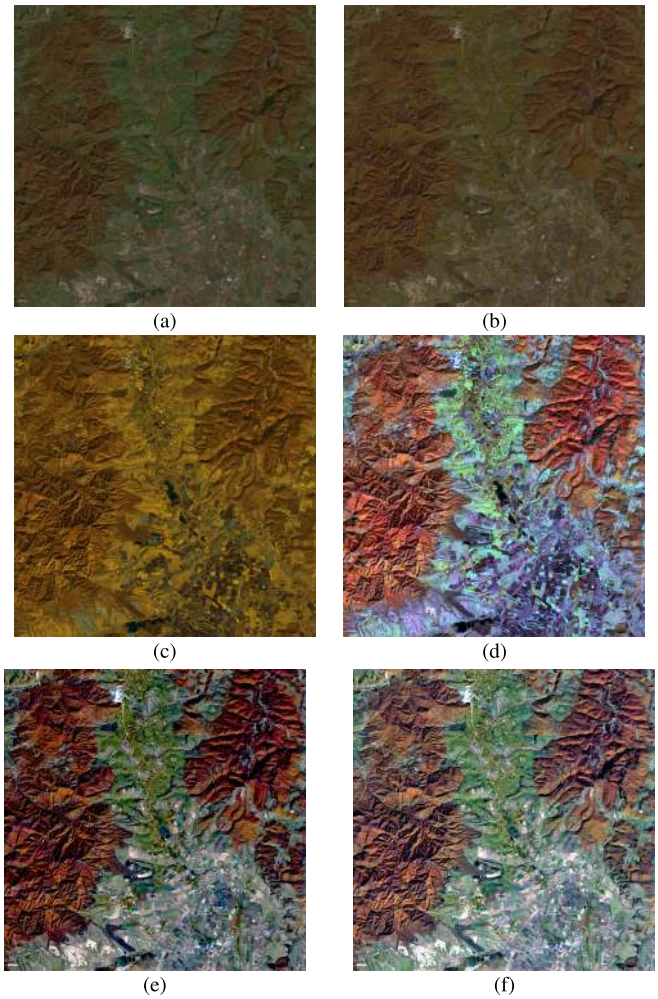


Fig. 12. Comparative results of classic and advanced visualization techniques with the new approach for the first PRISMA image. (a) Band selection. (b) Coloring with XYZ space. (c) Decolorization-based visualization. (d) MPCNN visualization. (e) Results with CAVE-trained FCNN. (f) UGR-trained FCNN.

with good contrast and light, even if the original images are affected by the atmospheric conditions. These results justify using this type of approach in the case of satellite images.

A. Comparative Quality Assessment

In order to perform a numerical quality comparison between the results presented in Figs. 12 and 13, some no-reference image quality assessments [38], [39] like entropy, which estimates information quantity, fractal dimension, which estimates the complexity and standard deviation, which estimates nonuniformity, were used. These values were calculated using adequate scripts offered by Ivanovici and Richard [40] for entropy and Ivanovici [41] and Caliman et al. [42] for fractal dimension, and the MATLAB *std2* function for the standard deviation of an image. The PRISMA images were correspondingly scaled as the scripts were calibrated for 256×256. For calculating the fractal dimension, the parameters used were $\text{LMAX} \in \{41, 71, 101\}$, representing the maximum size of the hypercubes and a threshold of 0.00001 for the standard deviation [43].

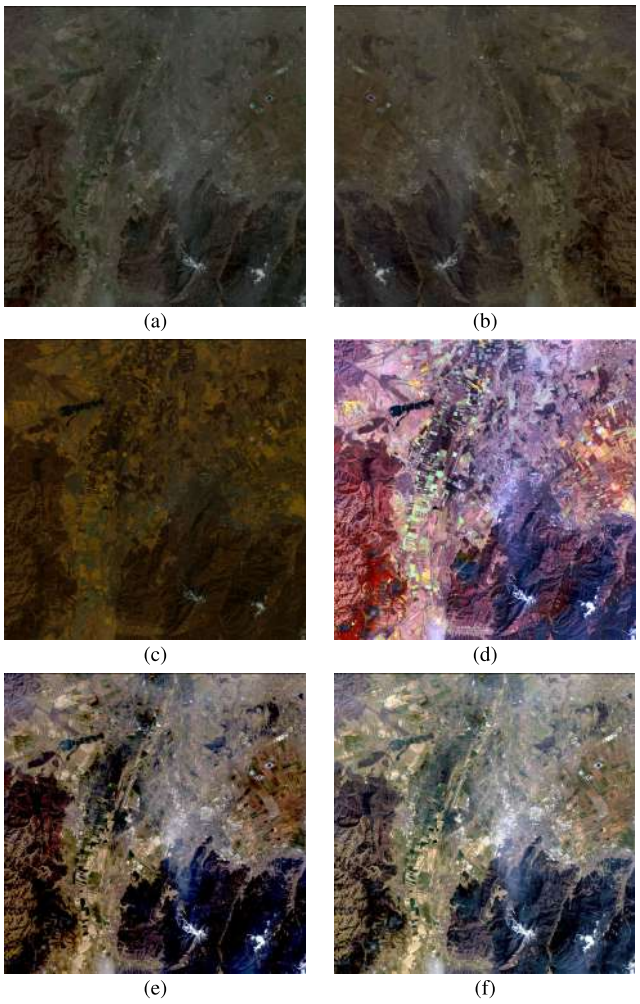


Fig. 13. Comparative results of classic and advanced visualization techniques with the new approach for the second PRISMA image. (a) Band selection. (b) Coloring with XYZ space. (c) Decolorization-based visualization. (d) MPCNN visualization. (e) Results with CAVE-trained FCNN. (f) UGR-trained FCNN.

It is important to note that the actual entropy values of color images will depend on the specific content and distribution of pixel intensities in the image. Images with complex color patterns, a wide range of intensities, and diverse color distributions are likely to have higher entropy values. Hence, a higher value of the entropy indicates a higher informational content, and a higher value of the fractal dimension indicates higher complexity of the scene and is directly correlated with the contrast of the image. A larger variance is also characteristic of images with a larger variety, thus greater informational content.

When satellite land cover images include agricultural areas, as well as urban and nonagricultural surfaces, as seen in the PRISMA images presented in this study, we expect a consistent visualization to exhibit large values for the entropy, the fractal dimension, and the variance.

The processed values for these measures in the case of the visualizations presented in Figs. 12 and 13 are displayed in Tables I and II. It can be observed that the largest values for entropy and variance are obtained in the case of the MPCNN method, closely followed by the results

TABLE I
COMPARATIVE QUALITY ASSESSMENT FOR THE FIRST PRISMA IMAGE

Sample	***Entropy***	***Standard Deviation***	***Fractal Dimension with*** L_{max}		
			41	71	101
FCNN / CAVE	15.6314	47.0056	3.7421	3.8271	3.8291
FCNN / UGR	15.8624	44.7105	3.7199	3.8064	3.7940
Band selection	12.6552	16.1236	2.7463	2.7484	2.8136
XYZ method	13.5262	20.4681	2.9795	2.9251	2.8930
Decolorization	14.0430	39.5901	2.9668	3.0568	3.1282
MPCNN	15.9284	48.0310	3.7245	3.9921	3.9566

TABLE II
COMPARATIVE QUALITY ASSESSMENT FOR THE SECOND PRISMA IMAGE

Sample	***Entropy***	***Standard Deviation***	***Fractal Dimension with*** L_{max}		
			41	71	101
FCNN / CAVE	14.9011	50.5844	3.1309	3.1925	3.2909
FCNN / UGR	15.5923	48.5822	3.0698	3.1678	3.2157
Band selection	14.2838	27.2990	2.4924	2.7755	3.1075
XYZ method	13.7575	20.7137	2.6343	2.7082	2.8104
Decolorization	12.9841	23.2493	2.6714	2.7646	2.8926
MPCNN	15.8832	53.4223	3.2908	3.4603	3.6736

provided by our FCNN. As the visualization results presented in Figs. 12(d) and 13(d), the MPCNN method provides images with large variability, but with completely unnatural colors, being thus unusable in the case of the PRISMA images. Thus, the visualizations obtained by our method, exhibit the best visual results and almost the best quantitative ones. Consequently, it is evident that the FCNN visualizations yield the best results for all the measures, with slightly greater contrast for the CAVE-trained FCNN. These quantitative results further justify the approach proposed in this article.

VI. CONCLUSION

Processing multisource spectral images is still challenging as more satellites with different sensor characteristics are launched, and their products are freely available for scientists and other users. This article aims to resolve part of the problems posed by a consistent and qualitative visualization of such images, using an FCNN trained on two of the most known public spectral datasets appropriate for this purpose.

Furthermore, we studied and performed several preprocessing procedures, which are very important in this FCNN approach to visualize multisource images exhibiting different spectral signatures.

The results we obtained were evaluated both visually and by conventional measures of information content. This evaluation has shown that the images generated by our method can provide a deeper understanding of various aspects, such as identifying agricultural patches and urban centers.

The visual and quantitative results indicate that the proposed methodology is a promising direction for a consistent and qualitative multisource spectral image visualization. There are still some limitations of this approach. For example, in cases where the source image exhibits minor variance, the application of standardization may compromise the realism of coloring.

The reliance on labeled data for training limits our dataset choices to those that are annotated and publicly accessible. This limitation could affect the diversity and representativeness of the training data; hence, further research is needed to address these issues. In addition, when combining datasets, we must either interpolate or truncate information to balance the dataset differences, each option having its trade-offs.

Our future research efforts will be based on addressing some of these limitations. This preliminary work can prepare the ground for our next objective that is interpreting the data through the perspective of vegetation indices and other elements relevant to the agricultural sector, thus increasing the usefulness of our approach in this sector.

ACKNOWLEDGMENT

The hyperspectral images from the PRISMA satellite presented in this article were kindly provided by the Italian Space Agency (ASI), Rome, Italy.

REFERENCES

[1] M. Weiss, F. Jacob, and G. Duveiller, "Remote sensing for agricultural applications: A meta-review," *Remote Sens. Environ.*, vol. 236, Jan. 2020, Art. no. 111402.

[2] T. Adão et al., "Hyperspectral imaging: A review on UAV-based sensors, data processing and applications for agriculture and forestry," *Remote Sens.*, vol. 9, no. 11, p. 1110, Oct. 2017.

[3] P. Rodríguez-Veiga et al., "Forest biomass retrieval approaches from Earth observation in different biomes," *Int. J. Appl. Earth Observ. Geoinf.*, vol. 77, pp. 53–68, May 2019.

[4] E. Vangi et al., "The new hyperspectral satellite PRISMA: Imagery for forest types discrimination," *Sensors*, vol. 21, no. 4, p. 1182, Feb. 2021.

[5] L. Du et al., "A comprehensive drought monitoring method integrating MODIS and TRMM data," *Int. J. Appl. Earth Observ. Geoinf.*, vol. 23, pp. 245–253, Aug. 2013.

[6] Y.-T. Chan, S.-J. Wang, and C.-H. Tsai, "Real-time foreground detection approach based on adaptive ensemble learning with arbitrary algorithms for changing environments," *Inf. Fusion*, vol. 39, pp. 154–167, Jan. 2018.

[7] X. Kang, X. Zhang, S. Li, K. Li, J. Li, and J. A. Benediktsson, "Hyperspectral anomaly detection with attribute and edge-preserving filters," *IEEE Trans. Geosci. Remote Sens.*, vol. 55, no. 10, pp. 5600–5611, Oct. 2017.

[8] G. Cheng, J. Han, and X. Lu, "Remote sensing image scene classification: Benchmark and state of the art," *Proc. IEEE*, vol. 105, no. 10, pp. 1865–1883, Oct. 2017.

[9] M. Mehmood, A. Shahzad, B. Zafar, A. Shabbir, and N. Ali, "Remote sensing image classification: A comprehensive review and applications," *Math. Problems Eng.*, vol. 2022, pp. 1–24, Aug. 2022.

[10] W. Rao, L. Gao, Y. Qu, X. Sun, B. Zhang, and J. Chanussot, "Siamese transformer network for hyperspectral image target detection," *IEEE Trans. Geosci. Remote Sens.*, vol. 60, 2022, Art. no. 5526419.

[11] L. Zhuang, L. Gao, B. Zhang, X. Fu, and J. M. Bioucas-Dias, "Hyperspectral image denoising and anomaly detection based on low-rank and sparse representations," *IEEE Trans. Geosci. Remote Sens.*, vol. 60, 2020, Art. no. 5500117.

[12] L. Gao, D. Wang, L. Zhuang, X. Sun, M. Huang, and A. Plaza, "BS3LNet: A new blind-spot self-supervised learning network for hyperspectral anomaly detection," *IEEE Trans. Geosci. Remote Sens.*, vol. 61, 2023, Art. no. 5504218.

[13] L. Gao, X. Sun, X. Sun, L. Zhuang, Q. Du, and B. Zhang, "Hyperspectral anomaly detection based on chessboard topology," *IEEE Trans. Geosci. Remote Sens.*, vol. 61, 2023, Art. no. 5505016.

[14] X. Sun, L. Zhuang, L. Gao, H. Gao, X. Sun, and B. Zhang, "Information retrieval with chessboard-shaped topology for hyperspectral target detection," *IEEE Trans. Geosci. Remote Sens.*, vol. 61, 2023, Art. no. 5514515.

[15] S. Le Moan, A. Mansouri, Y. Voisin, and J. Y. Hardeberg, "A constrained band selection method based on information measures for spectral image color visualization," *IEEE Trans. Geosci. Remote Sens.*, vol. 49, no. 12, pp. 5104–5115, Dec. 2011.

[16] H. Su, Q. Du, and P. Du, "Hyperspectral image visualization using band selection," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 7, no. 6, pp. 2647–2658, Jun. 2014.

[17] Y. Zhu, P. K. Varshney, and H. Chen, "Evaluation of ICA based fusion of hyperspectral images for color display," in *Proc. 10th Int. Conf. Inf. Fusion*, Jul. 2007, pp. 1–7.

[18] V. Tsagaris, V. Anastassopoulos, and G. A. Lampropoulos, "Fusion of hyperspectral data using segmented PCT for color representation and classification," *IEEE Trans. Geosci. Remote Sens.*, vol. 43, no. 10, pp. 2365–2375, Oct. 2005.

[19] H. A. Khan, M. M. Khan, K. Khurshid, and J. Chanussot, "Saliency based visualization of hyper-spectral images," in *Proc. IEEE Int. Geosci. Remote Sens. Symp. (IGARSS)*, Jul. 2015, pp. 1096–1099.

[20] D. Liao, S. Chen, and Y. Qian, "Visualization of hyperspectral images using moving least squares," in *Proc. 24th Int. Conf. Pattern Recognit. (ICPR)*, Aug. 2018, pp. 2851–2856.

[21] P. Duan, X. Kang, and S. Li, "Convolutional neural network for natural color visualization of hyperspectral images," in *Proc. IEEE Int. Geosci. Remote Sens. Symp.*, Jul. 2019, pp. 3372–3375.

[22] R. Tang, H. Liu, J. Wei, and W. Tang, "Supervised learning with convolutional neural networks for hyperspectral visualization," *Remote Sens. Lett.*, vol. 11, no. 4, pp. 363–372, Apr. 2020.

[23] R.-M. Coliban, M. Marincaǧ, C. Hatfaludi, and M. Ivanovici, "Linear and non-linear models for remotely-sensed hyperspectral image visualization," *Remote Sens.*, vol. 12, no. 15, p. 2479, Aug. 2020. [Online]. Available: https://www.mdpi.com/2072-4292/12/15/2479

[24] I. C. Plajer, A. Baicoianu, and L. Majercsik, "AI-based visualization of remotely-sensed spectral images," in *Proc. Int. Symp. Signals, Circuits Syst. (ISSCS)*, Jul. 2023, pp. 1–4.

[25] F. Yasuma, T. Mitsunaga, D. Iso, and S. K. Nayar, "Generalized assorted pixel camera: Postcapture control of resolution, dynamic range, and spectrum," *IEEE Trans. Image Process.*, vol. 19, no. 9, pp. 2241–2253, Sep. 2010.

[26] J. Eckhard, T. Eckhard, E. M. Valero, J. L. Nieves, and E. G. Contreras, "Outdoor scene reflectance measurements using a Bragg-grating-based hyperspectral imager," *Appl. Opt.*, vol. 54, no. 13, pp. 15–24, 2015.

[27] ASI. (Mar. 2020). *Prisma Products Specification Document Issue 2.3*. [Online]. Available: https://prisma.asi.it/missionselect/docs/PRISMA%20Product%20Specifications_Is2_3.pdf

[28] M. Garana, M. A. Veganzones, and B. Ayerdi. (May 3, 2023). *Hyperspectral Remote Sensing Scenes*. [Online]. Available: https://www.ehu.eus/ccwintco/index.php/Hyperspectral_Remote_Sensing_Scenes

[29] S. Ozdemir and D. Susarla, *Feature Engineering Made Easy: Identify Unique Features From Your Dataset in Order To Build Powerful Machine Learning Systems*. Birmingham, U.K.: Packt Publishing, 2018.

[30] (Jun. 2023). *Standard Scaler*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

[31] (May 2023). *MinMax Scaler*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

[32] S. Singh. (Feb. 2023). *ELU As an Activation Function in Neural Networks*. [Online]. Available: https://deeplearninguniversity.com/elu-as-an-activation-function-in-neural-networks/

[33] A. Jung, *Machine Learning: The Basics*. Berlin, Germany: Springer, 2022. [Online]. Available: https://books.google.com/books/about/Machine_Learning.html?hl=it&id=1IBaEAAAQBAJ

[34] G. Aksu, C. O. Güzeller, and M. T. Eser, "The effect of the normalization method used in different sample sizes on the success of artificial neural network model," *Int. J. Assessment Tools Educ.*, vol. 6, no. 2, pp. 170–192, Jul. 2019.

[35] M. Magnusson, J. Sigurdsson, S. E. Armansson, M. O. Ulfarsson, H. Deborah, and J. R. Sveinsson, "Creating RGB images from hyperspectral images using a color matching function," in *Proc. IEEE Int. Geosci. Remote Sens. Symp.*, Sep. 2020, pp. 2045–2048.

[36] X. Kang, P. Duan, S. Li, and J. A. Benediktsson, "Decolorization-based hyperspectral image visualization," *IEEE Trans. Geosci. Remote Sens.*, vol. 56, no. 8, pp. 4346–4360, Aug. 2018.

[37] P. Duan, X. Kang, S. Li, and P. Ghamisi, "Multichannel pulse-coupled neural network-based hyperspectral image visualization," *IEEE Trans. Geosci. Remote Sens.*, vol. 58, no. 4, pp. 2444–2456, Apr. 2020.

[38] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, Apr. 2004.

[39] S. Winkler, "Vision models and quality metrics for image processing applications," Ph.D. dissertation, Ecole Politechnique Federale de Lausanne, Lausanne, Switzerland, 2001.

[40] M. Ivanovici and N. Richard, "Entropy versus fractal complexity for computer-generated color fractal images," in *Proc. 4th CIE Expert Symp. Colour Vis. Appearance*, Prague, Czech Republic, 2016, pp. 6–7.
[41] M. Ivanovici, "Fractal dimension of color fractal images with correlated color components," *IEEE Trans. Image Process.*, vol. 29, pp. 8069–8082, 2020.
[42] A. Caliman, M. Ivanovici, and N. Richard, "Colour fractal dimension for psoriasis image analysis," in *Proc. Signal Process. Appl. Math. Electron. Commun. (SPAMEC), EURASIP, Session VI*, Aug. 2021, pp. 113–116.
[43] M. Ivanovici and N. Richard, "Fractal dimension of color fractal images," *IEEE Trans. Image Process.*, vol. 20, no. 1, pp. 227–235, Jan. 2011.

Ioana Cristina Plajer received the B.E. and M.S. degrees in computer science from the University of Bucharest, Bucharest, Romania, in 1997 and 1998, respectively, and the Ph.D. degree in computer science from the Transilvania University of Braşov, Braşov, Romania, in 2011.

She is currently a Lecturer with the Faculty of Mathematics and Computer Sciences, Transilvania University. She is also a member of the Department's Machine Learning Research Group, founded in 2018 and part of the Project Artificial Intelligence and Earth Observation for Romania's Agriculture (AI4AGRI). Her research interests include machine learning, image processing, spectral imaging and remote sensing, formal languages, algorithms, and data structures.

Alexandra Băicoianu received the Ph.D. degree from Babes Bolyai University, Cluj-Napoca, Romania, in 2016.

She has been a Lecturer with the Transilvania University of Braşov, Braşov, since 2017, teaching various courses and seminars. She is currently a Research Engineer in informatics. She authored or coauthored more than 30 scientific articles and is the coauthor of six books. Also, she has supervised tens of graduation and dissertations thesis, programming training courses, programming Summer Schools, and code/tech Camps, some of them in collaboration with IT companies. She is also a member of the Department's Machine Learning Research Group, founded in 2018. She was part of various scientific projects, among them it is important to mention Advanced Technologies for Intelligent Urban Electric Vehicles, Powerful Advanced N-level Digital Architecture (PANDA), Intelligent Motion Control under Industry4.E (IMOCO4E), Artificial intelligence and Earth observation for Romania's agriculture (AI4AGRI), Digital Technologies and Artificial Intelligence (AI) solutions projects (DiTArtIS), and New modular Electrical Architecture and Digital Platform to Optimise Large Battery Systems on SHIPs (NEMOSHIP). Her research interests and expertise are in the field of machine learning, formal languages and compilers, algorithms, remote sensing and Earth observation data, autonomous driving, and electric and hybrid vehicles.

Luciana Majercsik received the B.S. degree in mathematics from the University of Bucharest, Bucharest, Romania, in 1998. She is currently pursuing the Ph.D. in computer science with the Transilvania University of Braşov, Braşov, Romania.

Her research interests include machine learning, multi-/hyperspectral image analysis and visualization, and graph-based methods in remote sensing.

Mihai Ivanovici (Member, IEEE) received the Ph.D. degree in electronics and telecommunications from Politehnica University, Bucharest, România, in 2006.

He was an Invited Researcher in 2008, 2010, and 2014; and a Visiting Professor with the University of Poitiers, Poitiers, France, in 2018, University Toulouse 3 Paul Sabatier, Toulouse, France, in 2019, and Technical University of Moldova, Chişinău, Moldova, in 2023. He is currently a Full Professor with the Electronics and Computers Department, Transilvania University of Braşov, Braşov, România. He is also the Head of Multispectral Imaging and Vision Research Laboratory. His research interests include in the field of algorithms and electronic system design for signal and data acquisition, processing, and analysis—including color, multi- and hyperspectral images, remote sensing, and Earth observation data, as well as data from particle detectors in the ATLAS Experiment, CERN, Geneva, Switzerland.

Dr. Ivanovici has been a member of the IEEE Signal Processing Society since 2008 and the IEEE Geoscience and Remote Sensing Society since 2018.

Heliyon 9 (2023) e18402

Contents lists available at ScienceDirect

Heliyon

journal homepage: www.cell.com/heliyon

Research article

Medical emergency department triage data processing using a machine-learning solution

Andreea Vântu [a], Anca Vasilescu [b,*], Alexandra Băicoianu [b]

[a] Faculty of Mathematics and Computer Science, Transilvania University of Braşov, Romania
[b] Department of Mathematics and Computer Science, Transilvania University of Braşov, Romania

ARTICLE INFO

Keywords:
Emergency medicine
Triage
Clinical decision support
Patient medical record
Medical data processing
Machine learning
Supervised learning algorithms

ABSTRACT

Over the years, artificial intelligence has demonstrated its ability to overcome many challenges in our day-to-day life. The evolution of it inquired more studies about Machine Learning possible solutions for different domains, including health care. The increasing demand for artificial intelligence solutions has brought accessibility to loads of data, including clinical data. The availability of medical records facilitates new opportunities to explore Machine Learning models and their abilities to process a significant amount of data and to identify patterns with the purpose of solving a medical problem. Understanding the applicability of artificial intelligence on this type of data has to be a compelling aim for emergency medicine clinicians. This paper focuses on the general clinical problem of the complex correlation between medical records and later diagnosis and, especially, on the process of emergency department triage which uses the Emergency Severity Index (ESI) as triage protocol. This study presents a comparison between three different Machine Learning models, such as Logistic Regression, Random Forest Tree and NN-Sequentail, with the purpose of classifying patients with an emergency code. We conducted four experiments because of imbalanced data. A web-based application was developed to improve the triage process after our theoretical and exploratory results. Overall, in all experiments, the NN-Sequential model had better results, having, in the first experiment, a ROC-AUC score for each ESI emergency code of: 0.59%, 0.76%, 0.71%, 0.78% 0.64%. After applying methods to balance the data, the model yielded a ROC-AUC score for each emergency code of 0.72%, 0.75%, 0.69%, 0.74%, 0.78%. In the last experiment consisting of a three-class classification problem, the NN-Sequential and Random Forest Tree models had similar metric outcomes, and the NN-Sequential algorithm had a ROC-AUC score for each emergency code of: 0.76%, 0.72%, 0.84%. Without any doubt, our research results presented in this paper endorse this tremendous curiosity in Machine Learning applications to enrich aspects of emergency medical care by applying specific methods for processing both medical data and medical records.

1. Introduction

From a theoretical point of view, Machine Learning provides various learning algorithms for different types of problems [1], [2], but practical solutions emerge to increase Machine Learning involvement and impact on quality of life. Artificial Intelligence

* Corresponding author.
E-mail addresses: andreea.vantu@unitbv.ro (A. Vântu), avasilescu@unitbv.ro (A. Vasilescu), a.baicoianu@unitbv.ro (A. Băicoianu).

https://doi.org/10.1016/j.heliyon.2023.e18402
Received 11 January 2023; Received in revised form 17 July 2023; Accepted 17 July 2023

Available online 22 July 2023

has many real-life applications, from solutions for renewable energy system [3] to Emergency Department improvements [4] [5] [6]. Over the last two decades, many research papers about the applications of Machine Learning in medicine and health care were published; for example, from [7] in 2001 to [8] in 2022, specific medical data processing methods were revealed. Part of them focuses on medically diagnosing a patient with a specific disease, like predicting diabetes [9] or trying to improve the Emergency Department flow with technology [10]. It is important to accumulate and recall the individuals' medical records efficiently and, therefore, accurately evaluate and classify symptoms in emergency medicine and services, which can be consistently improved by implementing Machine Learning algorithms. In papers like [10] or [11], predicting the admission of a patient after the Emergency Department triage and also predicting the emergency levels using the Machine Learning approaches are covered in various contexts. The admission of a patient represents a binary problem with a simple output as *Admit* or *Discharge*.

This study aims to research supervised Machine Learning models applied to a multi-class classification problem, such as predicting an emergency code to one individual in the specific context of the Romanian emergency medical system. This research focuses on a Machine Learning approach to the Emergency Room (ER) or an Emergency Department (ED) triage and follows the ideas from [12] and [13] that refer to a software web-based application developed to enhance the emergency triage process. For that particular work, information and data were collected from the local pediatric hospital, along with the demands and needs addressed by the hospital management team. The project was developed to digitize the workflow. That software covered three main aspects: (1) showing a waiting list according to every patient's assigned emergency code, (2) after having symptoms as input, the application made an emergency code suggestion, and (3) creating the patient file that could run on smart devices.

The demand for Machine Learning algorithms applied to emergency data has increased over the years [4]. However, there are still gaps in the literature, and more studies need to be conducted on this subject. By presenting this study, we want to evaluate Machine Learning models on classifying patients with an emergency severity index [14]. The proposed solution is based on medical records processing in order to classify patients on one of the 5-level ESI emergency codes [15] using supervised learning algorithms and also having a basic web interface for the clinicians as end-users to early monitor those patients. The better an AI-based prediction application is, the more convincing it could be, and eventually, clinicians could rely on the accuracy of a Machine Learning algorithm [16]. It yields that the research has to provide materials and methods, models and metrics, and computational experiments, which are significant from clinicians' perspectives. The next sections of our paper successively follow these topics. Three classifying supervised Machine Learning models and also four dataset approaches are chosen, and appropriate experiments are developed in order to provide patient classification emergency code as accurately as possible.

2. Materials and methods

The solution proposed by this research is meant to solve a multi-class classification problem where the dependent variable consists of 5 classes identified by numbers from 1 to 5. All necessary processes and the implementation were done using Python programming language (Python 3.10.6 version) and its corresponding packages [17].

2.1. Problem analysis

The Emergency Department is one of the most important decision points in a health care system. It is developed using a triage algorithm which prioritizes emergencies. This triage algorithm differs from country to country. In Romania, the triage protocol is based on ESI algorithm. ESI stood for *Emergency Severity Index* and was initially developed in 1999 [15]. It consists of 5-level emergency codes. *Level 1*, the red one, is the most critical level, and the intervention should occur immediately because the patient is in a life-or-death situation. The second level, or the yellow one, means that the patient has a high risk of deterioration, but the patient can wait up to 10 minutes. *Level 3*, the green level, means the patient needs urgent care, but they can wait up to 30 minutes without risking their lives. *Level 4*, or the blue one, represents a patient with a stable health status who requires one resource, and it is considered a non-urgent situation so that the patient can wait up to one hour. The last level, the white one, is also considered non-urgent, and here the patient can wait even two hours. Each patient follows the triage protocol represented here in Fig. 1, and the triage clinician decides to assign an emergency code out of all 5 codes based on specific rules (chief complaints, age, past medical history, triage vital signs, present treatment etc.) following the triage questions.

Overcrowding is the main issue of the Emergency Department. The ESI triage protocol is meant to be as accurate as possible, but it is prone to human error, especially when the medical staff is facing a large number of people at a time. Due to these issues, the health status of a patient can be easily overestimated or underestimated. Overestimation means using more medical resources, but underestimation could result in negative outcomes such as deaths [19].

2.2. Data preparation and preprocessing

The dataset used for this research [20] included adult ED visits within two years, specifically March 2014 and July 2017. The data were collected from three emergency departments. The initial paper [21] used this dataset to predict the admission outcome of a patient, but the features are suited to study emergency levels classification.

This dataset contains 972 features and 560486 entries. All information was gathered in a 1.32 GB CSV file. All these features can be grouped, and they determine separate bigger categories such as demographics, past medical history, historical vitals, triage vital signs, outpatient medication, chief complaints, historical labs, imaging, and hospital usage.

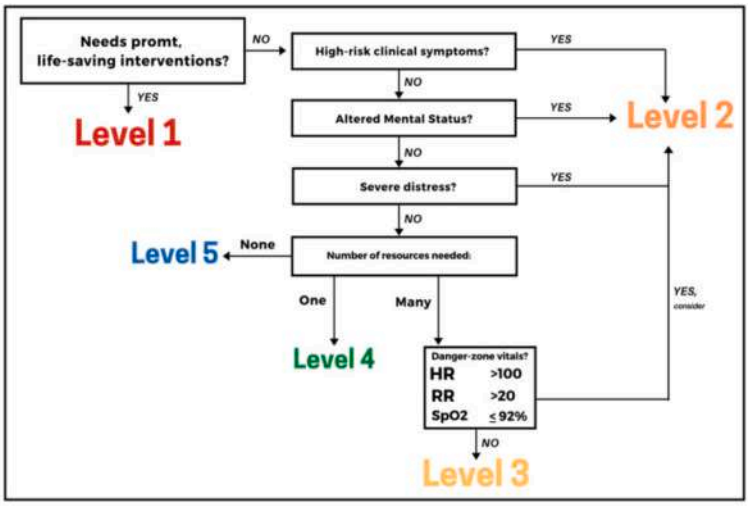


Fig. 1. ESI triage protocol [18].

Table 1
Data analysis.

Category	Attribute	Data type
Response variable	esi	Categorical
Past medical history	From 2sndarymalig to whtblooddx	Categorical
Demographics	age	Numeric
	gender	Categorical
	arrivalmode	Categorical
	previousdispo	Categorical
Triage Vital Signs	triage_vital_hr	Numeric
	triage_vital_rr	Numeric
	triage_vital_sbp	Numeric
	triage_vital_dbp	Numeric
	triage_vital_o2	Numeric
	triage_vital_temp	Numeric
Chief Complaints	From cc_abdominalcramping to cc_wristpain	Categorical

This study does not include all the information provided by this dataset. We have structured and removed the irrelevant features, and we only kept these categories: demographics, past medical history, triage vital signs and chief complaints. This selection is represented in Table 1, keeping those attributes which are more suitable for the Romanian ER patient file, with the intention of further work to fitting that medical system care. Therefore, only 409 features were selected for this research, and all the entries were kept. The resulting dataset is a mix of numerical, binary and text values. For a good workflow, the data types were determined.

The first step in this problem was to understand the variables, assess them and do some basic exploratory data analysis. The starting point consists of plotting the correlation matrix of the trained dataset and using the correlation coefficient's warnings and implications. The response variable is also called the dependent variable because the outcome of it depends on the other attributes. In this case, the *esi* response depends on the symptoms of the patient, for example.

One can evaluate the relationship between each target label and different features using correlations and selecting those features that have the strongest values. For this study, we started analyzing the relationships with triage vital signs, past medical history and chief complaints. The corresponding correlation coefficients with triage vital signs can be seen in Fig. 2.

Next, we scanned the results while looking at past medical history and chief complaints. Because there were plenty of features in the dataset, we selected only ten features for each category (see Fig. 3 and Fig. 4).

Chief complaints represent the symptoms which patients describe when they arrive at the emergency room or department. Back pain, rash and sore throat are the closest ones to 1. The very purpose of the data preprocessing phase is to turn the raw data into a more understandable, useful and efficient format for ML models. Such as, after all the necessary information regarding the classification of one of the 5 emergency codes was gathered from the original data frame, the dataset was split into training and test set. This step was accomplished using *sklearn* library within Python. The training set represents 80% of the data, and the rest of 20% represents the test set. Both training and test sets were preprocessed. The preprocessing step included: handling missing values, scaling the data, and handling text attributes.

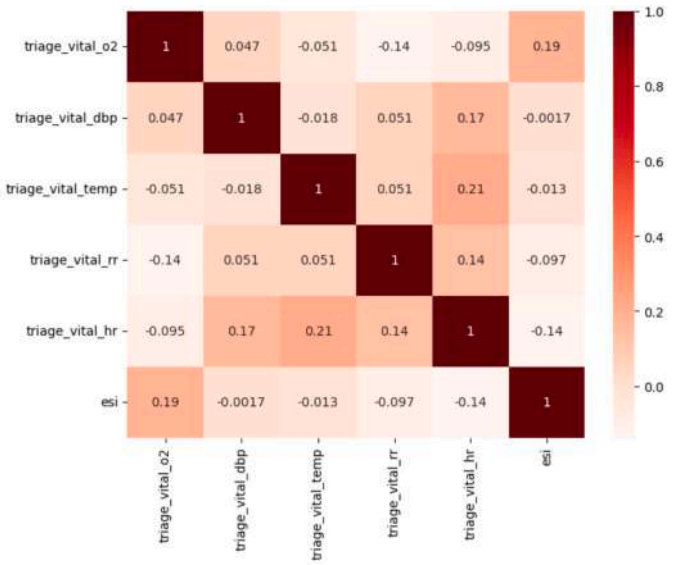


Fig. 2. ESI - Vital Triage Signs Correlation Matrix.

Firstly, it is important to see if the dataset has any missing values and the amount of them because some models do not know how to handle them, and the performance of the algorithms could be affected accordingly. There are many options to treat this case, as explained in [22].

The dataset in this study had missing values for the following features: *esi*, *age*, *arrivalmode*, *vital_triage_hr*, *vital_triage_rr*, *vital_triage_sbp*, *vital_triage_dbp*, *vital_triage_o2*, *vital_triage_temp*, plus some chief complaints attributes. For the *esi* attribute, we decided to drop all the entries with missing values because imputing the NaN value for this *Categorical* variable to the most frequent or median, does not necessarily represent the actual truth which would lead to a false reality. The records which had missing values on *esi* level had missing values on the other columns listed above. Hence, dropping those records fixed the NaN problem on the other features. The remaining missing values on the *age* column were treated in the same way because there were a few records, and it would not affect the outcome of the model's prediction in a significant way. The missing values presented by the triage vital signs were imputed with the median using the class *SimpleImputer* from Python *sklearn.impute* package. We chose median because these are continuous numeric attributes.

Then all the *Numeric* attributes were scaled using the *sklearn* object *StandardScaler()*. Scaling the data is not mandatory, but having big differences between the values could cost the performance of the model. Many Machine Learning models would have a better performance if scaling numerical input has been done before [23]. The dataset contains 7 *Numeric* features with values starting from 18 and going up to 105, and all the other features are *Categorical* having binary values such as 0, 1. We tested the given dataset with the feature selection, and we imputed the missing values, but the data was not scaled, so the model started to perform poorly. Therefore, in order to have a good or decent model performance, the scale was necessary.

Our dataset has three columns with *Categorical* values of type *String*. These columns are: *gender*, *arrivalmode* and *previousdispo*. We used the *OneHotEncoder* class to handle these categorical text features transforming them into *Numeric* ones. It will derive the category based on the unique values in each feature. For example, *gender* feature has only two possible and unique values: male and female. Using *OneHotEncoder* this column will be transformed into two columns with the names: *gender_male* and *gender_female*. These two new columns will be filled with 0 and 1 according to the values on the original column. The same will happen with the other text columns, each unique value being a new feature in the dataset.

The missing values within *arrivalmode* were treated with *SimpleImputer* class but with the most frequent strategy that can be used with both string and numeric types. The median strategy works only with numeric attributes.

3. Models development and evaluation

Since this research project focuses on a multi-class classification problem, namely predicting the patient's emergency code, it has been important to choose the model that suits the problem and has a good predicting score. Until then, one has to test different algorithms to see which one has a better performance. Supervised learning represents those types of Machine Learning in which machines are trained using well-labelled training data. This type of learning implies already-known output data.

For this study, we chose three classifying supervised Machine Learning models, particularly: Neural Network Models (i.e. NN Sequential Model), Decision Trees (i.e. Random Forest Tree Algorithm) and Regression Algorithms (i.e. Logistic Regression Algorithm).

Neural Networks models are designed and named after human biological neurons. They work using a defined number of layers, and each layer contains a precise number of neurons. They are applied to numerous real-world problems, from classifying to facial recognition. Random Forest Tree models represent groups of many individual decision trees that work together to predict the outcome

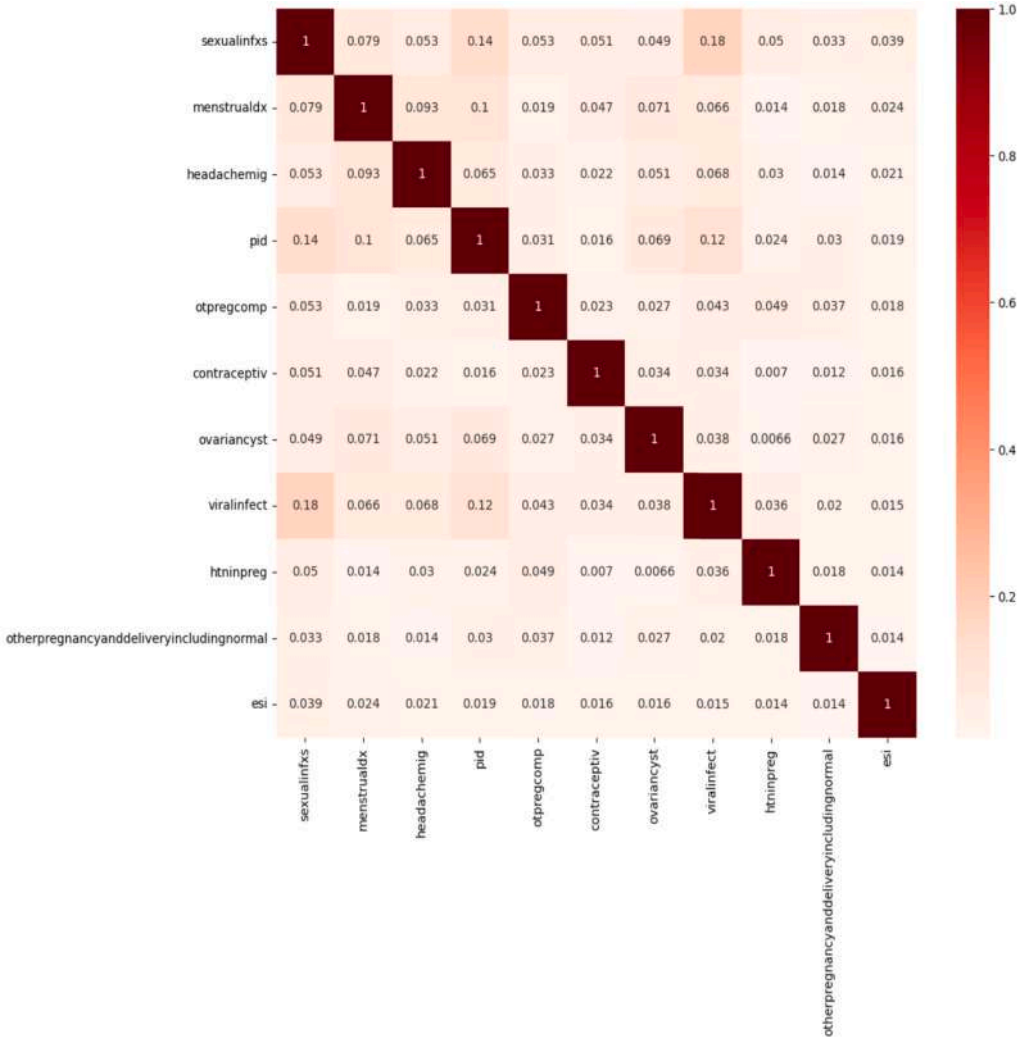


Fig. 3. ESI - Past Medical History Correlation Matrix.

of the input. The logic behind them is simple as it consists in evaluating nodes of a tree. Regression algorithms are used to assess the relationship between features and dependent variables, and they have applicability in fields such as forecasting, trend analysis and so on. We chose Logistic Regression for this study because it is a regression algorithm that can be used for classifying problems.

Usually, Logistic Regression is good for the first guess, it is a simple way to give some results, and it is a popular approach among medical data processing [24] [25] [26]. In addition, Random Forest Tree is a cheap variant, they tend to learn too well, so you usually need a comparative model to contrast them. In particular, one of the advantages offered by a decision tree algorithm is a lower prediction error due to its ability to recognise effects between a predictor and target variable [27]. The NN-Sequential model is sufficient for exploring the given data.

Once the models are chosen, one should take care of the measurements. When we think about a Machine Learning model and its prediction performance, most of the time, we think about accuracy. *Accuracy* is not necessarily the ultimate metric which tells if the model is good or bad, it is important to explore other metrics to give conclusions. A model could give amazing results evaluating one metric such as the accuracy score, but it could act poorly when it comes to another measurement like *F1-score*, which could give relevant insights about the dataset and about the problem in the first place.

This study, as we previously mentioned, treats a multi-class classification problem so that, for each model, the ROC (*Receiver Operating Characteristic*)-AUC (*Area Under the ROC curve*) score, *precision*, *recall*, confusion matrix and *specificity* were evaluated. These metrics prove to be suitable for medical data showing relevant results towards this direction, and besides, they provide clinicians with compelling insight. Some studies only measure the ROC-AUC score, but working with medical data should imply more metrics than this one [28]. The ROC-AUC score is a performance measurement telling how much a model is capable of distinguishing the classes. To make an analogy with medical data, a high score after evaluating this metric means the model is well performing at distinguishing patients who have a disease from the ones who are healthy, for instance, or distinguishing patients who were admitted to the hospital from patients who were not [21].

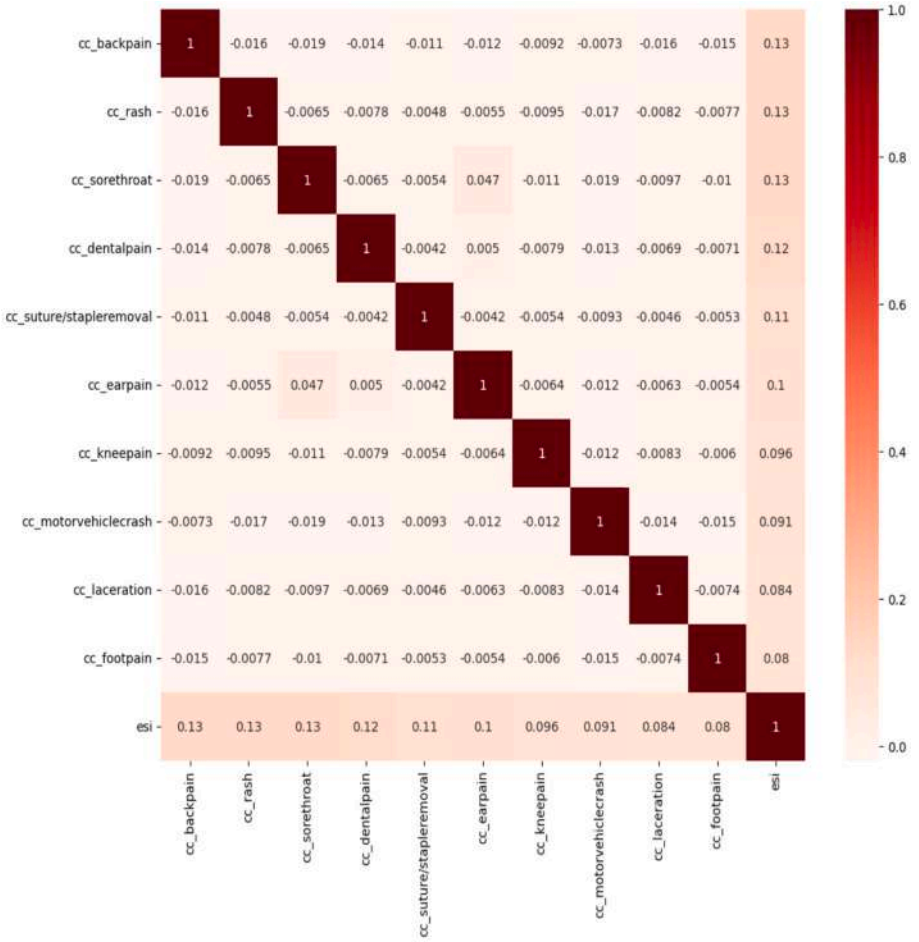


Fig. 4. ESI - Chief Complaints Correlation Matrix.

Considering the usual definitions for the outcomes TP as *True Positive*, FP as *False Positive*, TN as *True Negative*, and FN as *False Negative* and respectively *True Positive Rate* and *False Positive Rate* as in equation (1).

$$TPR = \frac{TP}{TP + FN} \text{ and } FPR = \frac{FP}{FP + FN} \tag{1}$$

the ROC curve maps TPR versus FPR for different classification thresholds, and AUC evaluates the area under the ROC curve between the thresholds corresponding points (0,0) and (1,1) [1]. If AUC yields a high score, then the model performs well at predicting false as false and true as true. An AUC score close to 0 means a poor performance by the model.

All of the metrics offer important details about the performance of the model. For a binary classification problem, it would be easier to pay more attention to a certain metric like *recall*. The admission of a patient could be a binary classification problem if the model has to predict whether the patient was admitted or discharged. *Recall* would show which patients were discharged when they actually needed admission. In our case, the question was not that simple, so we were interested to see all the chosen metrics. It was our target to see how many patients were predicted to have a *Level 3* emergency code, how many were classified on a higher level and also what amount of patients was underestimated, for example. Nevertheless, the ROC-AUC score gives an overview of how accurate the model would be in the future [29]. From a medical perspective, *precision*, *recall* and *specificity* metrics explain more than the performance of the model. All of these three metrics can be calculated using the confusion matrix. A confusion matrix consists of rows which represent the predicted classes and columns which are the true classes. Then, each cell represents the predicted output for each class. Table 2 represents a confusion matrix of our approach.

Considering *Level 2* as an example, the TP, FP, TN, and FN can be applied in a multi-class classification as follows. TP represents the cases predicted as *Level 2*, and the actual output was *Level 2*; its corresponding value following the confusion matrix is TP = 5. FP represents the cases predicted as *Level 2*, and the actual output represents other levels; its corresponding value is FP = 8 + 7 + 3 + 0 = 18. TN represents the cases predicted not as *Level 2* and the actual output was not *Level 2*; its corresponding value is TN = 9 + 3 + 2 + 4 + 8 + 1 + 2 + 4 + 10 + 15 + 1 + 2 + 3 + 4 = 68. FN represents the cases predicted not as *Level 2*, and the actual output was *Level 2*; its corresponding value is FN = 6 + 4 + 3 + 1 = 14.

Table 2
Confusion matrix.

	Level 1	Level 2	Level 3	Level 4	Level 5
Level 1	15	6	3	4	0
Level 2	0	5	8	7	3
Level 3	1	4	9	3	2
Level 4	0	3	4	8	1
Level 5	2	1	2	4	10

Precision and *recall* metrics are also useful when it comes to evaluating the performance of a model in a medical scenario [30] [16]. We understand *precision* as the capability of an algorithm to identify only samples of interest and *recall* as the ability of a classification model to identify all relevant examples. The corresponding evaluations are given by the formulas from equation (2).

$$Precision = \frac{TP}{TP + FP} \text{ and } Recall = \frac{TP}{TP + FN} \tag{2}$$

A model that has a very high *recall* has very less *precision*.

In other words, *recall* describes our TPs from the predicted results, whereas *precision* indicates our TPs from the actual results. Also, *recall* is the equivalent of the TPR parameter of the ROC curve. If we are interested to see the *recall* score, we should ask the right question: out of everything predicted, what is the ratio of missing *Level 2* patients? Same with *precision*: out of all *Level 2* patients, we predicted what is the ratio of *Level 2* patients when they did not represent a *Level 2* risk. A low *precision* means overuse of resources and also overcrowding. A low *recall* means that a patient should have been classified with a *Level 2* emergency code, but another code was assigned to them. Now, overestimating the health status does not necessarily imply a tremendous negative outcome, but again it will lead to overusing resources. Instead, if the patient's health condition is underestimated and a lower code is assigned, this could increase the mortality of the patient. In addition, *precision* and *recall* are better for highly imbalanced datasets [31].

In the medical world, any test for diagnosing people with a disease or not should recognize *sensitivity* and *specificity* metrics. *Sensitivity* measures how often people who have the disease get the result positive, and *specificity* measures the ability to detect negative results for cases which indeed have negative results. We know how bad the first scenario is, with no need for further explanations. In the second case, people who do not have the disease will undergo unnecessary testing or even treatments [32] [28]. In binary classification, *recall* is called *sensitivity*.

Specificity is another metric which can be calculated using the confusion matrix values, by the formula from equation (3).

$$Specificity = \frac{TN}{TN + FP} \tag{3}$$

4. Computational experiments and results

For this study, we used an NN-Sequential Model, a Random Forest Tree and a Logistic Regression Algorithm to test the results and choose the one with good performance from the medical point of view. We have evaluated the performance of the models on the given set of data in three ways: using the imbalanced data, handling imbalanced and reducing the volume of the original data. For handling imbalanced data, we did two experiments using two different sampling techniques: SMOTE and ADASYN.

Each Machine Learning model uses different hyper-parameters tuned after an empirical analysis. We conducted many experiments by tuning hyper-parameters, following the concept of trial and error. The final hyperparameters are presented in Table 3. The Logistic Regression algorithm required a 'multinomial' option and 'lbfgs' solver due to the multi-class classification problem. For the NN-Sequential model, we used 'categorical_crossentropy' as a loss function. For the optimizer parameter, we chose an SGD object initialized with the following values: lr = 0.01, decay = 1e-6, momentum = 0.9, nesterov = True. The model was trained on 100 epochs and used a batch size of 128 together with a callback parameter that monitored the accuracy. The network had 4 hidden layers, one input layer and one output layer. The first fifth layers had a *relu* activation function and 30 nodes, and the output layer used a *softmax* activation function and 5 nodes. The input layer received the features' number. For the last algorithm, we chose 50 estimators, meaning there would be 50 decision trees.

In terms of time and space complexity, the Logistic Regression model has a train time complexity of $O(n * m)$, the test time complexity is $O(m)$, and the space complexity of the algorithm is $O(m)$, where n is the number of training data and m is the number of features from the considered dataset.

For the Random Forest Tree approach, we have a training time complexity of $O(k * n * \log n * m)$, the test time complexity is $O(m * k)$, and space complexity is $O(k * depthOfTree)$, where n is the number of training examples, m is the number of features from the considered dataset, and k is the number of decision trees. Random Forest is comparatively faster than other algorithms.

For NN Sequential model, we used the in-build *Sequential() API* from *Keras* library. This is a feed-forward neural network that transfers the data from one layer to another until all sequential steps are finished. The space memory should be equal to the sum of all weights created by the network and batch normalization parameters or other specified hyper-parameters. The total number of parameters used by a neural network could be easily found using the function *summary()*, as we did. Overall, the neural network used in this paper has 12780 parameters after the input layer, the next fourth hidden layers have 930 parameters, and the output layer has 155 parameters. As long as the algorithm is coming from standard libraries, the time complexity of it is difficult to calculate, and, moreover, the result could be inaccurate.

Table 3
Models hyper-parameters.

Model	Hyper-parameter	Value
Logistic Regression	multi_class	multinomial
	solver	lbfgs
	C	10
NN-Sequential	loss	categorical_crossentropy
	optimizer	SGD
	metrics	accuracy
	epochs	100
	batch_size	128
	callback	EarlyStopping
Random Fores Tree	n_estimators	50

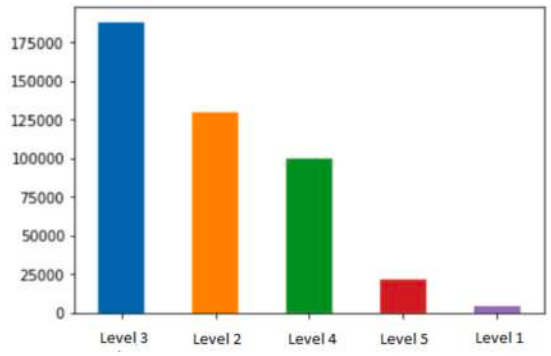


Fig. 5. The distribution of target classes.

The software and libraries used for the following experiments are Python 3.10.6, Jupyter 1.0.0, pandas 2.0.0, numpy 1.23.5, sklearn 0.0.post1, keras 2.12.0, imblearn 0.0.

4.1. Imbalanced data

The distribution of examples across the classes reveals the skewed levels, and they are represented in Fig. 5. Imbalanced data may affect the performance of a Machine Learning model, making a poor prediction. Imbalanced data represent a challenge, especially when it comes to a multi-class classification problem, because the model is prone to perform poorly on the majority class, trying to have better results on minority class [33].

The majority class of our model is represented by *Level 3* emergency code, and the values are decreasing for the other classes. The large amount of examples for *Level 3* is normal because in real life, statistically, *Level 3* is the most frequent emergency code. Here are the numbers for each of the classes: (1.0, 4265), (2.0, 130121), (3.0, 188245), (4.0, 99525), (5.0, 22197). The number of *Level 3* is huge compared to the number of examples of *Level 1*. *Level 2* is closer to *Level 3*, and, sometimes, a given object may be a borderline between these two classes.

Further, the performance of the models is explained starting with the imbalanced dataset and maintaining the same 80%-20% ratio for the train-test split during the experimental evaluations. Our *Experiment 1* is the Neural Network approach that uses a Sequential model with 6 layers provided by *Keras* library. Each of them used, as described at the beginning of Section 4, a *Relu* activation function except the last layer, which used *SoftMax* as an activation function because the target variable has more than one class. Python *SGD* class was used as an optimizer, and the metric was *accuracy*. We trained the model using 100 epochs but also used a callback function to stop the training when the accuracy was dropping. The model stopped training after 41 epochs, having a loss of 0.76 and an accuracy of 0.67.

Fig. 6 displays the confusion matrix after the model was trained and the test set was predicted. The specific results for the *Experiment 1* are outlined in Table 4. *Level 1* has a low *recall* (0.18), which means the FN rate is high. A low *recall* means that 794 patients who needed to be assigned to a *Level 1* emergency code were classified on the lower levels, and only 183 were classified correctly. On the other hand, the *precision* is high (0.71). If the *precision* is high, the FP rate is lower. Hence, out of all *Level 1* predictions, 0.71 were indeed *Level 1*. The *specificity* for *Level 1* is almost 0.99. Therefore, the patients who did not need to be assigned on *Level 1* were correctly assigned to other levels. *Level 2, 3* and *4* have the best scores for *precision* and *recall*.

Fig. 6. NN Sequential Model - Confusion Matrix.

Table 4
NN Sequential Model - measurements per classes.

Classes	Recall	Precision	Specificity	ROC-AUC
Level 1	0.18	0.71	0.99	0.59
Level 2	0.65	0.69	0.92	0.76
Level 3	0.71	0.65	0.72	0.71
Level 4	0.67	0.65	0.89	0.78
Level 5	0.29	0.54	0.97	0.64

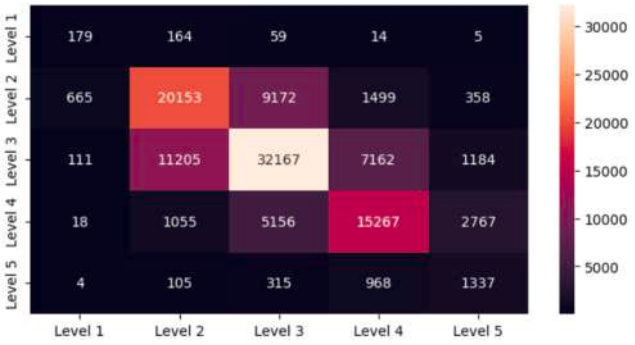


Fig. 7. RFT - Confusion Matrix.

Table 5
RFT - measurements per classes.

Classes	Recall	Precision	Specificity	ROC-AUC
Level 1	0.18	0.42	0.99	0.59
Level 2	0.61	0.63	0.85	0.73
Level 3	0.68	0.62	0.69	0.69
Level 4	0.61	0.62	0.89	0.75
Level 5	0.23	0.48	0.98	0.61

Although *Level 3* has good *precision* and *recall* scores, the *specificity* is not that high. The algorithm is best at distinguishing emergency codes of type 4 with a score of 0.78. *Level 2, 3*, and *4* again have the best scores overall. The confusion matrix shows that *Level 2* and *Level 3* are easily confused.

For our *Experiment 2* involving the Random Forest Tree classifier, we have used the Python *GridSearchCV* function to find the best estimators, and the result was to use 50 estimators. An estimator is a decision tree in the forest. The confusion matrix after training this Random Forest Tree model is presented in Fig. 7.

The specific values from Table 5 yield that this model is not performing as well as the previous one. The *recall* for *Level 1* is still 0.18, but the *precision* is 0.42. There are lower *precision* and *recall* scores for each class compared with the Neural Network measurements.

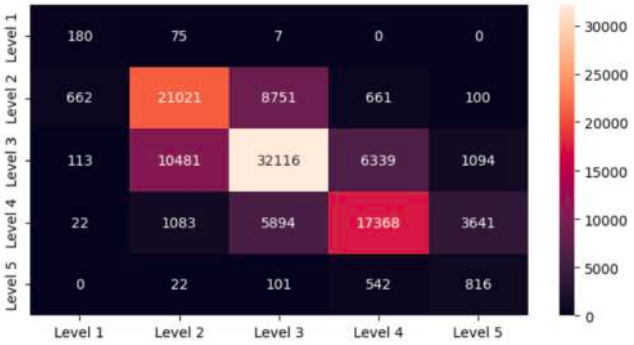


Fig. 8. LR - Confusion Matrix.

Table 6
LR - measurements per classes.

Classes	Recall	Precision	Specificity	ROC-AUC
Level 1	0.18	0.68	0.99	0.59
Level 2	0.64	0.67	0.87	0.75
Level 3	0.68	0.64	0.78	0.70
Level 4	0.69	0.62	0.89	0.78
Level 5	0.14	0.55	0.99	0.56

Looking only at the *specificity*, one would say that this model is very good, but in fact, it is misleading. The ROC-AUC curve is also lower compared to the first measurements. For *Level 3*, the *specificity* is lower, so the model falsely assigns patients on *Level 3*. Looking back to Table 4, we conclude that NN Sequential Model had better scores.

Our *Experiment 3* represents the last model we have considered here. It uses the Logistic Regression algorithm for which we set the parameter *multi_class* to *multinomial* because of the nature of the problem, and we have used *lbfgs* as a solver that handles *L2* or no penalty. *L2* represents regularization added to the algorithm to prevent overfitting. The corresponding confusion matrix is presented in Fig. 8 and we can notice in Table 6 the performance of the model.

Out of all three models, the *Level 4* class has the lowest *recall* score using this particular model, whereas the others are slightly changed. The ROC-AUC curve is similar to the one produced by the Random Forest Tree classifier, and the *precision* score is one percent less than the one produced by the Neural Network. The overall average per measurement is slightly behind the overall measurements from the first model except the *specificity* with an average of 0.9. The NN model had the best results.

The *recall* score for the emergency code of type *Level 1* is always the same (0.18). All three classifiers failed in assigning this type of emergency code to a large number of patients who really needed it. *Level 1* class represents the minority class therefore the low *recall*. Both this metric and *precision* are better to be studied for highly imbalanced datasets [31].

4.2. SMOTE for imbalanced data

Imbalanced data is a great challenge for building a model with considerable performance. The distribution of classes is important, and the difference between them affects the outcome of the algorithm. Following the data provided in the previous section in Fig. 5, the target classes are distributed as percentages in this way: 1% for *Level 1*, 29,% for *Level 2*, 42% for *Level 3*, 23% for *Level 4* and 5% for *Level 5*. It follows that *Level 1* and *Level 5* are missing enough information, and the distribution is skewed. Usually, imbalanced data is given, but having a multi-class classification with imbalanced data is even more daunting.

A popular way to treat this unequal distribution of classes is using SMOTE (*Synthetic Minority Over-sampling Technique*) [34]. Of course, there are other approaches, like under-sampling the majority class, but in this case, the model may lack useful examples. One can choose to use random over-sampling, but this strategy comes with its drawbacks. Random over-sampling simply recreates examples for the minority class without adding variety or useful information about those new objects, but it will increase the likelihood of the model overfitting. At the same time, SMOTE adds new samples in the minority class generating synthetic data.

The algorithm behind SMOTE implies searching for K-nearest neighbours of each minority item, then one of these neighbours will be randomly selected and, using linear interpolation, a new minority instance is produced.

SMOTE will target the minority class to generate synthetic points to almost match the number of data points in the majority class. Now as the synthetic data being generated is chosen from lines connecting the existing points (feature data) so we are basically incorporating less noise. Thus reducing the chances of overfitting. Generally, SMOTE is used to overcome the problem of overfitting, but it might also lead to overfitting as it is synthetic data and not real data.

The technique specified by SMOTE can be easily applied to a multi-class problem because the algorithm identifies the minority class against the remaining ones with the approach One-versus-All. Python has a support library for SMOTE algorithm, but the drawback is that it can be used only with continuous features, whereas the dataset used in this study is a mix of categorical and

Table 7
LR- SMOTE - measurements per classes.

Classes	Recall	Precision	Specificity	ROC-AUC
Level 1	0.42	0.08	0.95	0.69
Level 2	0.55	0.65	0.87	0.71
Level 3	0.59	0.67	0.79	0.69
Level 4	0.64	0.59	0.87	0.75
Level 5	0.42	0.24	0.97	0.67

Table 8
NN Sequential - SMOTE - measurements per classes.

Classes	Recall	Precision	Specificity	ROC-AUC
Level 1	0.35	0.14	0.98	0.66
Level 2	0.65	0.66	0.96	0.75
Level 3	0.63	0.68	0.78	0.70
Level 4	0.67	0.62	0.89	0.78
Level 5	0.39	0.33	0.95	0.67

Table 9
RFT - SMOTE - measurements per classes.

Classes	Recall	Precision	Specificity	ROC-AUC
Level 1	0.24	0.16	0.98	0.61
Level 2	0.60	0.58	0.74	0.71
Level 3	0.57	0.63	0.75	0.66
Level 4	0.61	0.54	0.85	0.73
Level 5	0.29	0.31	0.96	0.63

continuous attributes. Thus, we have used SMOTE-NC (SMOTE for *Nominal and Continuous features*) instead, and we have specified which are the *Categorical* features so that the SMOTE algorithm would resample these values instead of generating synthetic data. After synthetic samples were produced, the distribution was completely changed, and the amount of data was substantially increased to 188242 for each level.

All three models were re-evaluated using the same metrics to have an overview and to continue the study because we wanted to see how the models performed with the new samples. All the hyper-parameters were kept constant.

Again, the Neural Network model had a better performance in terms of ROC-AUC score than the previous Neural Network (Table 4), with an average of 0.71. The *recall* and *precision* of *Level 2, 3* and *4* were close like usual. The confusion matrix showed that most patients who were classified as non *Level 2* when they should have been assigned with that code were on *Level 3* and *4*. For *Level 3*, many patients were spread on *Level 2* and *Level 4*.

Out of all three models, *Levels 1,3,4* and *5* had the lowest *recall* score using Random Forest Tree model. Also, the ROC-AUC scores were lower than the previous ones depicted in Table 5. If we calculate the average on the *recall* metric, the result is the same as the one from the other RFT classifier.

In the end, according to Logistic Regression measurements, there was a higher *recall* for *Level 1* and *Level 5* comparing the results to the ones from Table 6. This time, a lot of patients were misclassified as high-risk (*Level 1*) patients. The ROC-AUC score was better than the previous Logistic Regression model using imbalanced data, but the *recall* and *precision* were lower.

In Machine Learning, more data usually means better predictions, but in the previous tests, it is clear that the differences are not that significant. Hence, we further check if we can get better predictions by augmenting the given dataset and we organize the results in Table 7, Table 8, and Table 9, respectively.

4.3. ADASYN for imbalanced data

ADASYN (Adaptive Synthetic) is an improved sampling technique than SMOTE, by reducing the bias of skewed data and sampling more data for the highly imbalanced classes that can be tough to learn by a model [35].

After we applied the ADASYN method to our imbalanced data, the number of samples was increased on each emergency level, but there were some minor differences between them as described in Fig. 9. In SMOTE case, the number of samples was the same for each class.

We evaluated all three models using that new dataset. The results were not far from the SMOTE experiment but slightly better.

Based on the results from Table 10, using the Logistic Regression model as in the case of SMOTE experiment, for *Level 1* and *Level 5* the emergency codes still had a high recall and a low precision, meaning the predictions were incorrect for this class.

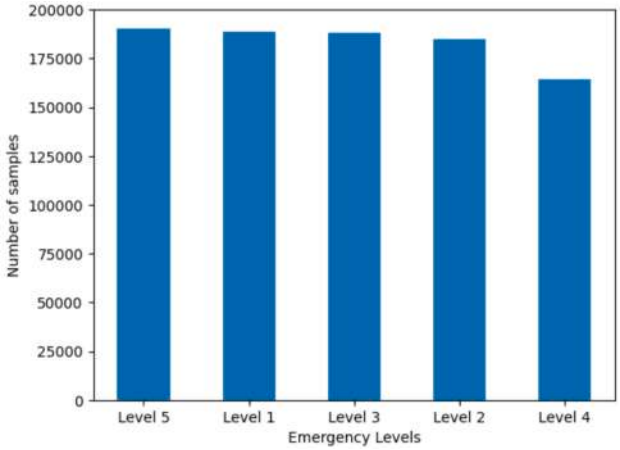


Fig. 9. ADASYN SAMPLING.

Table 10
LR- ADASYN - measurements per classes.

Classes	Recall	Precision	Specificity	ROC-AUC
Level 1	0.73	0.05	0.89	0.81
Level 2	0.49	0.60	0.86	0.68
Level 3	0.52	0.71	0.84	0.68
Level 4	0.53	0.62	0.90	0.72
Level 5	0.68	0.23	0.88	0.78

Table 11
NN Sequential - ADASYN - measurements per classes.

Classes	Recall	Precision	Specificity	ROC-AUC
Level 1	0.47	0.14	0.92	0.72
Level 2	0.66	0.64	0.84	0.75
Level 3	0.55	0.71	0.83	0.69
Level 4	0.58	0.62	0.89	0.74
Level 5	0.65	0.26	0.90	0.78

Table 12
RFT - ADASYN - measurements per classes.

Classes	Recall	Precision	Specificity	ROC-AUC
Level 1	0.25	0.27	0.99	0.62
Level 2	0.62	0.59	0.81	0.72
Level 3	0.61	0.62	0.73	0.67
Level 4	0.59	0.59	0.88	0.73
Level 5	0.31	0.41	0.97	0.64

In the case of the NN Sequential Model (see Table 11), there was an improved score of ROC-AUC, precision and recall for levels 2, 3, and 4. However, the specificity was lower.

After evaluating the Random Forest Tree model, we observe in Table 12 that ROC-AUC for levels 2, 3, and 4 was even lower, meaning the model was not capable of distinguishing between classes correctly.

4.4. The simplified problem

In search of even better predictions, another approach is considered here. Instead of having a 5-level classification, we have assumed a 3-level classification problem. We simplified the problem to the point where the model would predict only three emergency situations by grouping *Level 1* and *Level 2* into one class called `Critical`, *Level 3* standing to represent the `Urgent` class and grouping again *Level 4* and *Level 5* as `Non-urgent` class. Basically, everything above *Level 3* (higher levels) represents a critical situation which,

(a) Simplified problem - NN Sequential model - confusion matrix

(b) Simplified problem - RFT model - confusion matrix

(c) Simplified problem - LR model - confusion matrix

Fig. 10. Simplified problem - models - confusion matrix.

indeed, has to have maximum priority, and what is below this level (lower levels) represents a non-urgent case. In this manner, the classes were distributed naturally, following the organic meaning. The distribution of the new target features resulted as follows: `Critical` - 30%, `Urgent` - 43%, `Non-urgent` - 27%.

For the interest of this study, we kept the previous three models and the metrics as well for experiments in order to see how this perspective changes the algorithms' performance. The corresponding confusion matrix is resumed in Fig. 10 for all models, and Table 13 outlines the related measurements for the selected metrics.

Experiment 1 was an NN-Sequential approach providing that, for patients with an urgent emergency, the number of TP is 33158, whereas 13711 were wrongly assigned to another class. The amount of FN is pretty high. Looking at the measurements presented in Table 13, the *recall* of the `Urgent` class is slightly better than `Critical` one, but there is a lot of confusion between these two classes. The `Non-urgent` class has the best *recall*, followed by the `Urgent` one, which is the majority class. Although the `Critical`

Table 13
Simplified problem - measurements per classes.

Model	Class	Recall	Precision	Specificity	ROC-AUC
NN-Sequential	`Critical`	0.63	0.73	0.89	0.76
	`Urgent`	0.70	0.66	0.73	0.72
	`Non-urgent`	0.78	0.75	0.90	0.84
Random Forest Tree	`Critical`	0.55	0.70	0.89	0.72
	`Urgent`	0.75	0.58	0.61	0.67
	`Non-urgent`	0.62	0.78	0.93	0.77
Logistic Regression	`Critical`	0.29	0.21	0.51	0.40
	`Urgent`	0.19	0.28	0.63	0.41
	`Non-urgent`	0.76	0.73	0.69	0.83

class has the lowest *recall*, the *precision* is high. The ROC-AUC shows that the model is performing very well when predicting the `Non-urgent` class. The model predicted 78% of patients to be assigned on the `Non-urgent` class with a *precision* of 75% and predicted 90% as true negatives.

Experiment 2 applies the Random Forest Tree algorithm with the same number of estimators. In this case, the `Urgent` class has a better TP value than the one generated by the NN-Sequential, but the values for the other two classes decreased. The *recall* is higher for the `Urgent` class than the others, but the *precision* and *specificity* are low, even if the ROC-AUC shows the model is not distinguishing very well this class in this case. The others have low *recall* but better results for the other metrics. We may conclude that the Neural Network for the simplified problem has better scores overall.

The last one, *Experiment 3*, is again based on the Logistic Regression model. Its confusion matrix included in Fig. 10 indicates a lot of FNs and TPs for the first two classes. Looking also at Table 13, this approach has the lowest scores, and it is performing poorly compared to the other two algorithms. Even if the `Non-urgent` class has good scores here, the highest risk class, the `Critical` one, for example, is more important and has the lowest scores.

Considering all of these measurements made using the given dataset with different distributions of classes shows that sometimes a specific model is performing well and other times not. Analysing different measurements, the dataset should have more significant examples so the models can better distinguish the defined classes `Critical`, `Urgent` and `Non-urgent`, which are easily confused. Looking only at the *recall* score, the NN-Sequential algorithm performed best in all cases.

Emergency care is an important decision point in a hospital, and the solutions provided by Artificial Intelligence could solve many challenges [4]. This is the reason there are papers in the literature that use different strategies or other Machine Learning algorithms applied in emergency departments [36]. For example, the aim of paper [11] (KTAS) was to analyse different models on predicting "Korean Triage and Acuity Scale [11]" levels. In that paper, Logistic Regression, Random Forest Tree and XGBoost algorithms were evaluated both on clinical data and text data using NLP (Natural Language Processing). The authors also integrated NLP in [14] (KATE) for achieving better prediction values, and in this particular study, only XGBoost algorithm was evaluated. Another approach to the emergency room department is to predict the patient's admission like researchers of this paper [21] (ED-Admission) did.

For the purpose of creating a context, the models used in this study are compared with the ones used in the papers mentioned above. To have a diversity of techniques, the models trained on the simplified problems were used for this comparison, calculating an average on each metric, with the claim that this study is notated as eUPU. For KTAS we chose the measurements evaluated on all data as well for KATE ("pediatric and adult patients in the gold set [14]" were used in the latter case). The ED-Admission paper studied a binary problem, and we chose the evaluation of the models on all data. The diversity of algorithms and metrics, as well as scores, is represented in Table 14.

The classification performance and inferences of Machine Learning models could be improved by applying other methods and approaches, such as information on hyper-parameter calibration to make models more extensible and different methods to identify predictors that could affect the triage workflow. Modelling this problem is not the most complex and therefore, more powerful models can be proposed for this study (e.g. XGBoost [11], [14]). Also, uncertainty handling methods would be of interest in this particular case.

4.5. Memory and CPU usage

When dealing with Machine Learning algorithms, memory and CPU usage need to be evaluated. Although the accuracy of a model requires the main focus, execution time and memory increment should be also considered and evaluated.

For this topic, we utilized specific Python commands that are built into IPyhton that runs on Jupyter Notebook. To evaluate the CPU usage of *fit* and *predict* we used the command *%%time*, and to evaluate the memory usage, we used *%%memit*, which required a specific package, *memory_profiler*.

According to Fig. 11 and Fig. 12, on one hand, Logistic Regression used on samples made by SMOTE technique had a significant memory increment when fitting the training data. On the other hand, the NN-Sequential model required more CPU usage than the others. In that particular case, the dataset resulting from ADASYN technique had the highest execution time. When the *predict* was executed, models did not require as much memory and the execution time was fast.

Table 14
Models and metrics comparison.

Study	Model	Metric	Score
eUPU	NN-Sequential	ROC-AUC	0.76
		Precsion	0.70
		Recall	0.69
		Specificity	0.84
	Logistic Regression	ROC-AUC	0.54
		Precision	0.65
		Recall	0.41
		Specificity	0.61
	Random Forest Tree	ROC-AUC	0.72
		Precision	0.68
		Recall	0.64
		Specificity	0.81
KTAS	Logistic Regression	Precision	0.71
		Recall	0.70
		F1-score	0.70
		AUROC	0.90
	Random forest	Precision	0.73
		Recall	0.73
		F1-score	0.70
		AUROC	0.92
	XGBoost	Precision	0.75
		Recall	0.75
		F1-score	0.74
		AUROC	0.92
KATE	XGBoost	AUC	0.84
		F1-score	0.73
		Sensitivity	0.69
		Precision	0.80
ED-Admission	Logistic Regression	AUC	0.90
		Sensitivity	0.80
		Specificity	0.85
		PPV	0.69
		NPV	0.91
	XGBoost	AUC	0.92
		Sensitivity	0.83
		Specificity	0.85
		PPV	0.70
		NPV	0.92
	DNN	AUC	0.92
		Sensitivity	0.82
		Specificity	0.85
		PPV	0.70
		NPV	0.92

5. Interfacing by the web-based application

The challenging research for finding a Machine Learning model well performing in emergency department triage flow led us to develop an interface between the model and the clinician as the end-user. A series of software resources have been selected for this approach, successfully guided by the Python [17] ability to export the Machine Learning model as an API (*Application Programming*

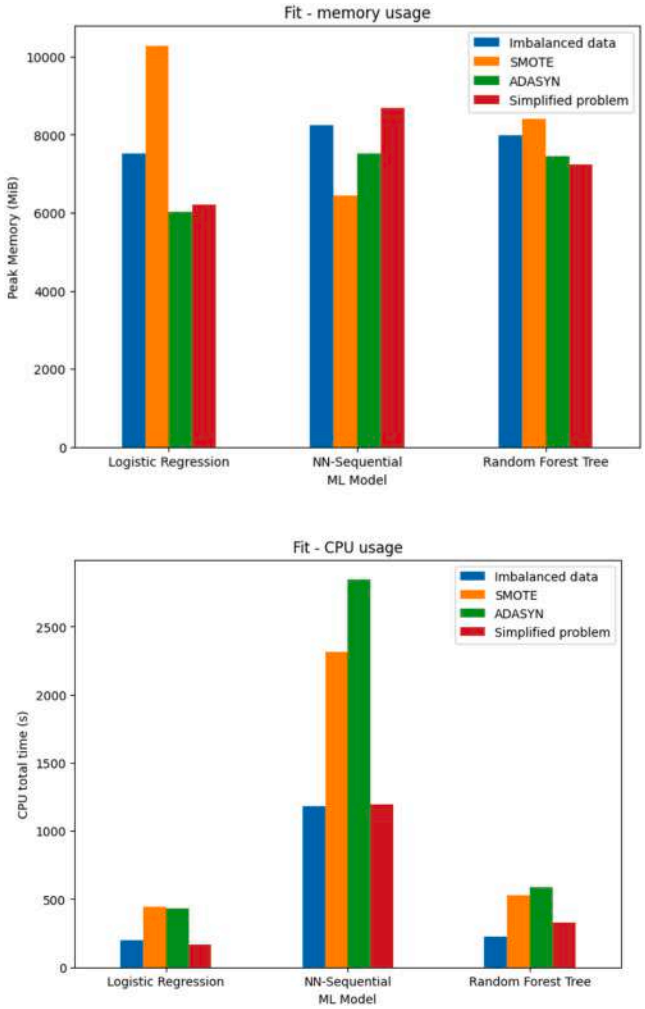


Fig. 11. Fitting data - Memory and CPU usage.

Interface) using Flask [37]. The end-user interface was built with React technology [38] for getting the predictions of the trained model using the API provided by Flask, following the application architecture represented in Fig. 13.

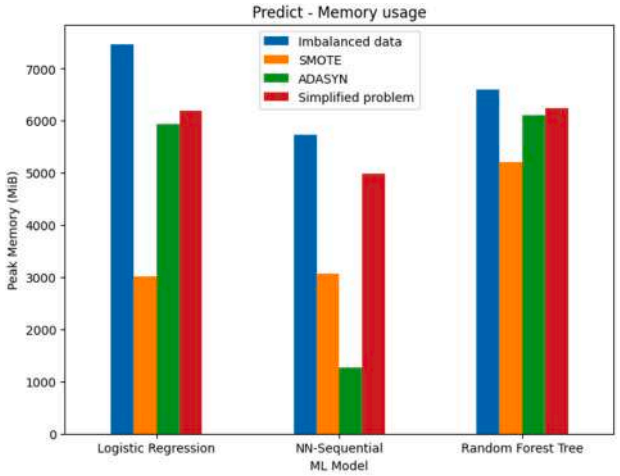
The Machine Learning model was saved using the Python *joblib* package and imported as an instance in Flask app. We chose to integrate the NN-Sequantial model trained on the simplified problem due to average scores described in Table 14. The same preprocessing transformers described in Section 3 and Section 4 have been applied and after the appropriate dataset was preprocessed, the imported classifier would make predictions. Both the simple data flow and the user-friendly interface facilitate the interaction with clinicians, keeping the main focus on the performance of the used Machine Learning algorithm.

Making the ML-based application available and applicable to the various domains allows people with high expertise in those domains, here medical care, to accept the IT tools for supporting a more reliable clinical decision.

6. Discussion and conclusions

This research study focuses on a Machine Learning approach to assign emergency codes to each patient after triage. We demonstrated the potential of an automatic classification system using supervised models to enhance early medical diagnosis and facilitate medical record processing. AI applications could significantly enhance the efficiency of the emergency triage process. Emergencies need to be prioritized, so studying these models in real-time and evaluating the dynamic power of machine learning for classifying which emergency code should a patient receive is entirely justified.

Both the Machine Learning algorithms and the medical domain turned out to be more challenging than expected. Medical data is not easy to find, and it has to be customized by country, e.g. Romanian patient file differs from the one used in the USA. Therefore, we also targeted to adjust the dataset to the Romanian medical care system. Also, we had to take care of missing data, and we were surprised to see how imbalanced the classes were, but the aim was to study the dataset and see how it could be applied in this



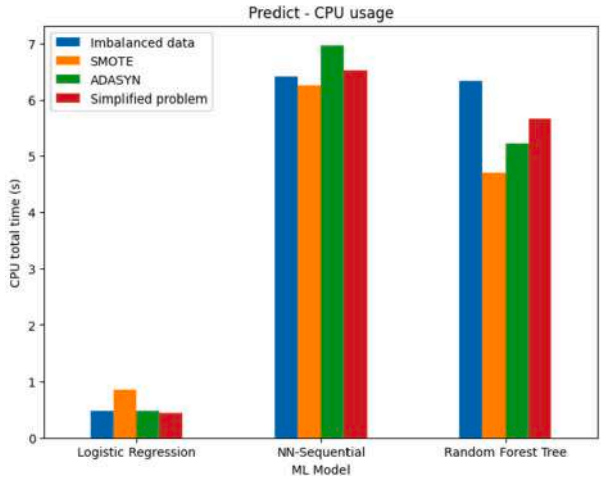


Fig. 12. Predicting data - Memory and CPU usage.

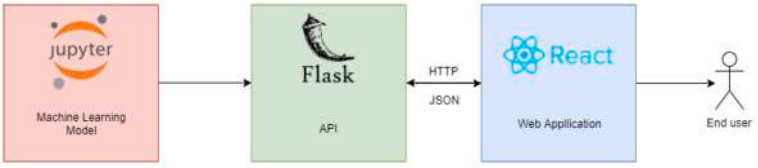


Fig. 13. Application Architecture.

environment. Eventually, we simplified the problem due to imbalanced data and kept the algorithm with the best results. After our research, the experimental results demonstrated that the NN-Sequential algorithm performed better on the simplified problem.

Without any doubt, there are opportunities to improve this study, and there is further work to do, but for this, more data needs to be collected. Besides this, good quality feedback from experts in that particular domain is needed, and this can be achieved by our web application, where we use the best classifier as an API. The tool itself proves easy to use as an interface for patient overview and early monitoring, and our findings indicate that clinicians were able to reliably use the tool. In the end, having impressive accuracy in predicting emergency codes, a digital smart patient file could be created.

Ethics statement

We hereby confirm that our study complies with all regulations and we correspondingly cited as [21] the source of our dataset with respect to the original source [20] that mentions: *We provide the de-identified dataset and R scripts for the paper "Predicting hospital admission at emergency department triage using machine learning". All processing scripts in the Scripts/R/ subdirectory take as input .csv files extracted from the enterprise data warehouse using SQL queries. The analysis scripts in the main directory take as input the de-identified*

dataset provided in this repository. The working directory should be set to the main directory with the analysis scripts. All research using this dataset should cite the original paper. "Hong WS, Haimovich AD, Taylor RA (2018) Predicting hospital admission at emergency department triage using machine learning. PLoS ONE 13(7): e0201016.".

CRediT authorship contribution statement

Andreea Vantu: Conceived and designed the experiments; Performed the experiments; Analyzed and interpreted the data; Contributed reagents, materials, analysis tools or data; Wrote the paper.

Anca Vasilescu: Conceived and designed the experiments; Performed the experiments; Contributed reagents, materials, analysis tools or data; Wrote the paper.

Alexandra Baicoianu: Conceived and designed the experiments; Analyzed and interpreted the data; Contributed reagents, materials, analysis tools or data; Wrote the paper.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data included in article/supp.material/referenced in article.

References

[1] E. Alpaydin, Introduction to Machine Learning, fourth edition, Adaptive Computation and Machine Learning Series, MIT Press, 2020.
[2] J. Alzubi, A. Nayyar, A. Kumar, Machine learning from theory to algorithms: an overview, in: Second National Conference on Computational Intelligence, NCCI 2018, 5 December 2018, Bangalore, India, J. Phys. Conf. Ser. 1142 (012012) (2018), https://doi.org/10.1088/1742-6596/1142/1/012012.
[3] P. Sharma, Z. Said, A. Kumar, S. Nizetic, A. Pandey, A. Hoang, Z. Huang, A. Afzal, C. Li, T. Le Anh, X.P. Nguyen, V. Tran, Recent advances in machine learning research for nanofluid-based heat transfer in renewable energy system, Energy Fuels 36 (13) (2022) 6626–6658, https://doi.org/10.1021/acs.energyfuels.2c01006.
[4] Kenneth Jian Wei Tang, Candice Ke En Ang, T. Constantinides, V. Rajinikanth, U. Rajendra Acharya, K.H. Cheong, Artificial intelligence and machine learning in emergency medicine, Biocybern. Biomed. Eng. 41 (1) (2021) 156–172, https://doi.org/10.1016/j.bbe.2020.12.002.
[5] R. Sánchez-Salmerón, J.L. Gómez-Urquiza, L. Albendín-García, M. Correa-Rodríguez, M.B. Martos-Cabrera, A. Velando-Soriano, N. Suleiman-Martos, Machine learning methods applied to triage in emergency services: a systematic review, Int. Emerg. Nurs. 60 (2022) 101109, https://doi.org/10.1016/j.ienj.2021.101109.
[6] C.K. Wee, X. Zhou, R. Sun, R. Gururajan, X. Tao, Y. Li, N. Wee, Triaging medical referrals based on clinical prioritisation criteria using machine learning techniques, Int. J. Environ. Res. Public Health 19 (12) (2022) 7384, https://doi.org/10.3390/ijerph19127384.
[7] G.D. Magoulas, A. Prentza, Machine learning in medical applications, in: G. Paliouras, V. Karkaletsis, C.D. Spyropoulos (Eds.), Machine Learning and Its Applications, ACAI 1999, in: Lecture Notes in Computer Science, vol. 2049, Springer, Berlin, Heidelberg, 2001.
[8] M. Shehab, L. Abualigah, et al., Machine learning in medical applications: a review of state-of-the-art methods, Comput. Biol. Med. 145 (June 2022), https://doi.org/10.1016/j.compbiomed.2022.105458, 2022.
[9] A. Mujumdar, V. Vaidehi, Diabetes prediction using machine learning algorithms, Proc. Comput. Sci. 165 (1) (2019) 292–299, https://doi.org/10.1016/j.procs.2020.01.047.
[10] S. Levin, M. Toerper, et al., Machine-learning-based electronic triage more accurately differentiates patients with respect to clinical outcomes compared with the emergency severity index, Ann. Emerg. Med. 71 (5) (September 2017), https://doi.org/10.1016/j.annemergmed.2017.08.005, 2017.
[11] S.W. Choi, T. Ko, K.J. Hong, K.H. Kim, Machine learning-based prediction of Korean triage and acuity scale level in emergency department patients, J. Phys. Conf. Ser. 25 (4) (2019) 305–312, https://doi.org/10.4258/hir.2019.25.4.305.
[12] A. Vântu, Using technology to improve the emergency room triage, Bachelor Thesis supported by Siemens Romania SRL under the contract BCT 25101/10.04.2017, Transilvania University of Braşov, Romania, June 2018.
[13] A. Vântu, A. Vasilescu, e-UPU: using technology to improve the emergency room triage, in: Poster Session, 7th ACM Celebration of Women in Computing: womENcourage 2020, ADA University, Baku, Azerbaijan, 24-27 September 2020, also Available Online, 2020.
[14] O. Ivanov, L. Wolf, D. Brecher, E. Lewis, K. Masek, K. Montgomery, Y. Andrieiev, M. McLaughlin, S. Liu, R. Dunne, K. Klauer, C. Reilly, Improving ED emergency severity index acuity assignment using machine learning and clinical natural language processing, J. Emerg. Nurs. 47 (2) (2021) 265–278, https://doi.org/10.1016/j.jen.2020.11.001.
[15] N. Gilboy, et al., Emergency Severity Index (ESI) - a Triage Tool for Emergency Department Care, Implementation Handbook, Version 4, ENA Emergency Nurses Association, available online (2020).
[16] H. Harvey, How data scientists can convince doctors that AI works, available online.
[17] Python 3.9.1 documentation, https://docs.python.org/release/3.9.1/. (Accessed 26 August 2022).
[18] D.N. Lipe, S.S. Bourenane, M.K. Wattana, S. Gaeta, P. Chaftari, M.T. Cruz Carreras, J.-G. Manzano, C. Reyes-Gibby, A modified emergency severity index level is associated with outcomes in cancer patients with COVID-19, Am. J. Emerg. Med. 54 (2022) 111–116, https://doi.org/10.1016/j.ajem.2022.02.002.
[19] M.H. Yarmohammadian, F. Rezaei, A. Haghshenas, N. Tavakoli, Overcrowding in emergency departments: a review of strategies to decrease future challenges, J. Res. Med. Sci. 22 (2017) 23, https://doi.org/10.4103/1735-1995.200277.
[20] W.S. Hong, A.D. Haimovich, A.R. Taylor, Predicting hospital admission at emergency department triage using machine learning, https://github.com/yaleemmlc/admissionprediction, dataset available from 6 Sep 2018.
[21] W.S. Hong, A.D. Haimovich, R.A. Taylor, Predicting hospital admission at emergency department triage using machine learning, PLoS ONE 07 (13) (2018) 1–13, https://doi.org/10.1371/journal.pone.0201016.
[22] A. Géron, Hands on Machine Learning with Scikit-Learn, Keras, and TensorFlow, second edition, O'Reilly Media Inc., USA, 2019.
[23] scikit-learn Homepage, available online (Accessed 26 August 2022).
[24] E.C. Zabor, C.A. Reddy, R.D. Tendulkar, S. Patil, Logistic regression in clinical studies, Int. J. Radiat. Oncol. Biol. Phys. 112 (2) (2022) 271–277, https://doi.org/10.1016/j.ijrobp.2021.08.007.

[25] E. Boateng, D. Abaye, A review of the logistic regression model with emphasis on medical research, J. Data Anal. Inf. Process. 7 (2019) 190–207, https://doi.org/10.4236/jdaip.2019.74012.
[26] S. Dreiseitl, L. Ohno-Machado, Logistic regression and artificial neural network classification models: a methodology review, J. Biomed. Inform. 35 (5–6) (2002) 352–359, https://doi.org/10.1016/S1532-0464(03)00034-0.
[27] M. Fernandez-Delgado, E. Cernadas, S. Barro, D. Amorim, Do we need hundreds of classifiers to solve real world classification problems?, J. Mach. Learn. Res. 15 (2014) 3133–3181, https://dl.acm.org/doi/10.5555/2627435.2697065.
[28] A. Lekhtman, Data Science in Medicine — Precision & Recall or Specificity & Sensitivity? Available online (Accessed 26 August 2022).
[29] D.G. Levy, In Machine Learning Predictions for Health Care the Confusion Matrix is a Matrix of Confusion, available online (Accessed 26 August 2022).
[30] B. Bowers, Triage to AI: a Machine Learning Approach to Hospital Admissions Classification, available online (Accessed 26 August 2022).
[31] S.E. Awan, M. Bennamoun, F. Sohel, F.M. Sanfilippo, G. Dwivedi, Machine learning-based prediction of heart failure readmission or death: implications of choosing the right model and the right metrics, ESC Heart Fail. 6 (2) (2019) 428–435, https://doi.org/10.1002/ehf2.12419.
[32] Understanding medical tests: sensitivity, specificity, and positive predictive value, available online (Accessed 16 January 2021).
[33] B. Krawczyk, Learning from imbalanced data: open challenges and future directions, Prog. Artif. Intell. 5 (4) (2016) 221–232, https://doi.org/10.1007/s13748-016-0094-0.
[34] A. Fernández, S. Garcia, F. Herrera, N.V. Chawla, SMOTE for learning from imbalanced data: progress and challenges, marking the 15-year anniversary, J. Artif. Intell. Res. 61 (1) (January 2018) 863–905, https://doi.org/10.1613/jair.1.11192, 2018.
[35] H. He, Y. Bai, E.A. Garcia, S. Li, ADASYN: adaptive synthetic sampling approach for imbalanced learning, in: 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), Hong Kong, 2008, pp. 1322–1328, https://doi.org/10.1109/IJCNN.2008.4633969, 2008.
[36] B. Wang, W. Li, et al., Improving triaging from primary care into secondary care using heterogeneous data-driven hybrid machine learning, Decision Support Systems 166, https://doi.org/10.1016/j.dss.2022.113899, 2023.
[37] Flask's documentation, https://flask.palletsprojects.com/en/2.2.x/. (Accessed 26 August 2022).
[38] React Getting Started, https://reactjs.org/docs/getting-started.html. (Accessed 26 August 2022).

mathematics

MDPI

Article

A Concretization of an Approximation Method for Non-Affine Fractal Interpolation Functions

Alexandra Băicoianu [1], Cristina Maria Păcurar [1] and Marius Păun [2,*]

[1] Department of Mathematics and Computer Science, Faculty of Mathematics and Computer Science, Transilvania University of Braşov 50, Iuliu Maniu Str., 500090 Braşov, Romania; a.baicoianu@unitbv.ro (A.B.); cristina.pacurar@unitbv.ro (C.M.P.)

[2] Department of Forest Engineering, Forest Management Planing and Terrestrial Measurements, Faculty of Silviculture and Forest Engineering, Transilvania University of Braşov, 1 Şirul Beethoven, 500123 Braşov, Romania

* Correspondence: m.paun@unitbv.ro

Abstract: The present paper concretizes the models proposed by S. Ri and N. Secelean. S. Ri proposed the construction of the fractal interpolation function (FIF) considering finite systems consisting of Rakotch contractions, but produced no concretization of the model. N. Secelean considered countable systems of Banach contractions to produce the fractal interpolation function. Based on the abovementioned results, in this paper, we propose two different algorithms to produce the fractal interpolation functions both in the affine and non-affine cases. The theoretical context we were working in suppose a countable set of starting points and a countable system of Rakotch contractions. Due to the computational restrictions, the algorithms constructed in the applications have the weakness that they use a finite set of starting points and a finite system of Rakotch contractions. In this respect, the attractor obtained is a two-step approximation. The large number of points used in the computations and the graphical results lead us to the conclusion that the attractor obtained is a good approximation of the fractal interpolation function in both cases, affine and non-affine FIFs. In this way, we also provide a concretization of the scheme presented by C.M. Păcurar.

Keywords: fractal interpolation function; non-affine FIFs; countable affine probabilistic scheme; countable affine deterministic scheme; countable non-linear probabilistic scheme; countable deterministic non-linear scheme

MSC: Primary 28A80; 41A05; Secondary 26A18; 37C25; 37C70

Citation: Băicoianu, A.; Păcurar, C.M.; Păun, M. A Concretization of an Approximation Method for Non-Affine Fractal Interpolation Functions. *Mathematics* **2021**, *9*, 767. https://doi.org/10.3390/math9070767

Received: 10 March 2021
Accepted: 28 March 2021
Published: 1 April 2021

1. Introduction

The notion of fractal interpolation has been introduced by Barnesley in [1] (see also [2]) and it represents a different interpolation method, which results in functions that are continuous, but not necessarily differentiable at every point. The fractal interpolation function (FIF) is a continuous function with real values, which interpolates a given set of data

$$\{(x_i, y_i) \in [x_0, x_N] \times \mathbb{R}, i = \{0, \ldots, N-1\}\}$$

(i.e., $f(x_i) = y_i$ for all $i \in \{0, \ldots, N-1\}$), where x_i are sorted in an ascending order such that its graph is the attractor of an iterated function system.

The significance of FIFs is emphasized by the numerous research directions that have been broadly studied ever since they were introduced. Among these directions, we mention the hidden variable fractal interpolation, which was introduced by Barnsley et al. (see [3]) and generates functions which are not self-referential; thus, being much more less restrictive (see [4–6]), the extension to a countable iterated function systems (a notion introduced in [7–9]) to obtain the corresponding FIFs (see [10,11]) and the replacement of the fixed

point result (Banach fixed point theorem), which guarantees the existence of the FIF with different fixed point results (see [12–15]).

Among the different types of FIFs existing in the literature, there were studied affine FIFs (see [1]), but also non-affine FIFs (see [16]). However, if for the affine case, there have been studies undertaken towards the computational part (see [17–21]), as far as we know, there have not yet been any studies related to non-affine FIFs in this respect.

The aim of the present paper is to offer a concretization of an approximation method for non-affine fractal interpolation functions. Starting from the results in [8,12], in this paper, we propose two different algorithms to produce the fractal interpolation functions in both cases affine and non-affine FIFs. The theoretical context we were working in, suppose a countable set of starting points and a countable system of Rakotch contractions. Due to the computational restrictions the built algorithms, in applications, have the weakness that they uses a finite set of starting points and a finite system of Rakotch contractions. With this respect, the attractor obtained is a two-step approximation. The big amount of points used in the computations and the graphical results lead us to the conclusion that the attractor obtained is a good approximation of the fractal interpolation function in both affine and non-affine cases. In this way, we also provide a concretization of the scheme presented by C.M. Păcurar (see [15]).

In this study, we want to solve also the problem of viewing a big set of data, generated by the iterations schemes mentioned above, in order to better understand the theoretical knowledge in the function plotting field. We study the nature of data plotting in C++ regarding its pros and cons (limitations). The scope of the application is to generate graphs for various functions and to observe the steps taken by the algorithm in order to obtain the correct plotting. Using C++ (one of the fastest and most memory efficient languages) and Qt (a C++ cross-platform framework for GUI - Graphical User Interface), we developed an application that puts into use most of the modern features offered by C++ (especially, C++11 issues).

2. Mathematical Preliminaries

Let (X, d) be a metric space.

Definition 1. *The map $f : X \to X$ is called a Picard operator if f has a unique fixed point $x_* \in X$ (i.e., $f(x_*) = x_*$) and*

$$\lim_{n\to\infty} f^{[n]}(x) = x_*,$$

for every $x \in X$, where $f^{[n]}$ denotes the n-times composition of f with itself.

Definition 2. *1. A map $f : X \to X$ is called Lipschitz if there exists a real non-negative C such that*

$$d(f(x), f(y)) \leq Cd(x, y),$$

for every $x, y \in X$. The smallest C in the above definition is called Lipschitz constant and it is defined as

$$lip(f) = \sup_{x\neq y} \frac{d(f(x), f(y))}{d(x, y)}$$

2. A map $f : X \to X$ is called Banach contraction if there exists $C \in (0, 1)$ such that

$$d(f(x), f(y)) \leq Cd(x, y),$$

for every $x, y \in X$.

3. A map $f : X \to X$ is called φ-contraction if there exists a function $\varphi : [0, \infty) \to [0, \infty)$ such that

$$d(f(x), f(y)) \leq \varphi(d(x, y)),$$

for every $x, y \in X$.

4. *A map $f : X \to X$ is called Matkowski contraction if it is a φ-contraction where $\varphi : [0, \infty) \to [0, \infty)$ is non-decreasing and $\lim_{n\to\infty} \varphi^{[n]}(t) = 0$ for all $t > 0$.*
5. *A map $f : X \to X$ is called Rakotch contraction if it is a φ-contraction where $\varphi : [0, \infty) \to [0, \infty)$ is such that the function $t \to \frac{\varphi(t)}{t}$ is non-increasing for every $t > 0$ and $\frac{\varphi(t)}{t} < 1$ for every $t \in (0, \infty)$.*

Remark 1. 1. *Every Banach contraction is Lipschitz where the Lipschitz constant is smaller than* 1.
2. *Every Banach contraction is a φ-contraction, for*

$$\varphi(t) = C \cdot t,$$

for every $t > 0$.
3. *Every Rakotch contraction is a Matkowski contraction.*

In [22], the following fixed point result was proved.

Theorem 1. *Every Matkowski contraction on a complete metric space is a Picard operator.*

2.1. Iterated Function Systems

Hutchinson introduced the notion of iterated function systems in [23]. Secelean extended the notion to countable iterated function systems, composed of a countable number of constitutive functions (see [8]).

Definition 3. *Let (X, d) be a compact metric space and the continuous functions $f_n : X \to X$. The system of all functions f_n is called a countable iterated function system (CIFS), which will be denoted by $\mathcal{S} = \{(f_n)_{n\geq 0}\}$*

Let $\mathcal{P}_{\text{cp}}(X)$ be the class of all non-empty compact subsets of X.
The fractal operator associated to $\mathcal{S}$ is the map $F_{\mathcal{S}} : \mathcal{P}_{\text{cp}}(X) \to \mathcal{P}_{\text{cp}}(X)$ defined as

$$F_{\mathcal{S}}(K) = \overline{\bigcup_{n\geq 1} f_n(K)}$$

for every $K \in \mathcal{P}_{cp}(X)$.

If the functions f_n are Matkowski contractions (or Rakotch contractions, or φ-contractions, or Banach contractions), the fractal operator associated to the CIFS $\mathcal{S}$ is a Picard operator and its unique fixed point is called the attractor of $\mathcal{S}$, which will be denoted by $A_{\mathcal{S}}$.

2.2. Countable FIFs

Let (Y, d) be a compact metric space and the countable system of data

$$\{(x_n, y_n) \in [x_0, m] \times Y, n \geq 0\}, \tag{1}$$

where the sequence $(x_n)_{n\geq 0}$ is strictly increasing and bounded and $m = \lim_{n\to\infty} x_n$, and the sequence $(y_n)_{n\geq 0}$ is convergent. We make the notation $M = \lim_{n\to\infty} y_n$.

Definition 4. *An interpolation function for the system of data (1) is a continuous function $f : [x_0, m] \to Y$ such that $f(m) = M$ and $f(x_n) = y_n$ for all $n \geq 0$.*

Let us recall from [15], the way we can construct a family of functions associated to the system of data (1):

Let $l_n : [x_0, m] \to [x_n, x_{n+1}]$ be a family of contractive homeomorphisms such that

(i) there exists $C_n \in [0,1)$ such that

$$|l_n(x) - l_n(x')| \leq C_n|x - x'|$$

for every $x,\ x' \in [x_0, m]$;

(ii)

$$l_n(x_0) = x_{n-1} \quad \text{and} \quad l_n(m) = x_n;$$

(iii)

$$\sup_{n\geq 1} C_n < 1.$$

By $diam(A)$ we denote the diameter of A.

Let $F_n : [x_0, m] \times Y \to Y$ be continuous functions such that

(j)

$$F_n(x_0, y_0) = y_{n-1} \quad \text{and} \quad F_n(m, M) = y_n;$$

(jj) $\lim_{n\to\infty} diam(Im\, F_n) = 0$.

We can now define the family of functions $(f_n)_{n\geq 0}$

$$f_n : [x_0, m] \times Y \to [x_0, m] \times Y$$

associated to the system of data (1) as

$$f_n(x, y) = (l_n(x), F_n(x, y)),$$

for every $x \in [x_0, m]$ and $y \in Y$.

Let

$$\mathcal{F}([x_0, m]) = \{f : [x_0, m] \to Y |\, f(x_0) = y_0,\ f(m) = M,\ f \text{ - continuous}\}$$

endowed with the uniform metric $d_{\mathcal{F}([x_0,m])}$.

Remark 2. *(see Theorems 3.2 and 3.3 from [15])*

1. *If the functions F_n are Lipschitz with respect to the first variable and Rakotch contractions with respect to the second variable, then the functions f_n are Rakotch contractions with respect to d_θ, where*
$$d_\theta((x, y), (x', y')) := |x - x'| + \theta d(y, y')$$
for all $(x, y),\ (x', y') \in [x_0, m] \times Y$, where $\theta = \frac{1-\sup_{n\geq 1} C_n}{2(C+1)} \in (0, 1)$.
2. *Given the same aforementioned framework, there exists an interpolation function f_* corresponding to the system of data (1) such that its graph is the attractor of the CIFS $\mathcal{S} = (([x_0, m] \times Y, d_\theta), (f_n)_{n\geq 1})$.*

In the particular case that Y is a compact real interval, $Y \subset (0, \infty)$, we can choose the non-affine functions f_n as follows (see [15]):

$$\begin{aligned} f_n(x, y) = \Bigg(& \frac{x_n - x_{n-1}}{m - x_0} x + \frac{m x_{n-1} - x_0 x_n}{m - x_0}, \\ & \left(\frac{y_n - y_{n-1}}{m - x_0} - \frac{1}{m - x_0} \left(\frac{M}{1 + nM} - \frac{m}{1 + nm} \right) \right) x + \frac{y}{1 + ny} \\ & + y_{n-1} - x_0 \frac{y_n - y_{n-1}}{m - x_0} + \frac{x_0}{m - x_0} \frac{M}{1 + nM} - \frac{m}{m - x_0} \frac{m}{1 + nm} \Bigg). \end{aligned}$$

For the affine case, when Y is a compact real interval, one can choose the functions f_n as follows (see [10]):

$$f_n(x,y) = \left(\frac{x_n - x_{n-1}}{b-a}x + \frac{bx_{n-1} - ax_n}{b-a}, \right.$$
$$\left. \left(\frac{y_n - y_{n-1}}{b-a} - d_n \frac{M-m}{b-a} \right) x + d_n y + \frac{by_{n-1} - ay_n}{b-a} - d_n \frac{bm - aM}{b-a} \right).$$

3. Computational Background

3.1. Applied Technologies. Motivation (Pros)

Qt is a widget toolkit for creating graphical user interfaces as well as cross-platform applications that run on various software and hardware platforms such as Linux, Windows, macOS, Android or embedded systems with little or no change in the underlying codebase, while still being a native application with native capabilities and speed. Qt Creator is a cross-platform C++, JavaScript and QML integrated development environment which simplifies Graphical User Interface (GUI) application development. It includes a visual debugger and an integrated WYSIWYG (What You See Is What You Get) GUI layout and forms the designer. The editor has features such as syntax highlighting and autocompletion.

One of the main problems encountered during development was using user-input flexible functions, that is why we prioritised adding a fairly robust mathematics parsing engine written in C++. We chose CmathParser (https://github.com/NTDLS/CMathParser, accessed on 15 February 2021) that provides a robust collection of functions and structures that give users the ability to parse and evaluate various types of expressions. Although it is fairly lightweight, CMathParser can interpret a various list of mathematical functions and operations and its performance is convenient in relation to the advantages that this engine brings. The mathematical functions used need to follow a specific syntax (for example, $\sqrt{x}$ is SQRT(x)), this is why we found it useful to read our functions from a file. We opted for reading from an XML (Extended Markup Language) file because we can use the tags in our advantage and clearly define every function and every parameter for that function.

Below, Listing 1, is an example for an XML file accepted by the application:

Listing 1: Example for an XML file.

```
<?xml version="1.0"?>

<function>
<Xn>SQRT(n)/(1+SQRT(n))</Xn>
<Yn>SIN(n)/SQRT(1+n)</Yn>
<FnX>(((XN)-(XN-1))*X+b*(XN-1)-a*(XN))/(b-a)</FnX>
<FnY>((YN)-(YN-1)-(DN)*((M)-m))*X/(b-a)+(DN)*Y+(b*(YN-1)-a*(YN)-(DN)*(b*m-a*(
    M)))/(b-a)</FnY>
<a>0</a>
<b>1</b>
<m>0</m>
<M>0</M>
<dn>0.4</dn>
</function>
```

QcustomPlot (https://www.qcustomplot.com/, accessed on 15 February 2021) is a Qt C++ widget for plotting and data visualization. It has no further dependencies and is well documented. This plotting library focuses on making good looking, publication-quality 2D plots, graphs and charts, as well as offering high performance for real time data visualization applications.

3.2. Technical Notes on Performance

Our target regarding the application's performance focused around optimising the algorithm in a way that it brings up the graphs as soon as possible. To increase the performance, we used multithreading: we use all the available threads on the CPU and developed the algorithm in a way that favors concurrency. The algorithm finds out how

many threads are available and uses them (for a CPU with 4 threads, the algorithm will use a maximum of 4 threads on max load) although it is possible to start any number of threads (the OS scheduler will put them in a priority queue). A program that starts 100 threads for 100 tasks on a 4 threaded CPU will be less performant than the same program that splits those 100 tasks in a way that the CPU takes 4 tasks at a time.

This, Listing 2, is a threading code snippet that spawns as many threads as available to generate points:

Listing 2: Multithreading.

```
int numberThread;

if (std::thread::hardware_concurrency() > 0)
numberThread = std::thread::hardware_concurrency();
else
numberThread = 4;

auto spawnThreads = [&]()
{
std::vector<std::thread> threads;
for (int index = 1; index < numberThread; index++)
{
threads.push_back(std::thread(generate, 0, numberPoint / numberThread,
"..//.//QtExample\\Step1\\file" + std::to_string(index) + ".txt"));
}
threads.push_back(std::thread(generate, 0, numberPoint - numberPoint /
    ↪ numberThread *
(numberThread - 1), "..//.//QtExample\\Step1\\file
    " + std::to_string(numberThread) + ".txt"));

for (auto& th : threads)
{
th.join();
}
};
```

The number of used threads impacts performance, and overall, the quality and time; see Figure 1. It can also be seen that as the number of points increases, the difference between the time associated with running with 8 threads versus 16 threads increases considerably.

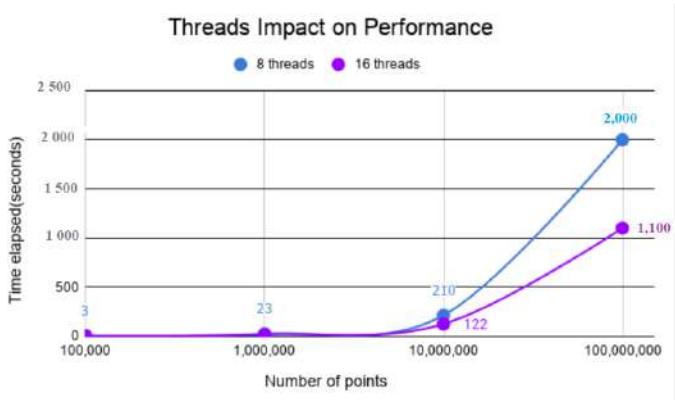


Figure 1. Threads impact on performance.

3.3. Limitations (Constraints)

At the present time, the only known limit of the application is the time required to generate and plot the points when they are billions or even more in number. The next examples were run on a i7-7700HQ with 8 threads. RAM memory is not really relevant because the algorithm is very CPU heavy.

For the probabilistic scheme, generating and plotting 100,000 points are made in approximately 2–3 s, and for every $p * 100{,}000$ (p is a signed integer), the time will be approximately $p * x$, where $x \in [2, 3]$.

For the deterministic scheme, things become challenging. At a low level, the time for generating and plotting points will be the same as for Step 1. The difference occurs in the number of points the scheme generates for parameters k, n, p. leading to run time fluctuations and may occur due to the insertion and processing of points in the files.

4. Main Results

4.1. Countable Fractal Non-Affine Interpolation Schemes

We start the study producing an approximation for the fractal countable non-affine interpolation scheme in two different ways. Let us consider the positive, increasing, convergent sequence $(x_n)_{n\in\mathbb{N}}$, the convergent sequence $(y_n)_{n\in\mathbb{N}}$ defined by $(x_n)_{n\in\mathbb{N}}$, $(y_n)_{n\in\mathbb{N}}$ given by the following:

$$x_n = \frac{3\sqrt{n}+1}{\sqrt{n}+1}, \quad y_n = \frac{\left|\sin\left(\frac{180\cdot n}{\pi}\right)\right|+1}{\sqrt{n}+1}$$

and the sequence of non-affine functions given by the following:

$$\begin{aligned} f_n(x,y) = \Bigg(& \frac{x_n - x_{n-1}}{m-x_0}x + \frac{mx_{n-1} - x_0x_n}{m-x_0}, \\ & \left(\frac{y_n - y_{n-1}}{m-x_0} - \frac{1}{m-x_0}\left(\frac{M}{1+nM} - \frac{m}{1+nm}\right)\right)x + \frac{y}{1+ny} \\ & + y_{n-1} - x_0\frac{y_n - y_{n-1}}{m-x_0} + \frac{x_0}{m-x_0}\frac{M}{1+nM} - \frac{m}{m-x_0}\frac{m}{1+nm}\Bigg). \end{aligned}$$

for all $(x, y) \in [x_0, m] \times Y$. The argument of the function sinus we used when defining y_n was imposed by the fact that the Mathematical Function Library demands us to work in radians.

The absolute value of sinus in the definition of y_n was imposeed by the fact that every second coordinate of the points obtained in the process must be positive in order to have a Rakotch contraction. The result obtained is an approximation of the countable non-affine interpolation schemes. For the desired approximation, we take the following subsets

$$XP = \{x_n \mid n \leq 100\}, YP = \{y_n \mid n \leq 100\}, FP = \{f_n \mid n \leq 100\}$$

The two schemes we used are the probabilistic interpolation scheme and the deterministic interpolation scheme. The probabilistic scheme is described in Algorithm 1.

The deterministic scheme we are triyng to apply is given by the Algorithm 2.

The probabilistic scheme (Algorithm 1), after 100,000 steps, leads us to the result shown in Figure 2.

Algorithm 1: The Probabilistic scheme.

1. Consider an empty set of points $\mathcal{P} \in \mathbb{R}^2$ and p a signifiant big signed positive integer.
2. Generate an arbitrary point $(xa, ya) \in [0,1] \times [0,1]$.
3. Determine $\mathcal{P} \bigcup \{(xa, ya)\}$.
4. Generate a random signed integer $0 < k \leq 100$.
5. Compute $(xa, ya) = f_k(xa, ya)$.
6. Repeat steps 3, 4, 5 p times
7. Sort the elements of the set $\mathcal{P}$ in ascending order regarding the first component of the elements.
8. Plot the function passing through all the points of the set $\mathcal{P}$.

Algorithm 2: The Deterministic Scheme.

1. Consider k the number of the initial points, n the number of functions involved, p the number of steps and define an empty set of points $\mathcal{P} \in \mathbb{R}^2$.
2. Generate randomly a set K_0 of k points in $[0,1] \times [0,1]$.
3. Determine $\mathcal{P} \bigcup K_0$.
4. Compute $\mathcal{P} = f_1(\mathcal{P}) \bigcup f_2(\mathcal{P}) \bigcup .. \bigcup f_n(\mathcal{P})$
5. Repeat step 4 for p times
6. Sort the elements of the set $\mathcal{P}$ in ascending order regarding the first component of the elements.
7. Plot the function passing through all the points of the set $\mathcal{P}$.

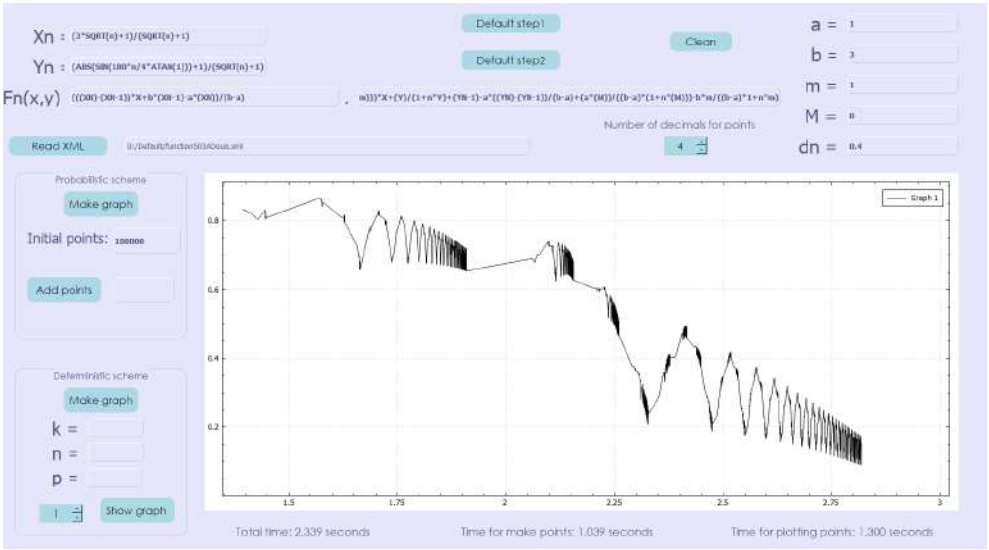


Figure 2. Non-affine probabilistic interpolation scheme (approximation with 100,000 points).

The time spent to generate the points of the scheme was 1.039 s and the plotting time 1.3 s.

Using Algorithm 2, in the initial conditions of the probabilistic scheme, for the deterministic scheme taking $k = 100$, $n = 100$ and $p = 3$, we obtain the graph given in Figure 3. The scheme generated 100,000,000 points and the total duration of computing and plotting was 1474.477 s.

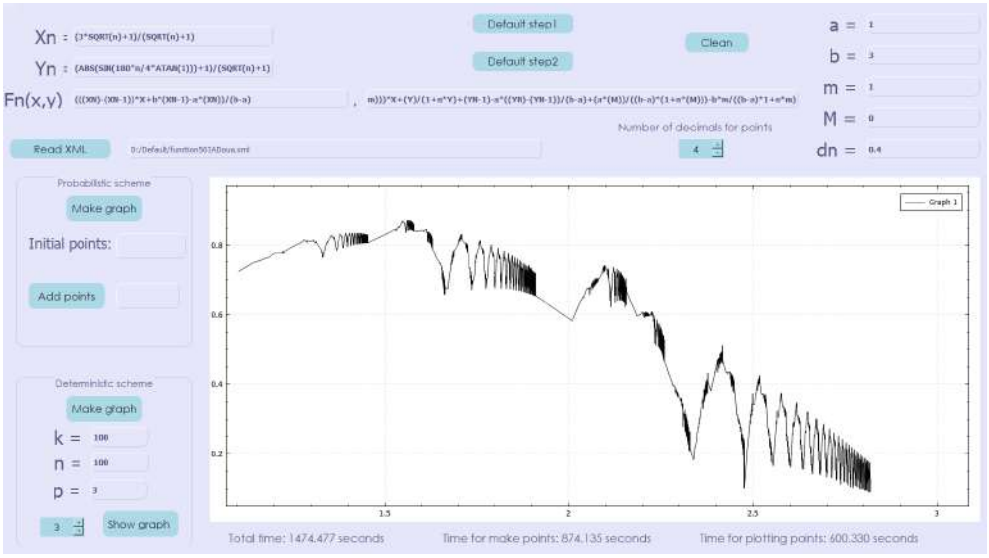


Figure 3. Non-affine deterministic interpolation scheme (approximation with 100,000,000 points).

In Figure 4, a graph with the plotting of the interpolation function in every step of the algorithm is shown.

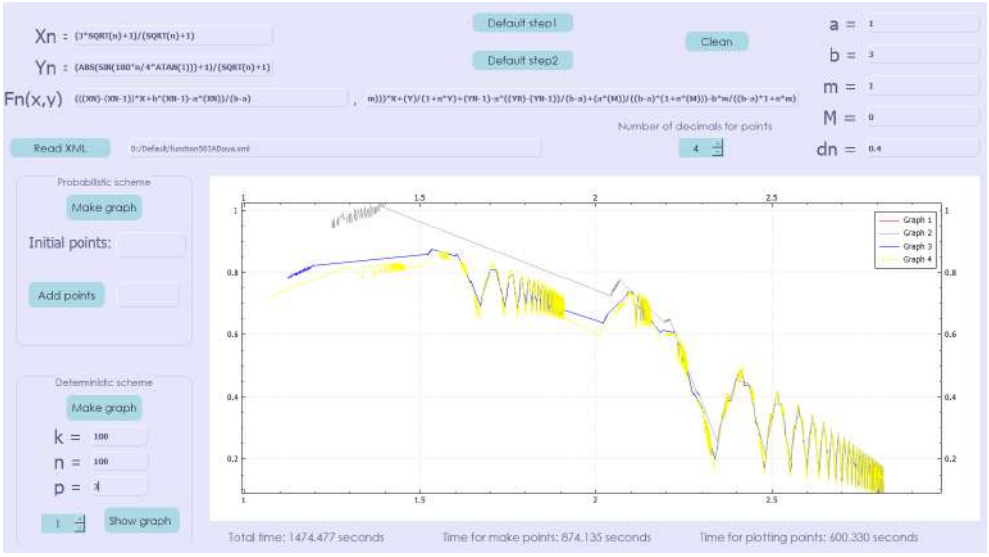


Figure 4. Non-affine deterministic interpolation scheme (approximation with 100,000,000 points), plotting the graph of every step in the algorithm.

4.2. Countable Fractal Affine Interpolation Schemes

We made the study producing an approximation for the fractal countable affine interpolation scheme in the same two ways as for the non-affine case. Let us consider the positive, increasing, convergent sequence $(x_n)_{n\in\mathbb{N}}$, the convergent sequence $(y_n)_{n\in\mathbb{N}}$ defined by

$$x_n = \frac{3\sqrt{n}+1}{\sqrt{n}+1}, \quad y_n = \frac{\cos\left(\frac{180\cdot n}{\pi}\right)+1}{\sqrt{n}+1}$$

and the affine sequence of functions $(f_n(x,y))_{n\in\mathbb{N}}$

$$f_n(x,y) = \left(\frac{x_n - x_{n-1}}{b-a}x + \frac{bx_{n-1} - ax_n}{b-a},\right.$$
$$\left.\left(\frac{y_n - y_{n-1}}{b-a} - d_n\frac{M-m}{b-a}\right)x + d_n y + \frac{by_{n-1} - ay_n}{b-a} - d_n\frac{bm - aM}{b-a}\right).$$

for all $(x,y) \in [x_0, m] \times Y$. The argument of the function cosinus we used when defining y_n was imposed by the fact that the Mathematical Function Library demands us to work in radians.

For the desired approximation, we take the following subsets

$$XP = \{x_n \mid n \leq 100\}, YP = \{y_n \mid n \leq 100\}, FP = \{f_n \mid n \leq 100\}$$

The probabilistic scheme (Algorithm 1), after 100,000 steps, leads us to the result shown in Figure 5.

The deterministic scheme produces, for $k = 100$, $n = 100$ and $p = 3$, the graph in Figure 6.

The time for obtaining the points in these conditions was 595.631 s and the time for plotting the function was 1594.470 s. The graph with the plotting of the interpolation function in every step of the algorithm is given in Figure 7.

The red graph is for the function given by the random generated 100 points. Graph 2 is the function for 10,000 thousand points (step for p = 1), graph 3 is for the function after step p = 2 (1,000,000 points) and the yellow graph is the final one—the same as in Figure 7.

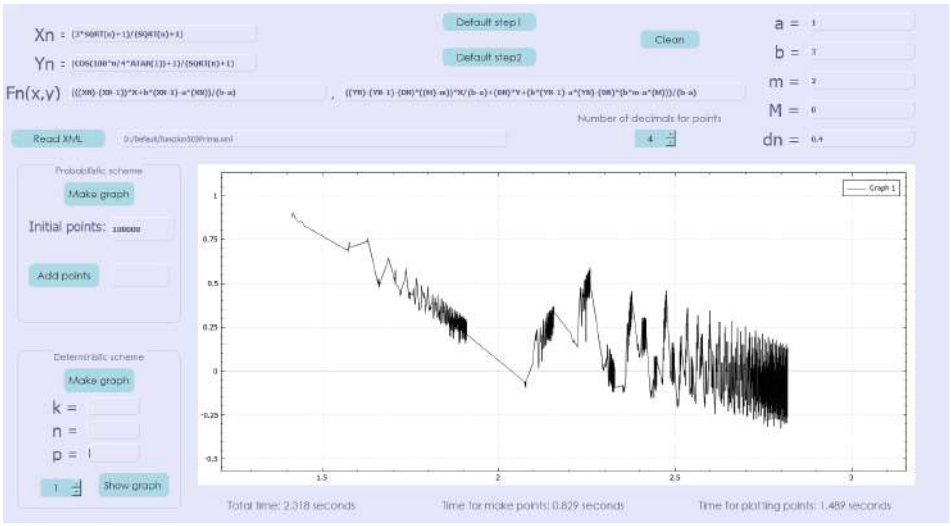


Figure 5. Affine probabilistic interpolation scheme (approximation with 100,000 points).

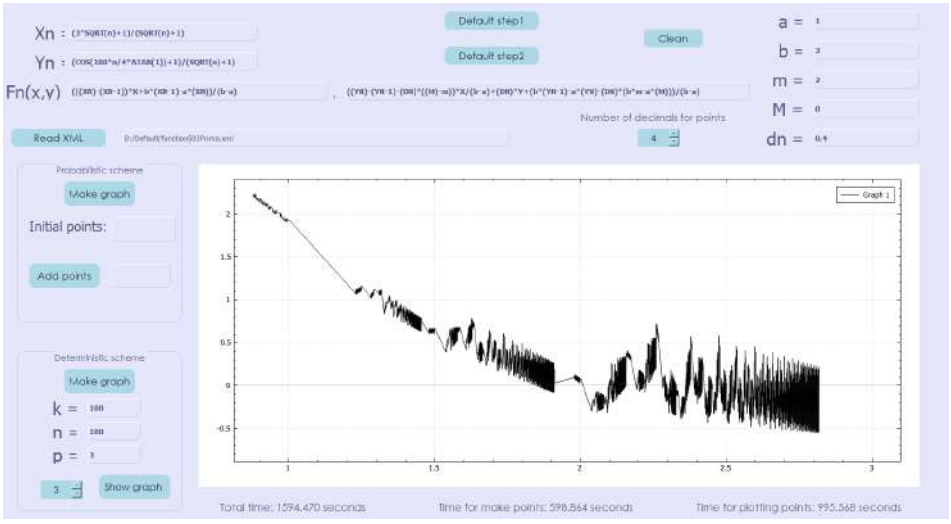


Figure 6. Affine deterministic interpolation scheme (approximation with 100,000,000 points).

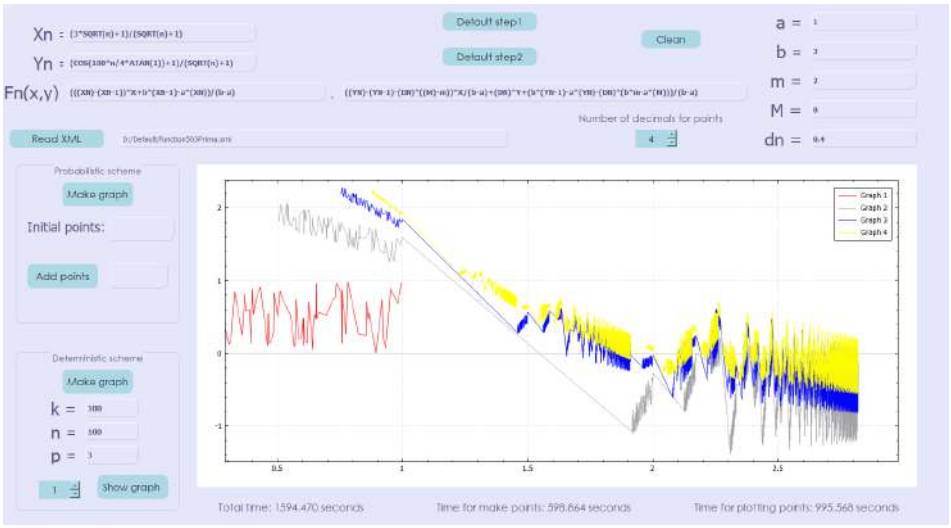


Figure 7. Affine deterministic interpolation scheme (approximation with 100,000,000 points, step by step).

5. Conclusions

The main conclusion of this study is that the algorithms presented give similar approximations of the FIFs for both schemes, affine and non-affine. For the probabilistic scheme (Algorithm 1), significant results are obtained for more than 10,000 steps and the time elapsed to plot the graph is less than two seconds. The deterministic scheme (Algorithm 2) permits the study of the variation of FIFs, step by step, but, in order to obtain significant results one must perform more than three steps. The time elapsed in this case is more than 1000 s. In the applications presented, both algorithms have an imposed number of steps.

Further studies are to be made in order to obtain a condition for stopping the algorithms if some conditions are fulfilled .

Besides the two algorithms, the computer application is a useful tool for plotting big sets of data generated by the iteration schemes described above through functions made in C++. It truly shows the capabilities of this programming language and it pushes it to the maximum using threading and modern programming techniques.

Author Contributions: Conceptualization: M.P.; introduction and preliminaries: C.M.P. and M.P.; methodology: M.P. and A.B.; visualization: A.B.; original draft preparation: M.P.; C.M.P. and A.B.; review, editing, validation, and formal analysis: M.P., C.M.P. and A.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors want to express their gratitude for the reviewers. Their observations were very useful to improve the scientific value of the paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Barnsley, M.F. Fractal functions and interpolation. *Constr. Approx.* **1986**, *2*, 303–329. [CrossRef]
2. Barnsley, M. *Fractals Everywhere*; Academic Press: New York, NY, USA, 1988.
3. Barnsley, M.F.; Elton, J.; Hardin, D.; Massopust, P. Hidden variable fractal interpolation functions. *SIAM J. Math. Anal.* **1989**, *20*, 1218–1242. [CrossRef]
4. Mazel, D.S.; Hayes, M.H. Hidden-variable fractal interpolation of discrete sequences. In *ICASSP 91: 1991 International Conference on Acoustics, Speech, and Signal Processing*; IEEE Computer Society: Toronto, ON, Canada, 1991.
5. Chand, A.K.B.; Kapoor, G.P. Hidden variable bivariate fractal interpolation surfaces. *Fractals* **2003**, *11*, 277–288. [CrossRef]
6. Bouboulis, P.; Dalla, L. Hidden variable vector valued fractal interpolation functions. *Fractals* **2005**, *13*, 227–232. [CrossRef]
7. Fernau, H. Infinite iterated function systems. *Math. Nachrichten* **1994**, *170*, 79–91. [CrossRef]
8. Secelean, N. Countable Iterated Fuction Systems. *Far East J. Dym. Syst.* **2001**, *3*, 149–167.
9. Secelean, N. *Countable Iterated Function Systems*; LAP Lambert Academic Publishing: Saarbrueken, Germany, 2013.
10. Secelean, N. The fractal interpolation for countable systems of data. *Univ. Beograd. Publ. Elektrotehn. Fak. Ser. Mat.* **2003**, *14*, 11–19. [CrossRef]
11. Secelean, N. Fractal countable interpolation scheme: Existence and affine invariance. *Math. Rep. (Bucur.)* **2011**, *13*, 75–87.
12. Ri, S. A new idea to construct the fractal interpolation function. *Indag. Math.* **2018**, *29*, 962–971. [CrossRef]
13. Kim, J.; Kim, H.; Mun, H. Nonlinear fractal interpolation curves with function vertical scaling factors. *Indian J. Pure Appl. Math.* **2020**, *51*, 483–499. [CrossRef]
14. Ri, S.; Drakopoulos, V. How Are Fractal Interpolation Functions Related to Several Contractions? In *Mathematical Theorems—Boundary Value Problems and Approximations*; Alexeyeva, L., Ed.; IntechOpen: London, UK, 2020.
15. Pacurar, C.M. A countable fractal interpolation scheme involving Rakotch contractions. *arXiv* **2021**, arXiv:2102.09855.
16. Dalla, L.; Drakopoulos, V.; Prodromou, M. On the box dimension for a class of non-affine fractal interpolation functions. *Anal. Theory Appl.* **2003**, *19*, 220–233. [CrossRef]
17. de Amo, E.; Chiţescu, I.; Diaz Carrillo, M.; Secelean, N.A. A new approximation procedure for fractals. *J. Comput. Appl.* **2003**, *151*, 355–370. [CrossRef]
18. Dubuc, S.; Elqortobi, A. Approximations of fractal sets. *J. Comput. Appl. Math.* **1990**, *29*, 79–89. [CrossRef]
19. Chiţescu, I.; Miculescu, R. Approximation of fractals generated by Fredholm integral equations. *J. Comput. Anal. Appl.* **2009**, *11*, 286–293.
20. Chiţescu, I.; Georgescu, H.; Miculescu, R. Approximation of infinite dimensional fractals generated by integral equations. *J. Comput. Appl. Math.* **2010**, *234*, 1417–1425. [CrossRef]
21. Miculescu, R.; Mihail, A.; Urziceanu, S.-A. A new algorithm that generates the image of the attractor of a generalized iterated function system. *Numer. Algorithms* **2020**, *83*, 1399–1413. [CrossRef]
22. Matkowski, J. Integrable solutions of functional equations. *Dissertationes Math.* **1975**, *127*, 68.
23. Hutchinson, J. Fractals and self similarity. *Indiana Univ. Math. J.* **1981**, *30*, 713–747. [CrossRef]

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Learning about Growing Neural Cellular Automata

SORANA CATRINA[2], MIRELA CATRINA[2], ALEXANDRA BĂICOIANU[1], IOANA CRISTINA PLAJER[1]

[1]Department of Mathematics and Computer Science, Transilvania University of Brasov, Romania (e-mail: a.baicoianu@unitbv.ro)
[2]Faculty of Mathematics and Computer Science, Transilvania University of Brasov)

Corresponding author: Alexandra Băicoianu (e-mail: a.baicoianu@unitbv.ro).

ABSTRACT Neural cellular automata have been proven effective in simulating morphogenetic processes. Developing such automata has been applied in 2D and 3D processes related to creating and regenerating complex structures and enabling their behaviors. However, neural cellular automata are inherently uncontrollable after the training process. Starting from a neural cellular automaton trained to generate a given shape from one living cell, this paper aims to gain insight into the behavior of the automaton, and to analyze the influence of the different image characteristics on the training and stabilization process and its shortcomings in different scenarios. For each considered shape, the automaton is trained on one RGB image of size 72×72 pixels containing the shape on an uniform white background, in which each pixel represents a cell. The evolution of the automaton starts from one living cell, employing a shallow neural network for the update rule, followed by backpropagation after a variable number of evolutionary steps. We studied the behavior of the automaton and the way in which different components like symmetry, orientation and colours of the shape influence its growth and alteration after a number of epochs and discussed this thoroughly in the experimental section of the paper. We further discuss a pooling strategy, used to stabilize the model and illustrate the influence of this pooling on the training process. The benefits of this strategy are compared to the original model and the behavior of the automaton during its evolution is studied in detail. Finally, we compare the results of models using different filters in the first stage of feature selection.
The main results of our study are the insights gained into how the neural cellular automaton works, what it is actually learning, and what influence this learning, as there are observable result differences depending on the characteristics of the input images and the filters used in the model.

INDEX TERMS Neural cellular automaton, cell state, pooling strategy, stabilizing strategy, image characteristics

I. INTRODUCTION AND BACKGROUND

WITHIN the domain of computational models, the intersection of regenerating models and cellular automata presents a compelling perspective on adaptive systems and emergent behaviors. Regenerating models within machine learning are designed to continuously evolve and enhance their performance over time, dynamically adjusting to fluctuations in data distributions. Simultaneously, cellular automata, a category of discrete dynamical systems, manifest self-regeneration through iterative update rules governing the states of individual cells. This convergence of concepts opens avenues for investigating the adaptability of computational models inspired by the inherent regenerative characteristics observed in cellular automata. A deeper exploration into the parallel principles of evolution in both domains promises valuable insights into the potential synergy between regenerating models and the self-organizing dynamics exhibited by cellular automata.

Cells are the basic building blocks of all living entities. Any multi-cellular organism evolves from one singular cell that knows how to divide, when to divide, and holds all useful information necessary for the organisms' growth and, sometimes, regeneration. Cells group and regroup, decide what tissue or organ to form and when to stop the growing process. All these complex behaviors are being built on cells that only know their own information and that of their neighbours.

Cellular automata have been developed to simulate cell behavior and to emulate some of the properties of real-world

organisms, like local behavior, parallelism, or self-replication [1]. One of their strengths is the capacity of modeling complex systems by simple local update rules, especially when combined with learning automata [2]. A cellular automaton is a dynamic system consisting of a grid of cells. Each cell holds useful information and a state that updates each time unit. This update is based on a predefined rule: given the current state, s_t and the neighboring cells states, this rule will output s_{t+1} [3], [4]. There are multiple types of cellular automata, varying in complexity and purpose. However, for this paper, we will separate them into two categories: regular cellular automata and neural cellular automata (NCA), based on how the update rule is defined manually or neural network-based [2], [5].

An elementary cellular automaton's cells have two possible states referred to as dead/alive, 0/1, white/black, etc. The first type of cellular automaton that has been studied was the one-dimensional elementary cellular automaton (ECA) [6]. The evolution in t_n discrete time units of such an automaton can be displayed as a grid, where the top row represents the cells' states at the start time (t_0) and each i-th row represents the states of the cells at the given time t_i. For these ECA a cell's state update is based on three values: its current state, the state of the cell on the left and the state of the cell on the right. This leads to a total of 2^8 possible rules, defined as $f : \{0, 1\}^3 \rightarrow \{0, 1\}$,

$$state(c_{i,t+1}) = f(state(c_{i-1,t}), state(c_{i,t}), state(c_{i+1,t}))).$$

Despite their seemingly straightforward rules, multiple studies have been conducted in order to analyze and categorize their behaviors and use-cases [7], [8].

As time progressed, an interest in cellular automata and their applications grew. This led to the development of bidimensional cellular automata - automata displayed as a grid where each cell's update rule is based on the state of its neighboring cells, the difference consisting of the type of neighbourhood chosen. The most common neighborhoods to be considered are Moore Neighbourhood (8 neighbors), and Von Neumann Neighbourhood (4 neighbors) [1], even though other types still exist [9], [10]. These rules allowed for more complexity, yet not enough, given the hand-written rules. The introduction of neural networks (NNs) in cellular automata allowed for complex update rules that can be learned. This type of CA has started to emerge and be introduced in different areas such as texture generation and regeneration [11], [12], 2D or 3D models growth and regeneration [13], [14], [15], [16], areas that were impossible to tackle with hand-written rules.

This paper analyzes the behavior of a cellular automaton built for growing and maintaining a 2D model. The NCA is trained on a given image and expected to form that image starting from one living cell, representing a *seed*. Considering the NCA architecture presented in [14], our study aims to analyze the evolution and learning process of this NCA on different images. We also compare the results obtained from these different targets and extract traits that affected them. Moreover, we also analyze the results on a stabilized model, i.e., one that is able to hold its shape after growing, a task that is not achieved in the initial model.

The experiments in this study were motivated by the interest in discovering, how and what such an automaton learns in the training phase and what image characteristics influence its evolution. These findings are explained and detailed in the experimental part of the paper. Throughout our experimental investigations, diverse behaviors have emerged, motivating our ongoing efforts to pursue specific objectives. One key objective is understanding the factors underlying an image's susceptibility to self-destruction following NCA processing. Furthermore, our focus extends to delineating the primary features that shape an image's evolution during the NCA expansion.

II. MODEL ARCHITECTURE AND TRAINING DETAILS

Motivated by the methodology from [14], we chose to test an NCA designed for image generation. The update rule, which determines how each cell changes depending on its neighbors, will now be learned by a NN. Before delving into the NN, we will thoroughly go over the steps involved in NCA training, with an emphasis on cellular automata.

A. CELL STATE

As stated before, a cellular automaton is a collection of cells arranged in a grid with a particular form, each of which has a predefined set of rules that it follows and changes state based on its neighbors' states.

Generally speaking, a cell's state is seen as binary, with 0 denoting death and 1 denoting life. However, since a NN performs best with continuous values, the interval [0, 1] was used in place of the binary-defined state of the cellular automaton, and a threshold $l_s = 0.1$, stating that if $alpha \geq l_s$ the cell is *alive*, and if $alpha < l_s$ the cell is *dead* [14].

The features of a cell have to be well-defined and compatible with our NN in order to create a final updating rule. We therefore used the following approach, in which the state of each cell is represented as a vector of 16 values (α and 15 extra attributes), determining thus an input image gird with 16 channels. Each feature vector has:

- on positions [1-3]: the RGB values of the pixel, each value being between 0 and 1;
- on position 4: the alpha channel; this is the value that determines if the cell is considered alive, dead, or in the growing phase (meaning $\alpha = 0 =$ the cell is *dead*, $\alpha = 1 =$ the cell is fully developed, all other values greater than 0.1 (l_s) are considered growing state);
- on positions [5, 16]: supplementary channels required by the NN in order to develop complex local rules.

Examples of the values encoded in such a cell of the snowflake emoticon are represented in Figure 1. The first 3 values correspond to the RGB channels, the 4th element is the alpha value (indicating if the cell is alive) and the remaining values offer additional information, learned during

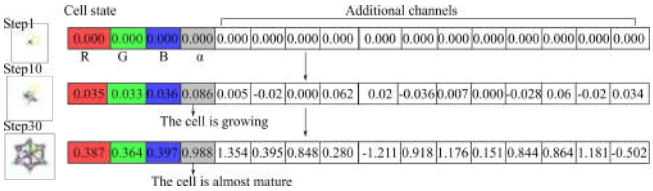


FIGURE 1. Encoding strategy.

the iterations. The top figure illustrates the feature vector of the cell highlighted by a yellow border at the beginning of the iterations. All the values of the array are at that point equal to 0. The middle figure shows the values for the given cell after 10 iterations, and the bottom figure shows the values for the given cell after 30 iterations. In the beginning, all the pixels except the seed pixel have all the values of the feature array equal to 0. The seed pixel has on the first 3 channels the RGB values normalized in the interval [0,1], and the alpha channel is equal to 1, as it is the only alive pixel. The other channels are also set initially to 0. Although dead pixels initially have RGB values equal to 0, which encodes black, these cells are displayed as white background pixels.

B. UPDATE RULE

The update rule for each living cell takes as input the current cells value and the values of its neighbours (states) and outputs a number representing the *update value*, i.e. *how much the cell's values (features and state) modify*. This value is a number to add/subtract from the current value, thus updating the current cell. It is worth noting that the outputs of the update rule are applied simultaneously on all the cells, as the entire image is passed through the NN at once. After the pass, the resulting living cells and their neighbours are selected, and processed according to the update rules, and so forth. Thus, the image evolves by iterating this update process. One such update iteration implies passing the feature array of the cell grid through a NN.

The first stage of the network implies convolving classical filter masks of size 3×3 with each channel of the input. These filters operate with the Moore Neighborhood of each cell.

The filter masks considered are:

- $Sobel_x$: a high-pass filter that considers 6 of the neighboring cells situated on the left and right.
- $Sobel_y$: a high-pass filter considering 6 top-bottom neighbours of the current cell.
- $Identity$: ensures that the current cell value is taken into consideration and influences the update rule

The role of a NN is to learn important input features. An image's most relevant features are edges, especially when the image represents one or more objects on a uniform background. The Sobel filter is a classical filter for edge finding, which also considers the edge's orientation. The $Sobel_x$ filter predominantly finds vertical edges, while the $Sobel_y$ the horizontal ones. Complementary to these filters, which are the ones used in [14], in this study, we also explored filters with other orientations, like the diagonal Sobel and the isotropic Laplace filter. The results of these experiments are detailed in the experiment section. A description of all these filters is provided by [17].

Each filter is applied to each channel of the input image and generates a $72\times72\times16$ array. The three arrays obtained thus are concatenated into the network's input feature array of size $72\times72\times48$. Although we could have experimented with extended variations of the neighborhoods, considering the subsequent filters, this would have gone against the cellular automaton's basic tenet that a cell only knows its neighbors.

C. THE NEURAL NETWORK

The neural network, as shown in Figure 2, comprises a perception layer followed by two convolutional layers. The perception layer receives an image of shape 72x72x16, representing the current state of the NCA, and applies hardcoded 3x3 filters (in this case $Identity$, $Sobel_x$ and $Sobel_y$), outputting a tensor of shape 72x72x48. The first convolutional layer filters this tensor with 128 kernels of size 1×1, utilizing ReLu activation. Following this, the second one employs 16 1×1 kernels. The network output is therefore a $72\times72\times16$-dimensional array representing the updated values, which are added to the original values of the considered cells. To offer a comprehensive understanding of the NN's architecture utilized in our study, Table 1 presents explicit details regarding the layer-wise dimensions and filter sizes and Figure 3 graphically illustrates the described network.

TABLE 1. Explicit details about the Neural Network.

Type	Kernel	Output Shape	Params
Depthwise	List of given g $3x3$ kernels (eg. $Identity$, $Sobel_x$, $Sobel_y$, Laplacian, etc)	(72,72, g*16)	0
Conv2d	(1x1)	(72, 72, 128)	(g*16+1)*128
Conv2d	(1x1)	(72, 72, 16)	2064

A single cell perceives its neighbourhood through the first depthwise layer. This is the layer that contains the filters we explained earlier ($Sobel_x$, $Sobel_y$, $Identity$). We can see from the output shape that each cell ends up with g*16 values encompassing their representation of the 3×3 neighbourhood they are in, where g represents the number of filters, in our case 3. After this, the NCA is employed in understanding how the cell should process the information and therefore evolve for that one time step. The outcome from the third layer precisely provides the adjustments that need to be implemented across all channels (in our case, 16 channels) for every cell.

This process is synthesized explicitly by Algorithm 1 and represented graphically in Figure 2. In Algorithm 1, the function $apply\_perception\_filters$ applies the two Sobel filters and the identity filter on the current cell and returns the concatenated feature array. The function $forward$ passes the feature array through the two neuron layers and returns the array, containing the the update values for all the cells. The function get_living_mask selects the cells that are alive and

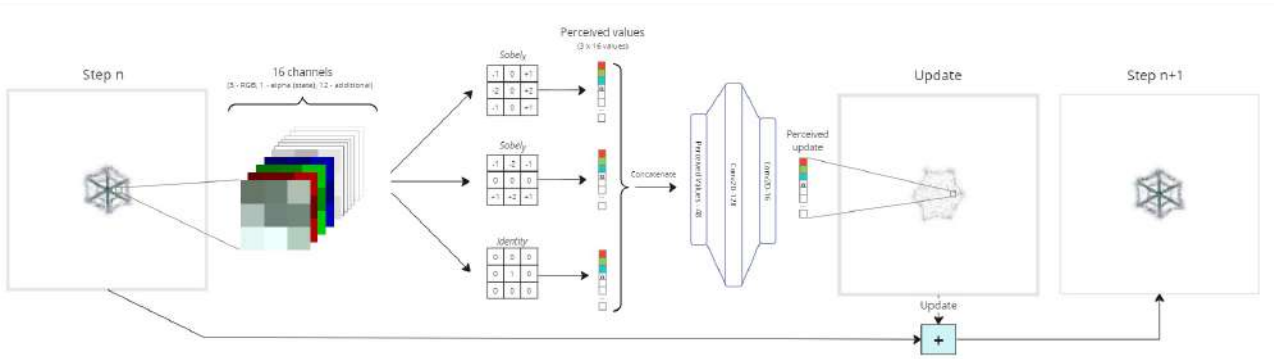


FIGURE 2. Neural Cellular Automaton model.

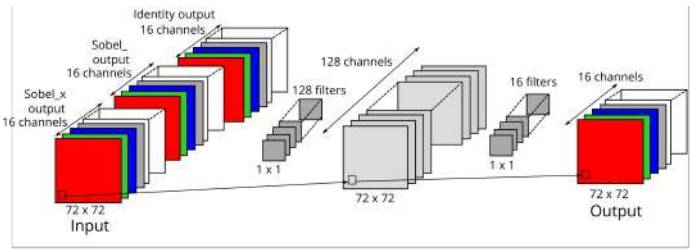


FIGURE 3. Network architecture.

their neighbors, in order to determine the cells which will be updated.

If we consider n the number of cells (in our specific case, 72×72) the complexity of Algorithm 1 is influenced by the selection of alive cells and their neighbors, which is linear in n, the application of the 3 filters on each cell, which is linear in n and the *forward* pass of the filtering result, which is a array of size $n \times 48$. During a *forward* pass for one cell, p multiplications are performed, where p, which is determined by the number of learnable weights, is approximately 8000. Therefore, the overall complexity of the algorithm is within an upper limit proportional to $p \cdot n$.

Algorithm 1 The application of the update rule.

Input: Current state of the automaton (the image)
Output: The updated state of the automaton after one step

function APPLY_UPDATE_RULE(state_nca)
  $cells\_to\_update \leftarrow get\_living\_mask(state\_nca)$
  $features\_vector \leftarrow apply\_perception\_filters(state\_nca)$
  $cell\_update\_values \leftarrow forward(features\_vector)$
  for $cell \in cells\_to\_update$ **do**
    $state\_nca[cell] \leftarrow state\_nca[cell] + cell\_update\_values[cell]$
  end for
  return $state\_nca$
end function

Complexity proportional with $p \cdot n$

During the training step of our experiments, the weights of the NN are not adjusted directly after one update iteration, but after evolving the NCA during a number of steps, i.e., several concatenations of the model in Figure 2. Such an evolution is described by Algorithm 2, in which the variable num_steps is a value between 64 and 96, as recommended in [14]. The complexity is proportional by the number of steps with the complexity of Algorithm 1.

It can be observed by experiments, that this provides a good time interval for the NN to learn. The NN is responsible for the update rule, and it is the one that receives the specified input and calculates the values that modify the cell's features and state. The NN's weights and biases are refined once a cycle is finished, i.e., once the update rule has been applied for 64-96 times, by backpropagation, considering the L_2 loss between the RGB channels of the result of the evolved NCA and the RGB channels of the target image.

Algorithm 2 Evolution of the NCA.

Input: The automaton (nca), number of steps to evolve (num_steps)
Output The evolved nca

function EVOLUTION_OF_THE_NCA(nca)
  for $steps \leftarrow 1; steps \leq num\_steps$ **do**
    $nca \leftarrow apply\_update\_rule(nca)$ ▷ Apply update for each living cell
  end for
  return nca
end function

Complexity proportional with $num\_steps \cdot A1$, where $A1$ is the estimated complexity of first algorithm, Algorithm 1

D. TRAINING OF THE NCA

Once the training is finished, we will simulate a *lifetime* of an NCA similarly: we start from a single living cell and use the trained NN to apply the update rule for all living cells and their neighbours at each point in time. This process can be applied infinitely, as we do not know, how long it has to evolve until it reaches the desired output. In our experiment, we let it

run a maximum of 2000 iterations. During the experiments, we observed that after approximately 300 iterations the images are distorted and the behavior of the automaton does not drastically change, resulting in the further destruction of the image based on similar behaviors observed in previous iterations, see Figure 9. This is a problem which has to be solved in order to obtain a stable result and will be addressed in the following sections.

The model presented above was trained using algorithm 3 on a number of 8000 steps. Each step includes the following iterations:

1) Get the *seed* image: all white illustrated pixels are background (RGB values set to 0) with $\alpha = 0$ (dead) except the one pixel in the middle with alpha set to 1. Notice that a cell is considered living if its alpha channel value exceeds 0.1; the update rule applies to all living cells and their neighbours. The living cell is usually centered, except the cases where the figure in the image does not have values in its center.
2) Let the NCA *live* for several iterations, which implies applying the update rule for each alive cell and its neighbors. We choose the number of iterations randomly between 64 and 96.
3) Calculate the loss, in our case with L_2 function, and adjust the NN's weights.

Algorithm 3 Training of the NCA.

Input the nca, representing the seed image, and the target image
Output the trained nca

function TRAIN_NCA(nca, target)
  for $iteration \in range(0, number\_of\_epochs)$ **do**
    $num\_steps \leftarrow random\ select\ a\ number\ of\ steps\ in\ range(64, 96)$
    $nca \leftarrow evolution\_of\_the\_nca(nca, num\_steps)$
    $loss \leftarrow L_2(nca, target)$
    $update\ weights\ by\ backpropagation$
  end for
  return $trained\_nca$
end function

Complexity proportional with $number\_of\_epochs \cdot (A2 + A1 + n)$, where $A2$ is the estimated complexity of the second algorithm, Algorithm 2

The complexity of Algorithm 3 is proportional to the complexity of Algorithm 2 and the complexity of the *backpropagation*, which is similar to that of the *forward* pass falling within the linear complexity class of type $p \cdot n$, where $p \approx 8000$. Note that we considered a total of 8000 epochs.

This training strategy led us to the compelling results, detailed in the III-A section, and loss plots similar to Figures 4 and 5 for all trained images. Each NCA is trained to recognize and evolve one type of image. For example, if we want our NCA to grow the spider web image, this image serves as the training data for the model during the training process. After the initial image of a single alive cell is passed the number of decided steps through the NCA, the output image is then compared against the preselected target image.

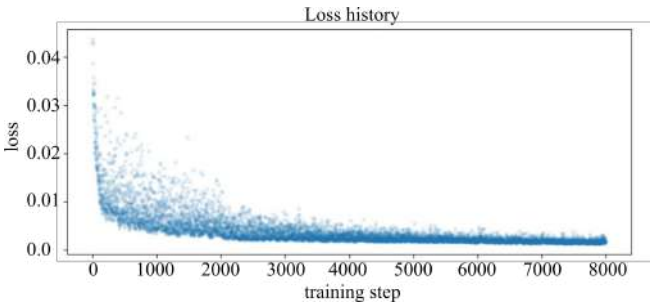


FIGURE 4. Loss history of spiderweb images.

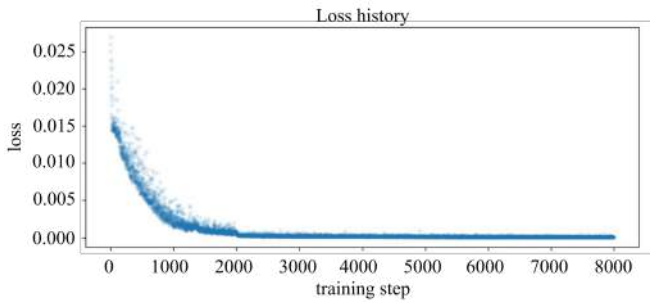


FIGURE 5. Loss history of caterpillar images.

E. STABILIZING TRAINING STRATEGY

The training described in the above approach produced interesting but unsatisfactory outcomes. When the automaton reaches the intended state, which is the image it was trained for, it continues to expand. Moreover, its growing behavior cannot be predicted (one cannot conclude if the image will continually expand, disappear, suffer few alterations, stabilize, etc.). Therefore, the stabilisation goal still needs to be reached.

To solve this problem, two solutions could be implemented:

- A proposed idea was to let the one iteration in the training step to run longer, forcing the NCA also to learn how to remain stable once it reached the desired state.
- A pooling strategy (presented by the original paper [14]).

We followed the second approach in choosing the solution, since training on our machines takes considerable time. This strategy is illustrated in Algorithm 4. How the strategy is applied for one epoch is further depicted in Figure 6.

The complexity of Algorithm 4 is upper bounded in the same way as Algorithm 3, as it only means to apply the pass through the NCA of $batch\_size = 8$ individuals selected from the training pool for a number of epochs.

For these experiments, the training set is represented by the pool of images the cellular automatas is trained on. In our experiments, the size of this pool is 1024. In the beginning, this is composed of 1024 images of the single alive cell. At each step, 8 (or batch_size) images are chosen randomly from this pool. The one with the highest loss value is then replaced by a new image of a single alive cell. This is done in

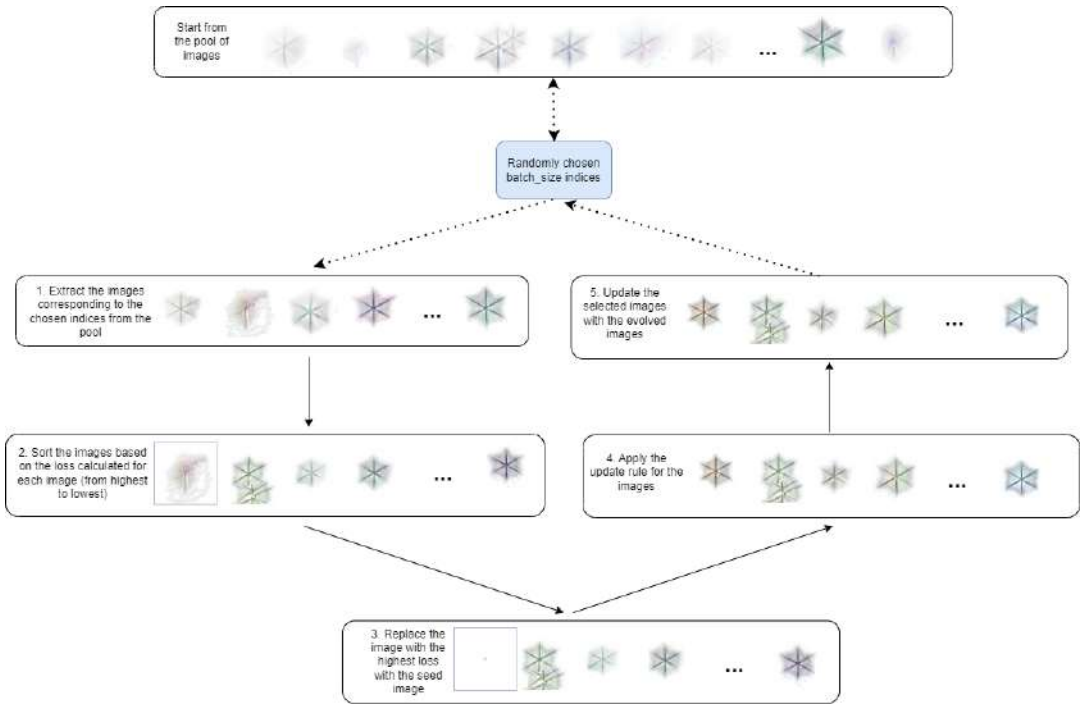


FIGURE 6. Pooling strategy depicted for one epoch.

Algorithm 4 Pooling strategy.

Input the pool of current organisms nca, the target image, the $number\_of\_epochs$, and $batch\_size$ representing the number of individuals that will be chosen
Output the updated pool of organisms

function APPLY_POOLING STRATEGY(nca, $number\_of\_epochs$, $batch\_size$)
  for $iteration \in range(0, number\_of\_epochs)$ **do**
    $individuals \leftarrow random\ pick\ batch\_size\ individuals$
    $worst\_cell \leftarrow cell\_with\_highest\_loss(individuals)$
    $worst\_cell \leftarrow seed\_image$
    $num\_steps \leftarrow random\ select\ a\ number\ of\ steps\ in\ range(60, 90)$
    for $individual \in individuals$ **do**
      $individual \leftarrow evolution\_of\_the\_nca(individual, num\_steps)$
    end for
    $loss \leftarrow L_2(individuals, target)$
    $update\ weights\ by\ backpropagation$
    $updated\_pool \leftarrow replace\ updated\ individuals\ back\ in\ the\ pool$
  end for
  return $updated\_pool$
end function

Complexity proportional with $A3$, where $A3$ is the estimated complexity of the third algorithm, Algorithm 3

order to preserve the capacity of evolving from a single alive cell, as this behavior would be forgotten if we trained only on partially evolved images. After evolving the images from this batch by passing them through the NCA for a number of steps, they are put back into the updated pool, meaning that, the evolved images from the initial batch replace the original images in the pool.

The pooling strategy which replaces the worst individual in the pool can be compared to a simplified genetic algorithm [18], a comparison presented in Table 2.

Similar to NCA, genetic algorithms start with a randomly

TABLE 2. NCA and genetic algorithms comparison.

Training step	Genetic algorithm specifics
Generate a number of seed images (this will be the pool)	First generation creation
For each step, randomly select a batch	Selection method
Calculate accuracy for each chosen image	Fitness function
Discard the lowest-scoring one and replace it with a seed image	-
Let NCA evolve images	Mutation
Replace the old images with the new, evolved images in the pool	New generation creation

generated population, akin to how the automaton begins with a pool of images and uses the update rules also randomly generated. In genetic algorithms, individuals are represented by chromosomes or genes, a string of information influencing how an individual performs a specific task. Afterward, the genetic algorithm selects a batch of individuals for each step to help create a new generation. Usually, in genetic algorithms, a fitness function is then applied to determine the best-performing individuals with a higher chance of combining their genes to create new individuals for the next generation. After this process, sometimes a mutation is applied in order for the genes not to converge and still add some differences among individuals. We can see the resemblance with the aforementioned strategy since this algorithm also selects the best-trained individuals and discards the lowest-scored ones, here with the scope of training the update rule to recognize when the image is starting to destroy itself. In contrast with the classic genetic algorithm, we observe no crossover step - it is not applicable here. Given this training strategy, we

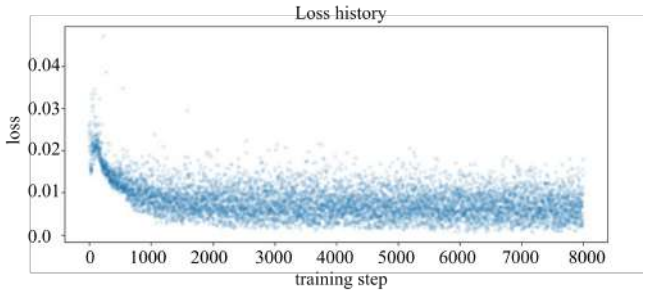


FIGURE 7. Loss history of spiderweb image.

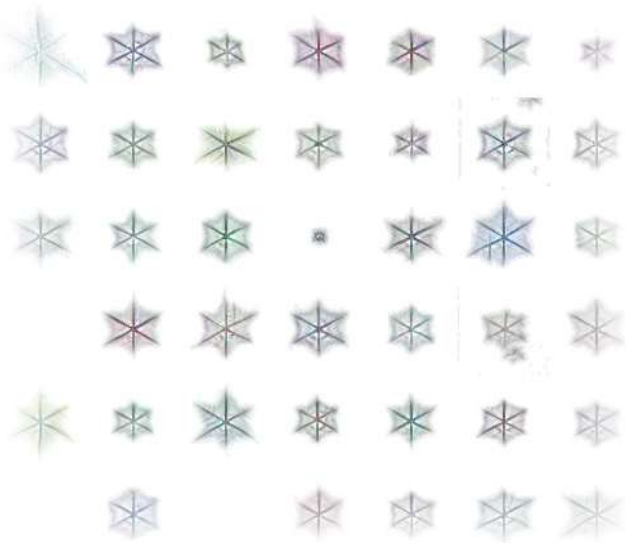


FIGURE 8. Pool of images observed to be in different stages of training.

obtained loss plots similar to Figure 7 for all trained images.

Now, let's further analyze the process of pooling-based training by examining images extracted in different stages of this process. In Figure 8 we sampled 42 pictures from our pool during the training process for the spiderweb image. Therefore we can observe the base idea put in practice: organisms in different stages of evolution will be sampled at a given time. The NN will randomly choose a batch of 8 images from this pool, evolve them according to their learned behavior, and put them back in the pool. Since images are chosen randomly, we reach the desired result: the NN will train on seed/very low evolved images (4th row, 4th column), mildly evolved images (2nd row, 1st column) and more advanced organisms (almost completed images). We can also see its flaws during training, such as duplication (4th and 5th rows, 6th column), disappearing (empty, *dead* images), or noise (3rd row, 6th column), facts also observed during the late stages of training.

The term *evolve* here is essential, since the update rules needed for the model to reach the desired target image are practically learned during the training of the NCA. After training, when these updated rules are fixed, we want to apply them on a new single living cell and see how the NCA

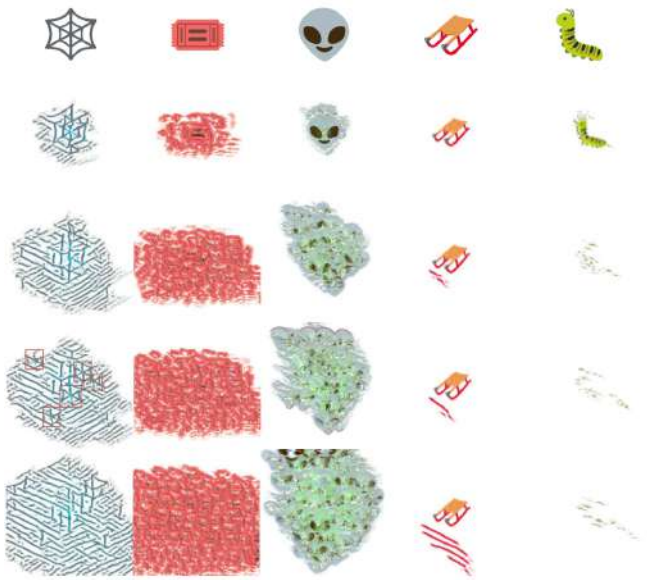


FIGURE 9. Expansion behaviors for figures in the 1st column exhibiting different symmetry properties, after 300, 500, 600 and 800 iterations.

behaves. As a consequence, from now on, we will name the process of applying the NCA in multiple steps on these evolved images *letting the image evolve*.

III. EXPERIMENTS AND DISCUSSIONS

The main focus of this paper is twofold. Firstly, it aims to understand the dynamics of the expanding NCA and the inherent characteristics of the target images, which influence both model growth and stabilization. Secondly, it investigates the effect o filters with varying orientations on the learning and growth processes, highlighting their impact within the model. Therefore, we conducted the following experiments on images with different symmetry and color properties:

Experiment 1: The NCA evolves without having any strategy in-place for the stabilization of the model, as in Algorithm 3, characterized by always training from the seed image. This leads to an unpredictable behavior, which was analyzed on multiple target images. Examples are illustrated in Figure 9.

Experiment 2: The same NCA model is trained using a strategy characterized by pooling as presented in Algorithm 4. This determines the cellular automaton to reach its goal and stop expanding the shape at this point.

Experiment 3: In the model the initial $Sobel_x$ and $Sobel_y$, used to select the features which are then passed to the trainable layers, are replaced by other high-pass filters and the behavior of the NCA is then studied. From this experiment, it results, that in this model directional filters are necessary in order to obtain a consistent growing of the figures, while using the isotropic Laplace filter does not produce any usable result. Nevertheless, combining the directional Sobel filters with Laplace, enhances the results.

Experiment 1 aimed to train the neuronal cellular automaton to learn to grow specific images. However, once this purpose was reached after a number of iterations, we observed that in the following iterations, the NCA continued to expand the figure abnormally, exhibiting different expansion behaviors for different train images. Thus, the experiments continued to study how different image features influence this expansion behavior. The features studied are symmetry, orientation, and contour of the figure in the training images. Section III-B details this experiment's purposes.

The second experiment is set to fix the first experiment's shortcomings, more precisely, to stabilize the model. The NCA has to learn to also keep the image it evolved, no matter how many iterations follow, and to this effect, the pooling training strategy described in Algorithm 3 was used. This experiment and its results are detailed in Section III-C.

In the first experiment, we observed, that the specific properties of the evolved image, influence in a different way the growth and alteration of the image during the different iterations. In the NCA model, a first feature selection is done by combining the outputs of the directional high-pass Sobel masks with the identity mask. As discussed, the $Sobel_x$ and $Sobel_y$ masks have maximum response for vertical respectively horizontal edges. This led us to the assumption, that filter orientation might influence the output of the NCA. Thus, the third experiment was devoted, to analyze the influence of different filter masks on the evolution of the model. This experiment and its results are detailed in Section III-D.

As for the implementation, PyTorch was used to train and evaluate the NCA. The simulations shown in our experiments were done by employing the trained network and starting from a single image with a single alive cell at the center. This image was then fed to the trained NCA for the number of steps we wanted the evolution to take place (each step represents one pass through the network).

A. THE SIMPLIFIED PROBLEM: GROWING NCA

Our first experiment was to train the NCA to learn to grow specific images. The implementation of this plain method as proposed by [14] was presented in Section II, and even though the results yielded did not meet our first expectations, we found them to reveal some interesting insights into how and what the NCA is actually learning, and therefor was worth analyzing.

Our first training image was the spiderweb (Figure 9 1st column), and the parameters used for this training, together with the hardware configurations, are present in Table 3. The same parameters were used for the training with four other shapes, for which the results are also presented and discussed. In order to check if the NN indeed learns to develop from a single seed, we fed it such an image and let it run for multiple iterations, using the rules already learned during training. Intuitively, we can think of this as letting the seed develop using the same set of learned rules for each time step. The results for the five shapes analyzed in this paper are presented in Figure 9. The original images are on the first row, and the following rows present the respective evolved CA after 300, 500, 600 and 800 iterations. It can be seen that for each image, after a certain number of iterations, by continuing to evolve, the CA ruins the target image. The interesting fact is how the different shapes are affected.

B. EXPERIMENTAL ANALYSIS OF VARIOUS SHAPES

Various shapes were chosen for testing in order to run as complete an experiment as possible. The very fact that they were chosen diversely underlines our final ideas.

Spiderweb: The first image we trained on was the spiderweb, Figure 9 - 1st column. The behavior displayed in this image is interesting and relevant since the manner in which it grows shows the following important observation. Once in a while, the new cells created by overgrowing will become and act similar to seeds, displaying a repetitive pattern.

These new *seeds* are highlighted by red squares in the 1st column and 4th row of Figure 9. Another interesting observation is the visual interpretation that diagonal lines seem to be preferred over other lines, similar to how gliders go through Conway's Game Of Life.

Ticket: The symmetric way of development cannot be observed in the growth of the ticket emoticon. We can observe that here, the model destroys itself by reproducing, but individual seeds cannot be observed (Figure 9, column 2). This can also be due to the uniformity and repeatability of the image.

Alien: The same phenomenon described for the spiderweb can also be seen in the training of the alien head, in which the image ruins itself completely by iteration 600, but multiple instances of the alien can be spotted to have been grown out of the border of the initial alien (Figure 9, column 3).

This also displays an interesting, slightly different behavior to the spiderweb and the ticket: it ruins itself later, with the inside of the alien's head remaining stable until iteration 300. On the one hand, this could be explained by the size of the image. However, this cannot be the only reason since, for example, the ticket and the sleigh are similar in size, while their behavior is entirely different.

Sleigh: The sleigh emoticon (Figure 9, column 4) displays, however, surprising results, behaving differently than the other images. Once evolved, the image does not ruin itself. Instead, it grows diagonal extensions at the bottom, which grow and multiply during the following iterations, similar to how gliders travel in the Game Of Life.

Caterpillar: The last image we trained our NCA on was the caterpillar. This is an interesting image since it is the only one we trained on that was destroying itself after evolving the NCA during multiple iterations. We can see that around iteration 300, the head and tail start to disappear.

We wondered why these emoticons behaved differently and eventually self-destructed after going through the NCA. Our first hypothesis was that an important criterion has to do with how symmetric the image is, since the caterpillar and the sleigh display "not-so-repeatable" behavior, and the others that have symmetry in them, do.

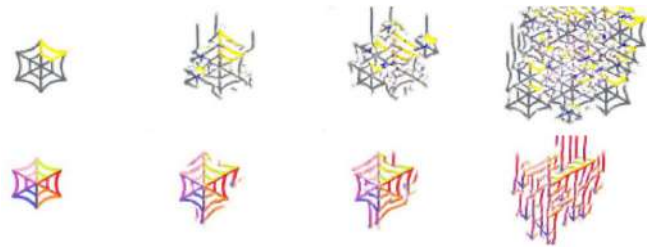


FIGURE 10. Expansion behaviours for the differently colored spiderweb after 100, 250, 300 and 600 iterations.

FIGURE 11. Evolution results for the sleigh image after 200, 500, 750, 1000, 1500 and 2000 iterations.

Symmetry and complexity might matter in the sense that the NCA works by learning different patterns in the data based on a NN. Therefore, it should be easier to learn patterns which are frequently present in the image. Intuitively, if the image is complex, then a lot of *rules* have to be learned in order to recreate the initial shape. On the contrary, if the parts of the image are more similar, exhibiting a greater degree of uniformity, this implies that the same pattern can be recognized more often. As a result, the NCA has more options to reinforce learning the same rule. In this sense, the structure of the image is important. We can also observe this interesting behavior in Figure 10. Here, we can see that based on which part of the image is colored, different patterns emerge. In the 1st row, we can observe the image starting to recreate itself, and ruining the initial organism by iteration 300, while in the 2nd row, we can see that the initial structure is preserved despite some expansion being observed by iteration 300. Additionally, we can observe that this growth sticks to the initial pattern and that the colors hold onto their nearby counterparts rather than expanding haphazardly.

The presence or absence of a colored contour of the figure seems to influence the evolution over time, which can be clearly observed in the alien figure's case. It can be seen, that in the image of the alien, copies of itself appear around iteration 250 outside the initial image, along the border of the figure. Adding a different colored contour to the figure will impact the way the image evolves compared to the initial experiment. Figure 12 illustrates an interesting behavior: both CA are destroying themselves. If we take a look closer, the genuine alien evolves duplicates of himself from the border. While this is partly true for the bordered alien, we can see that this phenomenon happens later. The disturbances from the first iterations are duplicates of the border rather than the rebirth of the alien. This is of course, because the seed is not of red colour, so it is not confused with a point of start. We also chose the same image but bordered so that other different factors aren't taken into consideration. These observations

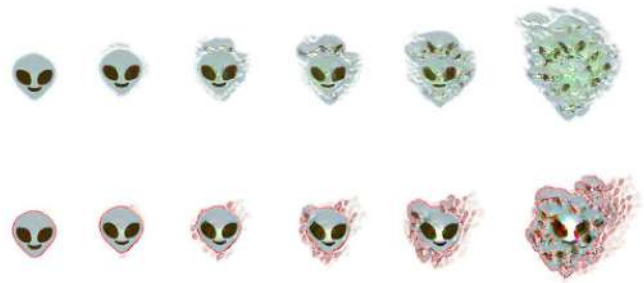


FIGURE 12. Evolution results for the unbordered and of the bordered alien after 100, 150, 200, 250, 300 and 500 iterations.

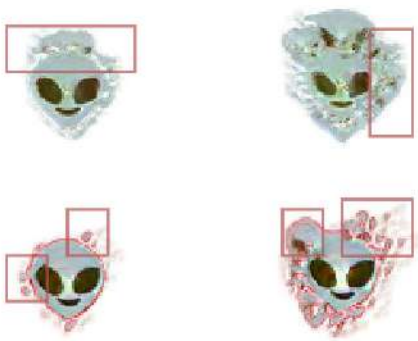


FIGURE 13. Details in the multiplication of the shape during the evolution of the unbordered and of the bordered alien after 200 respectively 300 iterations.

are detailed in Figure 13.

Another interesting problem is the way, in which lines propagate in the image during the evolution of the NCA. Diagonal lines might be preferred due to being the easiest to create. For example, in Conway's Game of Life, which works based on a Cellular Automaton, the glider is the easiest to find propagating model, which needs only five alive cells to start propagating diagonally. This preference for the diagonal lines can be observed in the sleigh emoticon, illustrated in Figure 11, and the uncolored spiderweb.

One explanation for the apparent preference for diagonal lines can reside in the orientation of the Sobel filters. In the case of horizontal, respectively vertical lines, one of the filters will produce a high response, while the other has a nearly 0 response. This will result in an unbalanced feature vector. In the case of diagonal lines, both filters exhibit similar responses. To test if the orientation of the filters influences the way in which the NCA evolves, a third experiment was conducted (Section III-D).

C. THE STABILITY OF NCA

The second experiment using the pooling strategy yielded the desired results. The NCA trained by this strategy is able to grow from one living cell and, once reaching the targeted image, stabilizes and the image is not expanded further.

In this section, the success of the algorithm described in Section II-E is presented on images sampled during and after the pool-based training. As for the comparison with

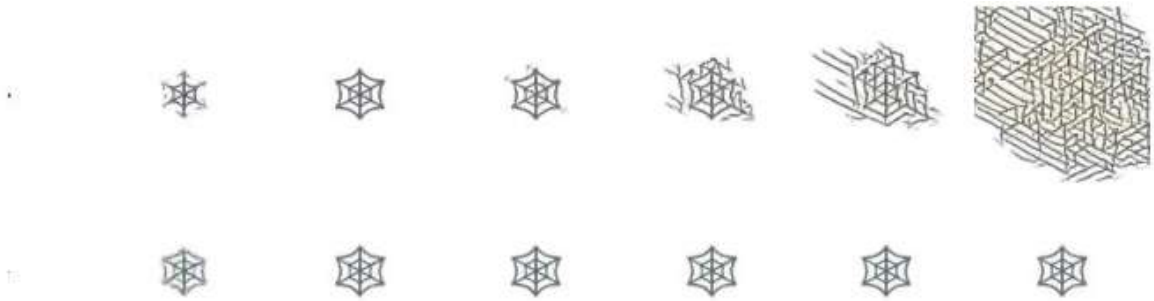


FIGURE 14. Comparison of an organism grown after training with Algorithm 3 (up) and respectively the Algorithm 4 (bottom) steps 0, 50, 100, 150, 250, 350 and 1000.

the first approach without pooling, Figure 14 illustrates how the NCA without pooling expands the spiderweb image in a span of 1000 steps post-training and how the NCA using the pooling algorithm stabilizes, keeping the figure unchanged even while further evolving the NCA.

In order to further analyze how the training strategy influenced the behavior of our NCA the spiderweb image sampled during training, more precisely from batch no. 700 out of 8000, is presented in Figure 15. In the first row are represented images from the batch, just before evolving them through the NCA, while in the second row are illustrated the results of the NCA for each of the samples in the first row.

In the 1st column of Figure 15 we observe that the NN can guide the NCA to evolve the seed towards an unfurnished final state of the target image. In the next six columns, the advantages of using the pool-based approach become clear: the NN has learned when to stop expanding the image. Due to the fact that this batch is extracted in the early stages of the training, we see a duplication problem in the last column where 4 intertwined spiderwebs in the input image are evolved simultaneously without the NCA realizing the need to remove the excess ones. However this problem is remedied in later stages of training.

D. DIRECTIONAL VERSUS ISOTROPIC FILTERS

In order to determine the influence of edge orientation we performed an experiment in which we replaced the original $Sobel_x$ and $Sobel_y$ filters with their, by 45^0 rotated, correspondents. The results on two of the described emoticons showed a similar behavior of the NCA, as for the original filters. After a number of iterations, the figures were distorted in a manner, which differed only by the direction of the distortions. This is shown in Figure 16 for the spiderweb and for the sleight. In both cases, pooling algorithm 4 is necessary for stabilization.

We were interested to see, if better results could be obtained, by considering an isotropic high-pass filter. Therefore, we replaced the pair of Sobel filters with a Laplace filter mask. The results obtained were unexpected in the sense that the NCA could not learn to regrow the images in either of the scenarios, i.e., with or without pooling. In Figure 17, the result of evolving the NCA for the spiderweb emoticon by Algorithm 3 are presented for iterations 1, 50, 100, 250, 500, 1000.

By using the pooling algorithm 4, we observed that none of the samples in the pool gets similar to the expected image and in later iterations most of the samples are replaced by the seed, while the others are only distorted examples, similar to those in the figure.

Although the results obtained when using the Laplace filter were not encouraging, we hypothesized that this filter could still augment the NCA by offering supplementary information, when utilized in conjunction with the $Sobel_x$ and the $Sobel_y$ filters. Thus we used them together for evolving and training the NCA. The obtained results are very promising as can be observed in Figure 18. It can be noticed that between iteration 250 and 500, the image is almost perfectly restored, with minimal distortions. Only during later iterations, the distortions become more pronounced. This is a first indication that, by integrating the Laplace filter, and thus providing additional features, the NCA can learn in a more reliable way how to evolve towards the target image. We still have to investigate, if we can optimize the process, as to avoid the elaborated pooling process altogether.

E. HARDWARE DETAILS

To ensure reproducibility, it is also essential to detail the technical specifications of the device on which the experiments were performed. Table 3 lists the laptop specifications used for training alongside with the specific input structure of the NCA.

TABLE 3. Hardware and software details

Type	Parameter	Value
Hardware	Processor	AMD Ryzen 7 5800U
	RAM	16.0GB (14GB usable)
	Trained on	CPU
NCA Specifics	Batch size	8
	Cells dropout rate	0.5
	Optimizer	Adam
	Loss function	L_2
	Number of generations	8000
	Iterations/generation's individual	random(64, 96)
Time	**Total training time 12h-16h**	

FIGURE 15. Batch 700. Based on the approach presented in Algorithm 4, the randomly selected pool of individuals before and after training of batch number 700.

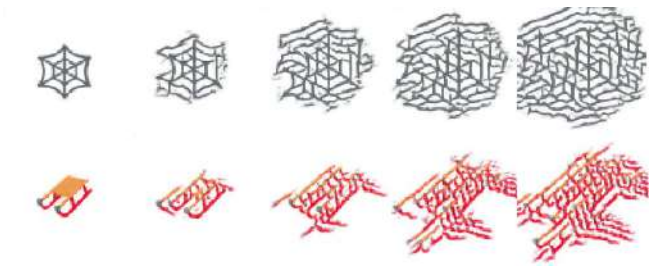


FIGURE 16. Evolution results for NCA using rotated Sobel filters for the spiderweb and the sleight: original and after 300, 500, 600 and 800 iterations.

FIGURE 17. Evolution results for NCA using the Laplace filter for the spiderweb after 1, 50, 100, 250, 500 and 1000 iterations.

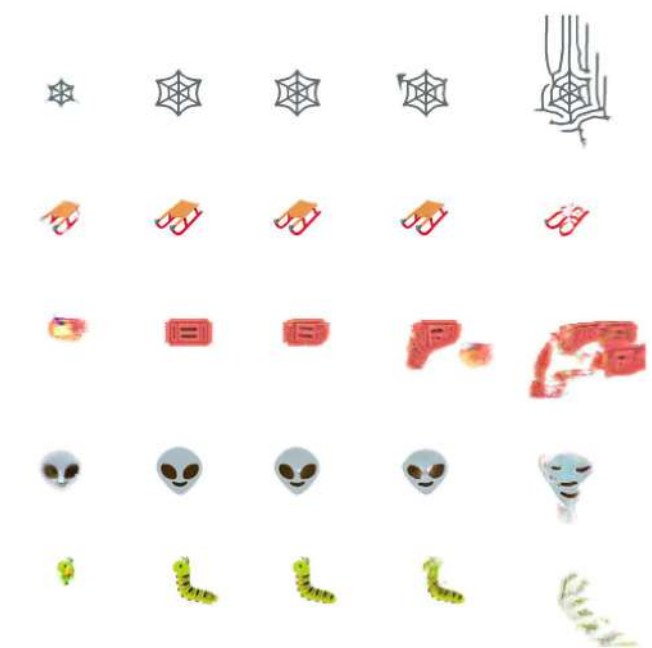


FIGURE 18. Evolution results for NCA using the $Sobel_x$, $Sobel_y$ and the Laplace filters after 50, 100, 250, 500 and 1000 iterations.

IV. CONCLUSIONS

The field of NCA is expanding. Models exploiting its behavior, such as robots training for regeneration and movement using cellular automata, research focussing on programmed cell death, cell division, cell grafting, models responsive to genomically-coded and environmental signals [19] are currently studied and applied in different areas. Moreover, applications in static and dynamic texture generation [20] have been proven more efficient than former solutions, opening this field for research. Other real-life applications of cellular automata include fine material migration [21] and spatial-temporal load forecasting [22]. Thus, neural cellular automata serve as an active research field with numerous future prospects.

During our study on cellular automata representing images of 2D objects we observed various behaviors, thus this article also aims to address some hypothetical issues. The fundamental issue is what causes such an image to self-destruct after evolving the NCA for a longer period and also what are the primary aspects of an image that affect how it evolves.

The main advantage of such an automaton is the possibility of reproducing a whole shape from one seed. It also opens the way to a series of applications, like automatic repair of structures or even reproduction of different shapes, patterns or textures starting from one or several sparse starting points. The disadvantage at the moment is that the automaton has to be trained separately for each form considered. But it is a matter of current research on how to train an automaton, to choose what shape to generate, depending on the input it gets in the beginning.

An important contribution of the study is that various properties were studied - symmetry, image complexity, image orientation, and image evolution considering the object's contours, and some relevant conclusions were drawn about the growth principles of the images considered. This growth maintains the original pattern, and the colors maintain their close neighbors rather than spreading randomly.

Another significant contribution is the study of the influence of different filters on the stabilization process, which has been thoroughly discussed. Concerning the influence of the orientation of the high-pass filters used for feature extraction, we discovered that for this architecture, oriented filters are essential in the evolution of the NCA, as our experiment

using the isotropic Laplace filter did not produce any useful results. Moreover, we found out that significant improvement in stability can be obtained by combining the Sobel filters with the Laplace, a result that still needs further investigation. Still, isotropic filters can perform well in more elaborated NCA models, as presented in [23].

A series of technical details about what is happening in the background in the functioning of the NCA were presented in an algorithmic, easy to follow manner.

This study is an explanatory study that deepens understanding of previous approaches, tests theories or hypotheses, offers various perspectives for making decisions, can increase the validity of research, and can also be utilized in conjunction with other research designs.

REFERENCES

[1] J. Kari, "Theory of cellular automata: A survey," Theoretical Computer Science, vol. 334, no. 1, pp. 3–33, 2005.
[2] M. Khanjary, "Cellular learning automata: Review and future trend," Computational Vision and Bio-Inspired Computing: Proceedings of ICCVBIC 2021, pp. 229–238, 2022.
[3] E. W. Weisstein, "Cellular automaton," https://mathworld. wolfram. com/, 2002.
[4] W. Li, N. Packard, et al., "The structure of the elementary cellular automata rule space," Complex systems, vol. 4, no. 3, pp. 281–297, 1990.
[5] A. Hernandez, A. Vilalta, and F. Moreno-Noguer, "Neural cellular automata manifold," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 10020–10028, 2021.
[6] E. W. Weisstein, "Elementary cellular automaton," https://mathworld. wolfram. com/, 2002.
[7] S. Wolfram, A New Kind of Science. Wolfram Media, 2002.
[8] K. Bhattacharjee, N. Naskar, S. Roy, and S. Das, "A survey of cellular automata: types, dynamics, non-uniformity and applications," Natural Computing, vol. 19, pp. 433–461, 2020.
[9] D. A. Zaitsev, "k-neighborhood for cellular automata," CoRR, vol. abs/1605.08870, 2016.
[10] B. Breckling, G. Pe'er, and Y. Matsinos, Cellular Automata in Ecological Modelling, pp. 105–117. 01 2011.
[11] A. Mordvintsev and E. Niklasson, "μnca: Texture generation with ultra-compact neural cellular automata," CoRR, vol. abs/2111.13545, 2021.
[12] E. Niklasson, A. Mordvintsev, E. Randazzo, and M. Levin, "Self-organising textures," Distill, 2021.
[13] K. Horibe, K. Walker, and S. Risi, "Regenerating soft robots through neural cellular automata," CoRR, vol. abs/2102.02579, 2021.
[14] A. Mordvintsev, E. Randazzo, E. Niklasson, and M. Levin, "Growing neural cellular automata," Distill, 2020. https://distill.pub/2020/growing-ca.
[15] S. Sudhakaran, E. Najarro, and S. Risi, "Goal-guided neural cellular automata: Learning to control self-organising systems," 2022.
[16] S. Sudhakaran, D. Grbic, S. Li, A. Katona, E. Najarro, C. Glanois, and S. Risi, "Growing 3d artefacts and functional machines with neural cellular automata," CoRR, vol. abs/2103.08737, 2021.
[17] R. Gonzales and R. Woods, Digital image processing 4th edition. Pearson, 2018.
[18] A. Lambora, K. Gupta, and K. Chopra, "Genetic algorithm-a literature review," in 2019 international conference on machine learning, big data, cloud and parallel computing (COMITCon), pp. 380–384, IEEE, 2019.
[19] J. Stovold, "Neural cellular automata can respond to signals," in The 2023 Conference on Artificial Life, MIT Press, 2023.
[20] E. Pajouheshgar, Y. Xu, T. Zhang, and S. Süsstrunk, "Dynca: Real-time dynamic texture synthesis using neural cellular automata," 2023.
[21] R. Castro, R. Gomez, and L. Arancibia, "Fine material migration modelled by cellular automata," Granular Matter, vol. 24, p. 14, 02 2022.
[22] S. Zambrano-Asanza, R. Morales, J. A. Montalvan, and J. F. Franco, "Integrating artificial neural networks and cellular automata model for spatial-temporal load forecasting," International Journal of Electrical Power Energy Systems, vol. 148, p. 108906, 2023.
[23] A. Mordvintsev, E. Randazzo, and C. Fouts, "Growing isotropic neural cellular automata," 2022.

SORANA CATRINA studies at Transilvania University of Brașov and is starting her third year pursuing a Bachelor's degree in computer science. She is passionate about quantum physics and quantum programming, participating in plenty of courses, contests, and Summer Schools on this topic. Alongside quantum, her interests revolve around algorithms, machine learning, and data structures.

MIRELA CATRINA is a student at Transilvania University of Brasov, currently starting the last year of her Bachelor's degree in Computer Science. She participates in numerous algorithmic contests, Summer Schools, and projects, affirming an interest in machine learning, algorithms, and data structures. Recent projects include a daily meal plan recommendation platform, a movie recommendation platform, Window Query Processing in Fast Dynamic Linear Quadtrees, and The Influence of Genetic Algorithms on hyperparameter optimization for neural networks.

ALEXANDRA BĂICOIANU is a research engineer in informatics and received her Ph.D. in 2016 from Babes Bolyai University, Cluj-Napoca. She has been a lecturer at Transilvania University of Braşov since 2017, teaching various courses and seminars. She has published more than 30 scientific papers and is the co-author of 6 books. Also, she supervised tens of graduation and dissertations thesis, programming training courses, programming Summer Schools, and code/tech Camps, some of them in collaboration with IT companies. She is also a member of the department's Machine Learning research group, founded in 2018. Her research interest and expertise are in the field of machine learning, formal languages and compilers, algorithms, remote sensing and Earth observation data, autonomous driving, electric and hybrid vehicles.

IOANA CRISTINA PLAJER received the B.E. and the M.S degree in computer science from the University of Bucharest, Romania, in 1997 and 1998 respectively. She received her Ph.D. degree in computer science at the Transilvania University of Braşov, Romania, in 2011. She is currently a Lecturer with the Faculty of Mathematics and Computer Sciences of the Transilvania University, Brasov, Romania. She is also a member of the department's Machine Learning research group, founded in 2018 and part of the project *Artificial intelligence and Earth observation for Romania's agriculture (AI4AGRI)*. Her research interests include machine learning, image processing, spectral imaging and remote sensing, formal languages, algorithms, and data structures.

Towards Generating Executable Metamorphic Relations Using Large Language Models

Seung Yeob Shin✉[1][0000−0001−9025−7173], Fabrizio Pastore[1][0000−0003−3541−3641], Domenico Bianculli[1][0000−0002−4854−685X], and Alexandra Baicoianu[2][0000−0002−1264−3404]

[1] SnT Centre, University of Luxembourg, Luxembourg, Luxembourg
{seungyeob.shin, fabrizio.pastore, domenico.bianculli}@uni.lu
[2] Siemens Industry Software, Braşov, Romania
alexandra.baicoianu@siemens.com

Abstract. Metamorphic testing (MT) has proven to be a successful solution to automating testing and addressing the oracle problem. However, it entails manually deriving metamorphic relations (MRs) and converting them into an executable form; these steps are time-consuming and may prevent the adoption of MT. In this paper, we propose an approach for automatically deriving executable MRs (EMRs) from requirements using large language models (LLMs). Instead of merely asking the LLM to produce EMRs, our approach relies on a few-shot prompting strategy to instruct the LLM to perform activities in the MT process, by providing requirements and API specifications, as one would do with software engineers. To assess the feasibility of our approach, we conducted a questionnaire-based survey in collaboration with Siemens Industry Software, a worldwide leader in providing industry software and services, focusing on four of their software applications. Additionally, we evaluated the accuracy of the generated EMRs for a Web application. The outcomes of our study are highly promising, as they demonstrate the capability of our approach to generate MRs and EMRs that are both comprehensible and pertinent for testing purposes.

Keywords: metamorphic testing · large language model · LLM · executable metamorphic relations

1 Introduction

In many sectors, software is typically verified with executable test cases that, at a high level, consist of a set of inputs provided to the software under test (SUT) and a set of test assertions verifying that the SUT outputs match the expected results. However, defining executable test cases is expensive because of the many intellectual activities involved. This cost limits the number of test cases that can be implemented. Moreover, software faults are often subtle and triggered by a narrow portion of the input domain. Therefore, they are detected only after exercising the SUT with a large set of test inputs.

Although several tools for the automated generation of test inputs have been developed [9, 10], they can only identify crashes or regressions faults, because they lack the capability of determining what the expected output for the software should be. Further, manually specifying test assertions for thousands of automatically generated test inputs is practically infeasible. The impossibility to programmatically derive expected outputs is known as the *oracle problem* [3].

Metamorphic testing (MT) has recently gained success as a solution to address the oracle problem in many contexts [6]. MT relies on the concept of metamorphic relations (MRs), which describe the relationships between input transformations and expected output changes, serving as oracles in software testing. Briefly, to perform MT on the SUT, engineers first need to derive MRs from the SUT's requirements and then convert the derived MRs into an executable form to determine the test outcome. These steps require substantial manual effort, which increases the MT cost and may prevent its adoption.

Many MT approaches have been developed to test various SUTs, including search engines [26], Web applications [5], and embedded systems [1]. These methods, however, require significant manual efforts to define MRs, which limits their scalability, particularly in cases where SUT executions are lengthy and expensive. A few recent studies [12, 25] have investigated using Large Language Models (LLMs) to automate MR derivation directly from LLM knowledge bases. However, they currently only work with SUTs known during LLM training and are not effectively applicable to test new, unseen systems. In addition, to the best of our knowledge, there is no existing work that aims at automating the conversion of MRs into an executable form, hereafter referred to as EMRs (Executable MRs), which is an essential step to fully automate the MT process.

Contribution. In this paper, we propose an approach for automatically deriving EMRs from the SUT's requirements using LLMs. Differently from existing work, our approach does not merely involve querying an LLM for MRs based solely on the LLM knowledge base. Instead, we rely on prompt engineering practices to teach the LLM both (1) the specifications of the SUT, which are necessary to derive MRs, and (2) the DSL to use for EMRs, which is necessary to enable MR execution.

We evaluated the feasibility of our approach through two experiments: (1) by conducting a questionnaire-based survey in collaboration with Siemens Industry Software, a worldwide leader in providing software and services across industry domains (hereafter, SISW), involving four of their software applications, and (2) by assessing the correctness of the generated EMRs for a Web application.

In our experiments, we used OpenAI GPT-3.5 and GPT-4 [15] as LLM, accessed (to provide prompts and generate responses for deriving MRs and subsequently converting them into EMRs) through the web interface of ChatGPT [14]. When converting MRs into EMRs, we instructed the LLM to use SMRL [13] as a domain-specific language (DSL) for specifying MRs in an executable form. The results are promising, indicating that the generated MRs and EMRs are understandable and relevant for testing, and that the EMRs were correctly converted from their corresponding MRs.

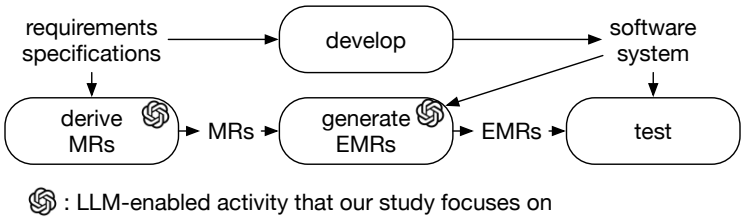


Fig. 1. Our approach for generating EMRs using LLMs.

Organization. The remainder of this paper is structured as follows: Section 2 describes our approach. Sections 3 and 4, respectively, present our experiment design and results. Sections 5 discusses threats to the validity and research opportunities. Section 6 surveys related work. Section 7 concludes the paper.

2 Approach

This section describes our approach to automatically deriving EMRs from the requirements of the SUT. Specifically, our approach relies on an LLM to automate the MT process, particularly by leveraging the LLM's capabilities in understanding and processing natural language documents, identifying relevant information, and synthesizing that information into structured forms (i.e., MRs and EMRs) useful for software testing.

2.1 Overview

Our approach, shown in Fig. 1, advances the MT process by introducing two automated activities: "derive MRs" and "generate EMRs". The "derive MRs" activity takes as input requirements specifications (in the form of specification documents) and outputs (MRs in natural language). Unlike existing work that queries an LLM for MRs based on the LLM built-in knowledge, we provide SUT specification documents to the LLM. This enables our approach to derive SUT-specific MRs, which would otherwise be unattainable because the LLM lacks awareness of the specific software properties that need to be considered. The "generate EMRs" activity takes as input MRs and the SUT (depicted as "software system" in Fig. 1). It outputs EMRs, which are executable programs used to verify that the SUT satisfies the properties captured by the MR. In the following subsections, we further describe these activities, focusing on their implementation as adopted in our preliminary experiments (see section 3).

2.2 Deriving metamorphic relations

We aim at automatically deriving MRs from the provided requirements specification documents. To this end, our approach relies on ChatGPT and develops

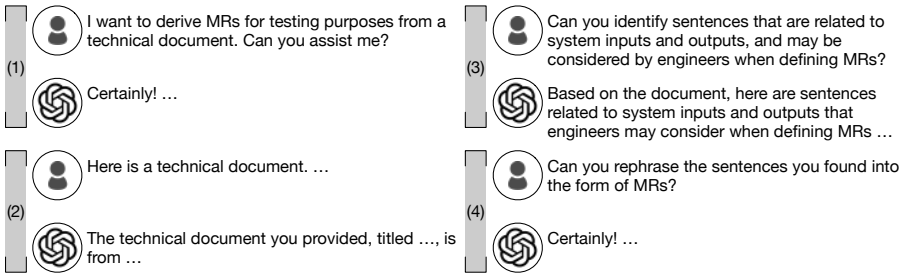


Fig. 2. Prompts to derive MRs from the SUT's requirements, grouped by phases in the conversation: (1) setting the context, (2) providing requirements specification document(s), (3) identifying relevant sentences, and (4) rewriting these sentences into MRs.

a prompt sequence for automation. Since deriving MRs from requirements is a complex task, when using ChatGPT for this purpose, we break the task down into simpler, decomposed steps[3].

Figure 2 shows the prompts that derive MRs from the requirements of the SUT. We have omitted some parts of the text, indicated by "...", to focus on the essential prompts of our work. Note that the complete prompts are available online [17]. As shown in Fig. 2 (1), our prompt sequence begins by setting the context, i.e., deriving MRs, to guide ChatGPT for the following interactions. The remaining sequence, which decomposes the task of deriving MRs, consists of prompts directing ChatGPT to perform simpler steps, as described below. First, our approach instructs ChatGPT to read a requirements document in order for ChatGPT to access the requirements needed for defining MRs for the SUT (see Fig. 2 (2)); in response, ChatGPT provides a short summary of it. Second, given the requirements document, ChatGPT is directed to find sentences that are specifically related to the SUT's inputs and outputs, as well as those that engineers may consider when defining MRs (Fig. 2 (3)). This step is important because MRs fundamentally describe how outputs should be changed in response to specific changes in inputs. The last step then asks ChatGPT to write MRs based on the identified relevant sentences (Fig. 2 (4)).

For example, consider a scenario where our approach is used to derive MRs from the requirements document of an online shopping system that includes a `searchItem` function. After setting the context and reading the requirements document, ChatGPT could identify requirement R1 in Table 1, which concerns the advanced search options in the `searchItem` function. Given requirement R1, ChatGPT then proceeds to define an MR, referred to as MR1 in the table. MR1 specifies the relationship between the outputs of an initial search query and those of a subsequent query where additional filters are applied.

[3] We note that when we passed a single, monolithic query to ChatGPT for deriving MRs, it was unable to derive meaningful MRs.

Table 1. An example requirement for an online shopping system and an MR derived from this requirement.

R1	The system should provide advanced search options to allow users to refine their searches based on specific attributes such as price range, category, brand, customer ratings, and availability.
MR1	For a given search query, applying additional filters (e.g., narrowing down by category or price range) should reduce the number of search results or refine them to match the filters more closely.

2.3 Generating executable metamorphic relations

We ask ChatGPT to convert MRs written in natural language into EMRs specified using SMRL [13], a DSL developed for specifying EMRs. SMRL provides MR-specific language constructs, which are built on Java, to specify EMRs. Table 2 presents a subset of SMRL constructs. The construct `Input(int i)` refers to the i-th sequence of actions performed by a user with the SUT. The construct `Output(int i)` refers to the sequence of outputs produced by the SUT in response to `Input(i)`. The construct `CREATE(Object y, Object x)` defines `y` as a copy of `x` and satisfies `y = x`. The last three rows in the table capture Boolean expressions corresponding to implication (`IMPLIES()`), negation (`NOT()`), and disjunction (`OR()`). Using these constructs, engineers can specify EMRs in terms of inputs, input transformations, outputs, output changes, and logical expressions to define the expected relations among them. In addition, SMRL is already integrated into the MST-wi framework [5], which generates executable Java code from specified EMRs to automatically perform MT.

Figure 3 depicts a sequence of prompts that converts MRs into EMRs, selectively omitting text using "..." for a clearer exposition. Our approach involves instructing ChatGPT to understand SMRL and to use SMRL for converting MRs into EMRs. Specifically, as shown in Fig. 3 (1), when using ChatGPT, we first set the context by indicating that the subsequent interactions will aim at converting MRs into EMRs. We then provide ChatGPT with the notations for the SMRL constructs (see Table 2), enabling it to use the SMRL constructs when converting MRs into EMRs (Fig. 3 (2)). To transform MRs into EMRs in a consistent form, we provide ChatGPT with an EMR template (Fig. 3 (3)). Subsequently, we apply the few-shot learning strategy [21] to instruct ChatGPT

Table 2. A subset of the SMRL constructs.

Construct	Description
`Input(int i)`	returns the i-th input sequence
`Output(int i)`	returns the sequence of outputs generate by Input(i)
`CREATE(Object y,Object x)`	creates y as a copy of x
`IMPLIES(boolean x,boolean y)`	is equivalent to the Java expression `!x || y`
`NOT(boolean x)`	is equivalent to the Java expression `!x`
`OR(boolean x,boolean y)`	is equivalent to the Java expression `x || y`

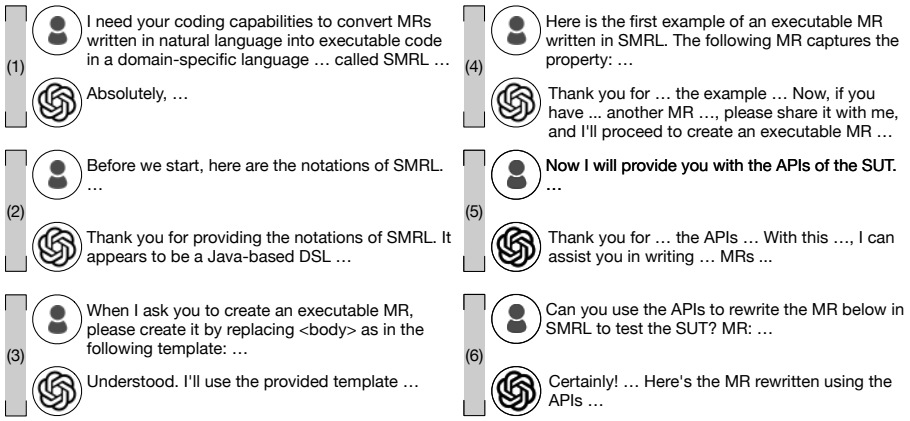


Fig. 3. Prompts to convert MRs into EMRs, grouped by phases in the conversation: (1) setting the context, (2) learning the syntax of SMRL, (3) instructing the output format, (4) few-shot learning of SMRL, (5) learning the SUT's APIs, and (6) transforming MRs into EMRs.

to understand SMRL, by providing three pairs of MR and corresponding EMR from prior work [5] (Fig. 3 (4)). The few-shot learning strategy is suitable in our study context, as SMRL is a state-of-the-art DSL with a limited number of examples available, and it is built on Java, which ChatGPT already knows. In addition, we further instruct ChatGPT to understand the SUT's APIs by providing the documentation and signature of the API methods (Fig. 3 (5)). For example, to test the Web application described in prior work [5], which includes 38 APIs, we need to provide ChatGPT with the details of these APIs. Last, we then ask ChatGPT to transform MRs into EMRs using SMRL and to invoke the APIs in the EMRs for testing the SUT (Fig. 3 (6)).

Figure 4 shows an EMR generated by our approach, corresponding to the MR described in Table 1. This EMR specifies that adding filters to a search query should either reduce the number of results or refine them to better match the filter criteria. Below, we provide the comments generated by ChatGPT to explain the EMR statements; each corresponds to one annotation `//(#)` in Fig. 4.

(1) & (2): Iterate over all actions within the input sequence to identify search actions. (3): Retrieve the original search results before any additional filters are applied. (4): Iterate over possible filter types (e.g., categories, price ranges) that can be applied to the search. (5): Create a new input sequence where a specific filter is applied to the identified search action. (6): Ensure the new input sequence (`Input(2)`) is properly created with the applied filter. (7): Check that the same filter hasn't already been applied to the search action to avoid redundant checks. (8): Verify that applying the filter results in fewer search results compared to the original, indicating a refinement of the search output. (9): Alternatively, check that the results are more relevant to the applied filter, even if the number of results does not decrease. This could involve

```
MR {{
for (Action searchAction : Input(1).actions()) { //(1)
  if (!isSearchAction(searchAction)) continue; //(2)
  var originalResults = Output(Input(1), searchAction.getPosition()); //(3)
  for (var filterType : getFilterTypes()) { //(4)
    var filteredInput = applyFilter(Input(1),searchAction.getPosition(),filterType);//(5)
    IMPLIES(
    CREATE(Input(2), filteredInput) && //(6)
    notSameFilterApplied(searchAction, filterType), //(7)
    OR(
    fewerResults(Output(Input(2), searchAction.getPosition()), originalResults), //(8)
    moreRelevantResults(Output(Input(2), searchAction.getPosition()), originalResults,
         filterType) //(9)
    )
    );//end-IMPLIES
  }//end-for
}//end-for
}}//end-MR
```

Fig. 4. An EMR generated by ChatGPT based on the MR presented in Table 1.

comparing the characteristics of the search results against the filter criteria to ensure they match more closely.

We note that we did not provide ChatGPT with the API details of the online shopping system. Hence, ChatGPT autonomously created some function invocations, such as `applyFilter()` and `notSameFilterApplied()`, to implement the EMR using SMRL. This capability of ChatGPT is desirable in test-driven development practices as it helps engineers identify which APIs are needed to test the SUT at early development stages.

3 Experiment Design

To assess the feasibility of our approach, we have carried out two experiments, namely EXP1 and EXP2. EXP1 aims at collecting feedback from practitioners. EXP2 aims at evaluating the correctness of generated EMRs.

EXP1 - SISW applications. To assess whether our approach can practically help software practitioners, it is important to collect and analyze their perceptions on our work. In collaboration with SISW, we have conducted a questionnaire-based survey to gather their feedback on the MRs and EMRs obtained by applying our approach to their software applications. Due to confidentiality reasons, SISW was not able to share the requirements and the APIs of their applications; instead, SISW selected four non-confidential technical documents based on their interests. These documents describe their applications (modelling and simulation) for marine design [19], wind turbine [18], aircraft propulsion [20], and aero-acoustics [4]. They provided us with these documents (83 pages in total) as requirements specifications. As remarked above, in absence of appropriate APIs, ChatGPT suggests invoking functions that should be implemented to automate the execution of the MR. Further, despite the lack of APIs,

considering the preliminary nature of this work, these subjects still enable us to gather feedback from practitioners on how well ChatGPT can generate EMRs.

In EXP1, we relied on GPT-4 because, at the time we designed our experiments (January 2024), it was the only available top-5 leaderboard LLM capable of processing PDF documents[4]. Note that our questionnaire-based survey study does not enforce practitioners to respond to all questions. We adopted this survey strategy to obtain higher quality responses to the questions that practitioners feel confident in answering, since the participating practitioners have varying levels of expertise across the four applications developed by SISW.

The questionnaire for MRs (resp. EMRs) contains three statements for each MR (resp. EMR). It requests practitioners to indicate their level of agreement with the statements on a Likert scale (i.e., strongly agree, agree, neutral, disagree, and strongly disagree). We designed the statements in our questionnaire-based survey, drawing inspiration from Rogers' theory of innovation diffusion [16]. In this theory, the following five characteristics, based on practitioners' perceptions, are introduced for their impact on the adoption of innovative solutions:

- Complexity: Reflecting the extent to which an innovation is perceived as challenging to understand or implement.
- Compatibility: Referring to the extent to which an innovation aligns with practitioners' existing values, experiences, and needs.
- Trialability: Indicating the extent to which an innovation can be tried on a limited scale or adopted incrementally.
- Observability: Pertaining to the visibility of the results of an innovation to others.
- Relative advantage: Describing the perception of an innovation's superiority compared to what is currently used.

Out of these five characteristics, our focus is on the first three: complexity, compatibility, and trialability. Note that the last two characteristics remain unaddressed in our survey, as our approach is still preliminary and has not yet been deployed in practice.

The questionnaire on MRs, available online [17], includes, for each MR, a form showing the generated MR (e.g., see Table 1) and three assessment statements (S1, S2, and S3) described in Table 3. These forms are grouped by application, and for each application, the questionnaire invites practitioners to provide open feedback. Statement S1 in Table 3 is about complexity; it assesses the degree to which a derived MR is understandable. Statement S2 relates to compatibility; it assesses the degree to which a derived MR aligns with practitioners' perceptions as a property to be considered in testing the SUT. Statement S3 concerns triability; it examines how helpful a derived MR is as an instrument for defining test cases (i.e., input and expected output pairs).

The questionnaire on EMRs, available online [17], first introduces SMRL by explaining its constructs, providing three examples of EMRs written in SMRL, and a link to the SMRL paper [5]. The questionnaire then, for each MR, provides

[4] https://lmsys.org/blog/2023-12-07-leaderboard/

Table 3. Statements (S1, S2, and S3) for collecting feedback on MRs.

S1 The MR is easy to understand. ☐ strongly agree ☐ agree ☐ neutral ☐ disagree ☐ strongly disagree
S2 The MR captures a property that needs to be considered when testing the application. ☐ strongly agree ☐ agree ☐ neutral ☐ disagree ☐ strongly disagree
S3 The MR helps identify the expected outputs for given inputs. ☐ strongly agree ☐ agree ☐ neutral ☐ disagree ☐ strongly disagree

a form with an MR (e.g., see Table 1), an EMR (e.g., see Fig. 4), and three statements ($S1^E$, $S2^E$, and $S3^E$, listed in Table 4). These forms are grouped by each SISW application. For each application group, the questionnaire includes an open question to collect additional feedback. Statement $S1^E$ in Table 4 is about complexity, assessing the extent to which an EMR is understandable. Statement $S2^E$ relates to compatibility, evaluating the degree to which an EMR aligns with its corresponding MR. Statement $S3^E$ concerns triability, examining the feasibility of implementing the functions invoked in an EMR.

Our approach derived 50 MRs and their corresponding EMRs from the four documents. Of these MRs, SISW independently assigned two practitioners to evaluate the 14 MRs derived for the aero-acoustics application, while one was assigned to 36 MRs for the remaining three applications, totaling 64 MRs analyzed. For the EMRs, one practitioner was assigned to all the applications. Note that these assignments were independently decided by SISW.

EXP2 - Web application. EXP2 evaluates the correctness of the EMRs generated by our approach. Recall that EXP1 studies EMRs that are incomplete with regard to invoking the SUTs' APIs. Therefore, EXP2 employs a different application, for which we have its API specifications. EXP2 uses the experiment dataset provided by the prior work on MST-wi [5], containing both the APIs enabling the testing of a Web application (hereafter, SUT) and MRs that are designed to detect security vulnerabilities. Specifically, in EXP2, we randomly selected ten MRs among the ones published online *after the cutoff date* of GPT-3.5 (in EXP2, we relied on GPT-3.5 for this reason). Given the ten MRs and the SUT's APIs, we applied our approach to produce ten corresponding EMRs.

Table 4. Statements ($S1^E$, $S2^E$, and $S3^E$) for collecting feedback on EMRs.

$S1^E$ The EMR is easy to understand. ☐ strongly agree ☐ agree ☐ neutral ☐ disagree ☐ strongly disagree
$S2^E$ The EMR is consistent with the corresponding MR. ☐ strongly agree ☐ agree ☐ neutral ☐ disagree ☐ strongly disagree
$S3^E$ It is feasible to implement all the functions used in the EMR. ☐ strongly agree ☐ agree ☐ neutral ☐ disagree ☐ strongly disagree

Table 5. Annotation labels used in EXP2.

Label	Type	Description
CLC	Simple	The statement contains a correct language construct.
C	Complex	The statement is correct.
AI	Complex	The statement does not perform exactly what the original MR does, but it is a valid alternative implementation.
WLC	Simple	The statement misuses a language construct.
WS	Complex	The statement is wrong; it misses some required actions, performs the wrong operation, and the explanation does not reflect what is requested in the MR text.
WI	Complex	The statement does not correctly implement what is suggested in the explanation.
IE	Complex	The statement invokes an API function invented by ChatGPT, even if an adequate one exists.
INE	Complex	The statement invokes an API function invented by ChatGPT because an adequate one does not exist.
ITE	Complex	The statement invokes an API function invented by ChatGPT but delegates too much logic to it.
ES	Complex	The statement invokes an existing and appropriate API function but swaps parameters.
ENO	Complex	The statement invokes an existing and appropriate API function but not in an object-oriented manner.
WAU	Complex	The statement misuses valid APIs.
MISS	Complex	The statement misses an instruction required to implement what is reported in the explanation.

To assess the correctness of the ten generated EMRs, we inspected and annotated each line of the EMRs with one of the 13 labels described in Table 5. In the annotation process, we considered both the EMRs and their explanations (i.e., comments on the EMRs) generated by our approach. Note that each EMR statement is associated with an explanation in natural language produced by ChatGPT. Our labels distinguish between *complex* and *simple* statements. Simple statements are the ones including only one language construct (e.g., `IMPLIES`, `NOT`, `OR`). Complex statements include at least one method invocation. Further, our labels are classified into two groups: *correct* and *incorrect*. The former contains three labels (i.e., C, CLC, and AI in Table 5) indicating, respectively, whether a statement in an EMR implements part of the corresponding MR exactly, uses SMRL constructs correctly, or implements valid alternatives. The latter includes the remaining ten labels, each indicating a different type of issue, such as "incorrect use of SMRL constructs" and "misuse of the SUT's APIs". For example, in Fig. 4, the statement `if (!isSearchAction(searchAction)) continue;` on line 3 is annotated with the INE label, indicating that ChatGPT invented `isSearchAction()` and the EMR invokes it because an adequate API

of the SUT was not provided to ChatGPT when generating the EMR from the MR described in Table 1.

4 Experiment results

EXP1. Table 6 shows the results of EXP1, focusing on the combined number of responses (i.e., Likert ratings) for each statement across the 64 MRs reviewed by the three practitioners at SISW. From the results of statement S1, we found that in 49 out of the 64 MRs (77%), the practitioners were able to understand the proposed MRs and expressed positive feedback, i.e., "strongly agree" or "agree". In 12 MRs, the practitioners were neutral, and in the remaining 3 MRs, they disagreed with statement S1. Regarding statement S2, the practitioners agreed that the MRs capture properties that need to be considered when testing the SUT for 41 out of the 64 MRs (64%). In 10 MRs, the practitioners were neutral; in the remaining 13 MRs, the practitioners disagreed or strongly disagreed with statement S2. In contrast to these results, which are more positive, the responses to statement S3 showed mixed ratings. This statement indicates whether the presented MR helps identify the expected outputs for given inputs. In 18 MRs, the practitioners agreed; in 19 MRs, they were neutral; and in the remaining 27, they disagreed or strongly disagreed.

Regarding the non-positive responses to statement S3, we identified possible reasons from the practitioners' qualitative feedback. In particular, one practitioner, who provided 11 ratings as "disagree" and one as "strongly disagree" to the statement, stated that while most of the MRs are correct and useful for reminding someone what to test, they are too generic to aid in deriving appropriate testing. This aligns with feedback from another practitioner at SISW who, despite not assessing the MRs, collected and reviewed the responses to the MRs questionnaire. The practitioner provided a possible reason for some of the MRs being very generic, stating that it could be attributed to the nature of the documents provided by SISW.

Regarding the survey results for the 50 EMRs in EXP1, one practitioner provided consistent responses across all EMRs, stating they perceived that the EMRs *were clean and understandable* and *consistent with their corresponding MRs*. However, the practitioner deemed that not all functions used in the EMRs were feasible to be implemented. Regarding the last opinion, the practitioner noted that the functions in the EMRs, generated by ChatGPT, were specific to each EMR. Therefore, there is little opportunity for reusing these functions across different EMRs, meaning the effort required for implementation would be

Table 6. Responses to the MRs survey, for each statement (S1, S2, S3) in Table 3.

	strongly agree	agree	neutral	disagree	strongly disagree
S1	3	46	12	3	0
S2	0	41	10	12	1
S3	0	18	19	24	3

significant. In addition, the specific constraints of the testing environment for their applications may pose challenges in using the exposed APIs to implement functions in the EMRs. Even though EXP1 generated incomplete EMRs with regard to using the SUT's APIs, as discussed above, this finding suggests that the characteristics of the SUT, such as its APIs, should be considered when further elaborating our approach to deriving EMRs.

The results for EXP1 indicate that our approach generates MRs and EMRs that practitioners deem to be understandable and relevant for testing.

EXP2. In EXP2, we annotated each line of the ten EMRs, which were converted from the corresponding ten MRs studied in prior work [5]. These EMRs have a minimum of 10 statements, a mean of 13.6 statements, and a maximum of 20 statements, totaling 136 statements.

Table 7 shows the results of EXP2 annotations using the labels defined in Table 5. Out of the 136 statements in the EMR code, the majority (107 statements, 78.6%) were classified as correctly converted. Specifically, 52 of the 107 statements are complex, and correctly and exactly implement part of the corresponding MRs, as shown by column C in Table 7. 54 of the 107 statements are simple, and correctly use SMRL constructs (column CLC). The remaining complex statement is valid and correct with respect to the corresponding MR (column AI). As for the 30 statements that were deemed incorrect, they can be further categorized as follows:

- 16 statements incorrectly use APIs (labeled with IE, INE, ITE, ES, and ENO). For example, five ITE cases introduce the method `isAuthorized` to check if a user is authorized to perform an action, instead of verifying that the outputs obtained by the source and the follow-up inputs differ.
- 10 statements incorrectly use SMRL constructs (column WLC). They are all related to the LLM "forgetting" how to use the construct `IMPLIES`; indeed, instead of separating the right-hand side of the implication with a comma (see end of Line 9 in Fig. 4), it uses an `&`. They can be easily fixed manually.
- Three statements are wrong (column WS); for example, the generated MR checks if the returned page is an error page instead of the opposite.
- Three statements miss implementing what is described in the corresponding explanations (column MISS); specifically, they do not compare the outputs of the source and the follow-up inputs. Further, these three statements had been annotated also with another label because, in addition to miss some in-

Table 7. Distribution of annotation labels (from Table 5) for the 10 EMRs obtained from EXP2.

C	CLC	AI	WS	WI	IE	INE	ITE	ES	ENO	WAU	WLC	MISS
52	54	1	3	0	3	1	9	1	2	0	10	3
38.2%	39.7%	0.7%	2.2%	0.0%	0.7%	0.0%	6.6%	0.7%	1.5%	0.0%	7.4%	2.2%

structions, they contained an incorrect implementation (i.e., ENO and WS), which is the reason why our labels sum up to 139 instead of 136.

The results for EXP2 indicate that the generated EMRs are largely correct with respect to their MRs.

5 Threats to Validity and Research Opportunities

Even though the experiment results are promising, we recognize that our work is in its preliminary stages, and there are some potential threats to its validity. In this section, we discuss these concerns, focusing on those that lead to challenging yet important directions for future research.

Prompt engineering. Our prompts are designed by decomposing the processes of deriving MRs from requirements and transforming MRs into EMRs into smaller, simpler steps. Despite these decomposed steps aligning with those that engineers would conduct manually, when using LLMs there might be better sequences of prompts that could improve the accuracy and relevance of the automatically generated MRs and EMRs. Further research is necessary to refine and compare the prompts, ensuring that they efficiently guide LLMs in producing effective MRs and EMRs.

Assessing MRs and EMRs. We evaluated the MRs and EMRs generated by our approach using the questionnaire-base survey and annotation studies, which rely on the quality of human assessments. Hence, such assessments are naturally biased by the participating personnel in the study. To minimize any threat to construct validity (i.e., researcher expectations affecting human subjects' assessment), our industrial partner, SISW, invited four practitioners with different backgrounds for the questionnaire-based survey study. For the annotation study, the second author, who is familiar with SMRL, annotated the EMRs to ensure that the annotations are accurate. Although our choice may introduce a construct validity threat, the availability of experimental data enables other researchers to further investigate our findings.

As part of our future work, to better address face validity (i.e., the selection of appropriate reflective indicators), we aim to leverage recent studies on the assessment of code generation models [8] and select additional quantitative metrics to measure the degree to which MRs and EMRs are accurate and relevant to the SUT. Such metrics are desirable not only for objective assessment but also for automating the MT process; indeed, such metrics might be used as reward to improve LLMs results.

Human expertise. In our work, we leveraged human expertise to select technical documents and assess the outputs, i.e., MRs and EMRs. Since our approach does not involve humans-in-the-loop (e.g., the textual MRs generated by the LLM for EXP1 are not rewritten by humans before generating EMRs), the quality of outputs highly depends on the quality of inputs. To mitigate any bias originated from the input documents (construct validity), SISW selected four technical documents independently from the authors who developed the approach.

In the future, even if we aim to automate the MT process, incorporating humans in the loop could be used to refine and validate the inputs and the intermediate outputs, to efficiently and effectively guide our approach to produce more accurate and relevant MRs and EMRs. Therefore, future research should explore methods to effectively incorporate human expertise into an AI-based MT process, balancing automation with human interventions.

External validity. To mitigate generalizability threats, for EXP1, we considered specifications of industrial modelling and simulation software for four different applications domains (marine design, wind turbine, air-craft propulsion, and aero-acoustics). Although such software may differ from other types of software systems (e.g., content management software), it is of high complexity and adopted in many industrial sectors including automotive and space. Further, for EXP2, we focused on MRs that demonstrated effective for content management (e.g., Joomla [11]) and Web-based automation software (e.g., Jenkins [7]), thus complementing the choice made for EXP1.

6 Related Work

Many MT approaches have been developed for testing various SUTs, such as search engines [26], Web applications [5], and embedded systems [1], accounting for the requirements specific to these SUTs. However, these approaches heavily rely on manual efforts to elicit MRs, inherently limiting the applicability of MT. To reduce the cost of defining MRs, researchers have proposed relying on meta-heuristic search [24, 23] and genetic programming [2] to automatically derive MRs from execution data. Although promising, such approaches require several executions of the SUT, which makes them suitable for small programs, but may not scale for large software where executions may take several minutes and a lot of outputs are produced.

Recently, a few studies [12, 25] have proposed the use of LLMs to automatically derive MRs from the knowledge base of LLMs. However, these studies are limited to deriving MRs for SUTs already known in the adopted LLMs. In other words, there are no studies demonstrating the applicability of LLMs to derive MRs that account for the requirements specific to an SUT that was unseen during the LLM training phases. Considering that, in industrial contexts, software testing activities often target new products implementing requirements different from those implemented by existing systems, LLM-based approaches for the automated generation of MRs should be capable of handling unseen requirements. This capability is what we presented in this paper.

Additionally, we note that, to the best of our knowledge, there is no work that automates the conversion of MRs into an executable form, which is required to fully automate the MT process. Chaleshtari et al. [5] proposed the adoption of a DSL to automate the execution of MRs; however, DSL-based MRs may have limited readability, contrary to MRs in natural language.

In summary, there is no solution in the literature that aims at fully automating the MT process, involving the derivation of MRs from SUT-specific

requirements and the generation of EMRs, for the automated determination of test results in MT. An alternative is the automated generation of test cases—including oracles—using LLMs, which is under active development [22]. However, since LLM-generated code may still require human validation, we believe that validating EMRs is more cost-effective than validating test cases. Indeed, a single validated EMR enables exercising the SUT with multiple inputs, which might otherwise be exercised by different test cases, each needing validation.

7 Conclusion and Future work

In this paper, we have introduced our approach for automatically deriving EMRs from requirements using LLMs. In our preliminary experiments, we applied our approach to four applications of our industry partner (SISW) and conducted a questionnaire-based survey study to collect opinions from SISW practitioners. In addition, we evaluated the correctness of the EMRs generated by our approach, by applying it to a Web application. The feedback from practitioners at SISW confirmed that our work is a promising direction for automating the derivation of MRs and EMRs from requirements. Furthermore, our annotation results indicate that our approach has strong potential for correctly generating EMRs from MRs. However, our results also reveal that the derived MRs need to be more specific to the SUTs, and the derived EMRs could be further improved with regard to the use of DSL constructs and APIs, which necessitate further research.

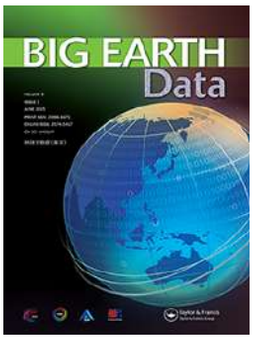
In our future work, we plan to explore several key areas related to our current research. A primary focus will be on automating prompt engineering to facilitate the generation of prompts for deriving MRs from requirements and their subsequent conversion into EMRs. Recognizing the significance of input quality, particularly in requirements, another important area will involve developing methods to assist engineers in providing high-quality inputs. Integrating human expertise, or "human in the loop", will also be an important aspect of our research, aiming to further enhance the accuracy and relevance of the generated MRs and EMRs.

Data availability. Our experiment package with prompts, MRs, EMRs, and questionnaires is available online [17].

References

1. Ayerdi, J., Segura, S., Arrieta, A., Sagardui, G., Arratibel, M.: QoS-aware metamorphic testing: An elevation case study. In: ISSRE. pp. 104–114 (2020)
2. Ayerdi, J., Terragni, V., Arrieta, A., Tonella, P., Sagardui, G., Arratibel, M.: Generating metamorphic relations for cyber-physical systems with genetic programming: An industrial case study. In: ESEC/FSE. pp. 1264–1274 (2021)
3. Barr, E.T., Harman, M., McMinn, P., Shahbaz, M., Yoo, S.: The oracle problem in software testing: A survey. IEEE TSE **41**(5), 507–525 (2015)
4. Bériot, H., Gabard, G., Hamiche, K., Williamschen, M.: High-order unstructured methods for computational aero-acoustics. Progress in simulation, control and reduction of ventilation noise, Publisher: (2015)

5. Chaleshtari, N.B., Pastore, F., Goknil, A., Briand, L.C.: Metamorphic testing for web system security. IEEE TSE **49**(6), 3430–3471 (2023)
6. Chen, T.Y., Kuo, F.C., Liu, H., Poon, P.L., Towey, D., Tse, T.H., Zhou, Z.Q.: Metamorphic testing: A review of challanges and opportunities. ACM Computing Surveys **51**(1) (2018)
7. Eclipse Foundation: Jenkins ci/cd server. https://jenkins.io/ (2018)
8. Evtikhiev, M., Bogomolov, E., Sokolov, Y., Bryksin, T.: Out of the bleu: How should we assess quality of the code generation models? J. Syst. Softw. **203**(C) (sep 2023)
9. Fioraldi, A., Maier, D.C., Eißfeldt, H., Heuse, M.: AFL++ : Combining incremental steps of fuzzing research. In: WOOT. pp. 1–12 (2020)
10. Fraser, G., Arcuri, A.: EvoSuite: automatic test suite generation for object-oriented software. In: ESEC/FSE. pp. 416–419 (2011)
11. Joomla: Joomla, https://www.joomla.org/ (2018)
12. Luu, Q.H., Liu, H., Chen, T.Y.: Can ChatGPT advance software testing intelligence? An experience report on metamorphic testing (2023)
13. Mai, P.X., Pastore, F., Goknil, A., Briand, L.: Metamorphic security testing for web systems. In: ICST. pp. 186–197 (2020)
14. Open AI: Chatgpt. https://openai.com/blog/chatgpt
15. OpenAI: GPT-4 technical report. CoRR **abs/2303.08774** (2023)
16. Rogers, E.M., Singhal, A., Quinlan, M.M.: Diffusion of innovations. In: An integrated approach to communication theory and research, pp. 432–448. Routledge (2014)
17. Shin, S.Y., Pastore, F., Bianculli, D., Baicoianu, A.: Replication package. https://doi.org/10.6084/m9.figshare.25046276.v1
18. Siemens: Improving wind turbine gearbox simulation. https://resources.sw.siemens.com/en-US/white-paper-full-scale-cfd (2021), accessed: 2024
19. Siemens: Full-scale simulation for marine design. https://resources.sw.siemens.com/en-US/white-paper-full-scale-cfd (2023), accessed: 2024
20. Siemens: Leveraging preliminary design and simulation to drive next-generation propulsion systems. https://view.highspot.com/viewer/65a7c309c5926b7b3e0192c6 (2023), accessed: 2024
21. Song, Y., Wang, T., Cai, P., Mondal, S.K., Sahoo, J.P.: A comprehensive survey of few-shot learning: Evolution, applications, challenges, and opportunities. ACM Computing Surveys **55**(13s) (jul 2023)
22. Wang, J., Huang, Y., Chen, C., Liu, Z., Wang, S., Wang, Q.: Software testing with large language models: Survey, landscape, and vision (2024)
23. Zacchia Lun, Y., D'Innocenzo, A., Smarra, F., Malavolta, I., Domenica Di Benedetto, M.: State of the art of cyber-physical systems security: An automatic control perspective. JSS **149**, 174–216 (2019)
24. Zhang, J., Chen, J., Hao, D., Xiong, Y., Xie, B., Zhang, L., Mei, H.: Search-based inference of polynomial metamorphic relations. In: ASE. pp. 701–712 (2014)
25. Zhang, Y., Towey, D., Pike, M.: Automated metamorphic-relation generation with ChatGPT: An experience report. In: COMPSAC. pp. 1780–1785 (2023)
26. Zhou, Z.Q., Xiang, S., Chen, T.Y.: Metamorphic Testing for Software Quality Assessment: A Study of Search Engines. IEEE TSE **42**(3), 260–280 (2016)



Big Earth Data

ISSN: 2096-4471 (Print) 2574-5417 (Online) Journal homepage: www.tandfonline.com/journals/tbed20

DACIA5: a Sentinel-1 and Sentinel-2 dataset for agricultural crop identification applications

A. Băicoianu, I. C. Plajer, M. Debu, M. Ștefan, M. Ivanovici, C. Florea, A. Cațaron, R. M. Coliban, Ș. Popa, Ș. Oprișescu, A. Racovițeanu, Gh. Olteanu, K. Marandskiy, A. Ghinea, A. Kazak, L. Majercsik, A. Manea & L. Dogar

To cite this article: A. Băicoianu, I. C. Plajer, M. Debu, M. Ștefan, M. Ivanovici, C. Florea, A. Cațaron, R. M. Coliban, Ș. Popa, Ș. Oprișescu, A. Racovițeanu, Gh. Olteanu, K. Marandskiy, A. Ghinea, A. Kazak, L. Majercsik, A. Manea & L. Dogar (09 Jun 2025): DACIA5: a Sentinel-1 and Sentinel-2 dataset for agricultural crop identification applications, Big Earth Data, DOI: 10.1080/20964471.2025.2512685

To link to this article: https://doi.org/10.1080/20964471.2025.2512685

BIG EARTH DATA
https://doi.org/10.1080/20964471.2025.2512685

DATA NOTE

OPEN ACCESS Check for updates

DACIA5: a Sentinel-1 and Sentinel-2 dataset for agricultural crop identification applications

A. Băicoianu[a], I. C. Plajer[a], M. Debu[a], M. Ștefan[b], M. Ivanovici[a], C. Florea[a,c], A. Cațaron[a], R. M. Coliban[a], Ș. Popa[a], Ș. Oprișescu[a], A. Racovițeanu[a,c], Gh. Olteanu[a], K. Marandskiy[a], A. Ghinea[b], A. Kazak[a,d], L. Majercsik[a], A. Manea[a] and L. Dogar[a]

[a]Department of Mathematics and Informatics, Transilvania University, Brașov, Romania; [b]National Institute of Research and Development for Potato and Sugar Beet, Brașov, Romania; [c]Department of Image Processing and Analysis Laboratory, National University of Science and Technology Politehnica Bucharest, Bucharest, Romania; [d]Department of Telecommunications and Electronic Systems, Technical University of Moldova, Chișinău, Republic of Moldova

ABSTRACT

Artificial intelligence and data analysis are essential in smart agriculture for enhancing crop productivity and food security. However, progress in this field is often limited by the lack of specialized, error-free labeled datasets. This paper introduces DACIA5, a multispectral image dataset for agricultural crop identification, complemented with Sentinel-1 radar data. The dataset consists of 172 Sentinel-2 multispectral images (800×450 pixels) and 159 Sentinel-1 radar images, collected over Brașov, Romania, from 2020 to 2024, with precise, in-situ verified labels. Additionally, 6,454 Sentinel-2 and 5,995 Sentinel-1 rectangular patches (32×32 pixels) were extracted, exceeding 6 million pixels in total. The cropland parcels considered in our dataset are used for research and are owned and cultivated by the National Institute of Research and Development for Potato and Sugar Beet, ensuring error-free labeling. The labels in our dataset provide detailed information about crop types, offering insights into crop distribution, growth stages, and phenological events. Furthermore, we present a comprehensive dataset analysis and two key use cases: crop identification based on a "past vs. present" approach and early crop identification during the agricultural season.

ARTICLE HISTORY
Received 24 February 2025
Accepted 18 May 2025

KEYWORDS
Sentinel-2 data; Sentinel-1 data; smart agriculture; artificial intelligence; crop identification; early crop identification

1. Introduction

With the growth of the world population (Ritchie & Rodes-Guiaro, 2024), the threat on food security induced by climate changes, conflicts, and the need for biodiversity protection, it becomes increasingly important to optimize the usage of resources for agricultural purposes (Anderson et al., 2020; Frona et al., 2019; Maja & Ayano, 2021). Precision

CONTACT A. Băicoianu a.baicoianu@unitbv.ro Department of Mathematics and Informatics, Transilvania University of Brașov, 500091, Romania

Supplemental data for this article can be accessed online at https://doi.org/10.1080/20964471.2025.2512685

agriculture, supported by the latest technological developments such as artificial intelligence (AI), offers a promising solution to these challenges (Ivanovici, Baicoianu, et al., 2024b; Marandskiy & Ivanovici, 2024). In this context, intense research on using AI and smart systems for crop monitoring has been done in recent years (Akkem et al., 2023). However, one of the main problems in using AI models in general, and machine learning (ML) and deep learning (DL) in particular, is the large amount of training data required for accurate detection and classification (Benami et al., 2021). In agreement with prior work (Alzubaidi et al., 2023), we observed that freely available labeled data is often insufficient. Moreover, in agricultural data sets, many labels rely on self-reported information from farmers, which sometimes is not accurate or misleading. This can induce serious biases in the resulting learning-based models (Cabrera et al., 2014). While ML and DL models can cope with erroneous labels and data, this robustness is limited (Song et al., 2022).

In this direction, while several datasets are available for land cover and slightly towards agricultural crop monitoring, they frequently exhibit limitations that prevent them from adequately addressing the specific requirements of precision agriculture.

One notable dataset is BigEarthNet (Sumbul et al., 2019), a large-scale benchmark archive comprising Sentinel-2 image patches spanning 10 European countries. Although comprehensive, the BigEarthNet dataset is primarily designed for multi-label remote sensing image classification, with a particular focus on land cover classes. Since it contains classes such as “agricultural land” and “pasture”, it lacks detailed crop type annotations, verified through in-situ validation, which represents a significant limitation for precise agricultural applications. Similarly, EuroSAT (Helber et al., 2019) provides a dataset based on Sentinel-2 imagery with a focus on land use and land cover classification. Although the dataset includes agricultural areas among its categories, it lacks the specific crop type information and multi-temporal data necessary for monitoring crop growth stages and phenological events. The SEN12MS dataset (Schmitt et al., 2019) provides also a selection of georeferenced multi-spectral Sentinel-1/2 imagery, suitable for use in deep learning and data fusion. While the SEN12MS dataset is useful for a range of remote sensing applications, it does not focus on agricultural parcels with accurate crop labels over multiple years. This limits its applicability for detailed crop monitoring tasks. Additionally, the CropHarvest dataset (Tseng et al., 2021) is a global repository for crop type classification, aggregating data from multiple sources. However, CropHarvest frequently relies on crowd-sourced labels and lacks the desired level of accuracy for precise AI modeling in agriculture. Additionally, the dataset may exhibit inconsistencies in temporal coverage and spatial resolution, which are essential for analyzing crop dynamics over time. A recent pixel-based dataset, TimeSen2Crop (Weikmann et al., 2021), includes over a million labeled multispectral Sentinel-2 pixels collected across Austria over two consecutive years, covering 16 different crop types. The labels were derived from farmers’ declarations. A notable feature of this dataset is the inclusion of reports on snow, shadows, and cloud coverage for each labeled unit.

The exhibited limitations, including the absence of accurate, in-situ validated crop type labels, insufficient multi-year data over the same parcels, the reliance on less precise crowd-sourced labels, and lack of regional specificity, underscore the need for a new dataset that effectively addresses the specific requirements of smart agriculture.

In order to help the international community and enrich the available data, we generated a new dataset for crop monitoring, certification, and identification/detection,

which has the important advantage of accurate labels, based on in-situ institutional validation. Moreover, the dataset was constructed on the same parcels over the interval of 5 years, thus including variability resulting from different climatic conditions. The construction and validation of the data set was done in collaboration with the National Institute of Research and Development for Potato and Sugar Beet from Brașov, Romania (INCDCSZ, n.d.). To enhance clarity and readability, the National Institute of Research and Development for Potato and Sugar Beet will be referred to as the *Potato Institute* throughout the remainder of this article.

2. Methods

The dataset presented in this study is based on data acquired by Sentinel-2 optical instruments (Drusch et al., 2012) as well as on Sentinel-1 radar data (Torres et al., 2012). This section provides a comprehensive description of the procedures and steps used in producing the dataset. It includes a detailed account of the experimental design and the computational processing involved. Specifically, we present the geographical and climate context, describe the experimental setting in relation to Sentinel-1 and Sentinel-2 data, and outline the process of dataset creation, starting from the engineering data, and detailing the tools utilized.

2.1. Geographical and climate setting

The land of the Potato Institute from Brasov, where the metadata for our dataset was collected, is part of the Bârsa premontane plain, which is a vast depression in Romania, surrounded by mountains. Bârsa is a part of the so-called "Potato country" in Romania. From a geographical perspective, Bârsa area extends from 45°27′ to 46°00′ north latitude and from 26°10′ to 26°13′ east longitude and has a surface of 2406 km^2 in extent. The altitude varies from 550 m at Bod village to 722 m at Zărnești city. This region experiences a climate transitional between Mediterranean and continental types. According to the Köppen classification (Beck et al., 2023; Peel et al., 2007) it falls into the climate region Dfb (cold, without dry season, warm summer). This climate is characterized by cold winters with significant snowfall and moderate precipitation throughout the year, as defined by the Köppen climate symbols and criteria. The region enjoys relatively warm springs, mild summers with temperatures suitable for potato cultivation, and long, sunny autumns extending into November. The humidity, indicated by an aridity index ranging from 34.8 to 40.8, supports favorable growing conditions for crops like potato and sugar beet. Bârsa has relatively low-to-medium annual precipitation (548–782 mm) due to its mountainous surroundings which absorb the fallen precipitation, with most rainfalls occurring in winter and drier periods in spring. Summer rainfalls are 250–300 mm. The region's average annual temperature of 7–8°C and summer temperatures of 15–17°C are suitable for potato cultivation, though recent droughts have increased the need for irrigation.

The soils within the Potato Institute belong to the mollic, hydromorphic, and weakly developed soil classes. It predominates the chernozimoid soil type, with a high humus content in the A horizon and a very large reserve in the first centimeters, moderately acidic reaction, good nitrogen supply, nitrification conditions negatively influenced by the acid reaction, medium insurance with mobile

forms of assimilable phosphorus and potassium, the degree of saturation in bases is high (approx. 70%), thus determining a good microbial activity. Under these conditions, the soil is favorable for potato cultivation, with a quality score of 81 out of 100. As noted in Teaci (1980), this score is based on an evaluation of the main climatic parameters (temperature, precipitation), edaphic factors (physical, chemical, and relief characteristics), and their ecological favorability for various agricultural crops. This type of information is highly relevant for the agricultural domain and experts. Combined with the publicly available dataset we provide researchers and agricultural professionals with supplementary information, to allow a more in-depth analysis and the evaluation of the impact of various parameters on both the quantity and quality of agricultural production.

From the flora point of view, the institute is located in an area where the spontaneous vegetation is represented by forest associations where *Quercus* and *Fagus* species predominate, mixed with shrubs (*Crataegus sp., Cornus sp*.) and herbaceous species, like grasses (*Lolium sp., Poa sp*.) and legumes (*Trifolium sp*.). Main crops include potato, sugar beet, cereals, legumes, and fodder plants. The flat terrain and soil texture support the full mechanization of these crops.

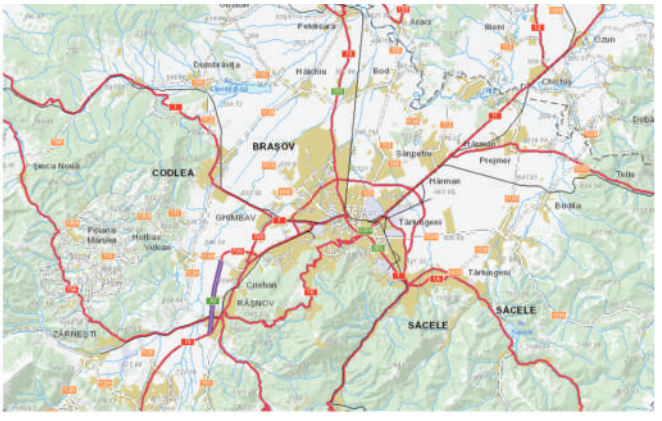
To provide a comprehensive view of the natural characteristics of the Brașov region, we have included a series of three thematic maps, each highlighting different aspects of the area, including height profile, soil types, and land cover, see Figure 1. For the creation of the height profile map, data were sourced from the official platform provided by the National Agency for Cadastre and Land Registration of Romania (https://geoportal.ancpi.ro/). The soil types map, depicted in Figure 1b, was generated utilizing information from (Liedekerke et al., 2006; Panagos, 2006; Panagos et al., 2022). The land cover map was produced using resources available on the Copernicus Land Monitoring Service platform (https://land.copernicus.eu/).

2.2. Dataset acquisition

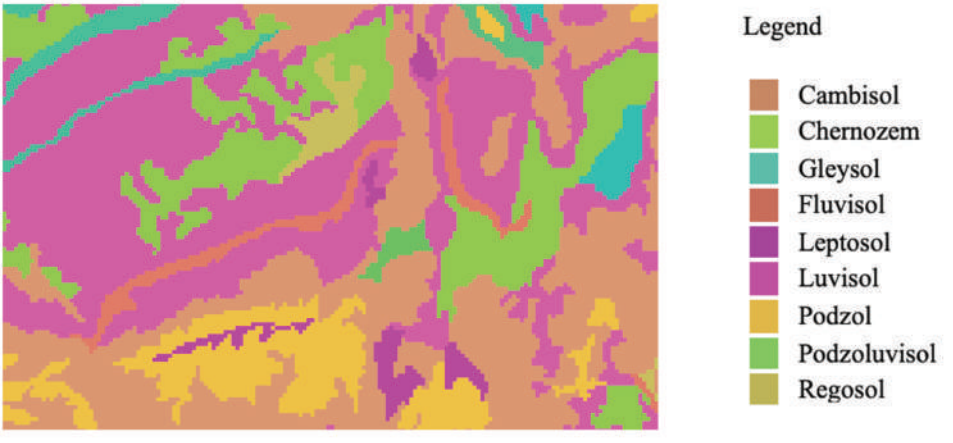
AI models, particularly in recent deep learning frameworks, require a lot of data (Borisov et al., 2024). In agriculture, a significant source of big data comes from remote sensing, especially through satellite imagery. More and more satellite missions offer free data for the international community. In this context, one of the most significant is the Copernicus Earth Observation Programme (Aschbacher, 2017). This is a European Union program using a constellation of satellites called "Sentinels" to gather extensive environmental data, which is freely available to users worldwide.

The Sentinel-2 satellites (Gascon et al., 2014), launched in 2015 and 2017, offer 13 spectral bands with spatial resolutions ranging from 10 m to 60 m (Gascon et al., 2017), enabling detailed analysis of vegetation, soil, and water. These satellites operate in a Sun-synchronous orbit, maintaining consistent sunlight angles for accurate, shadow-minimized imagery, with a 100-min orbit period at an altitude of 786 km and a 290 km swath width (n.d.).

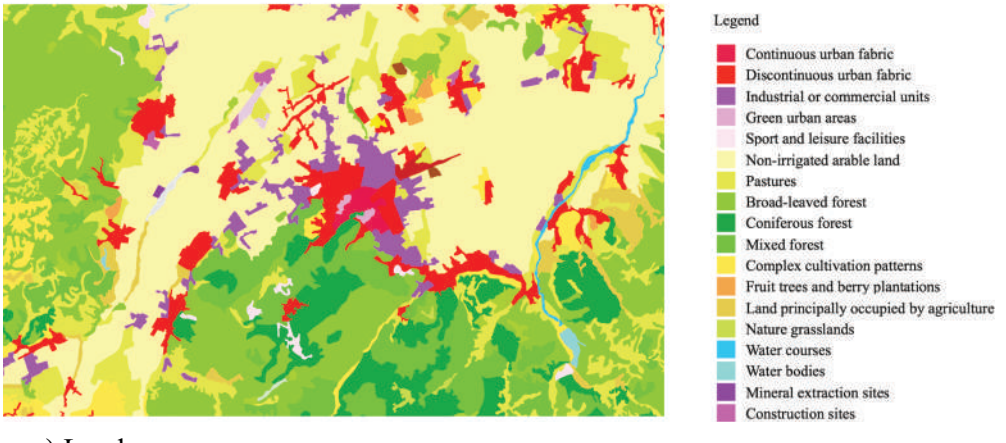
The dataset was obtained by downloading Sentinel-2 products from the Copernicus Browser platform (Copernicus Browser, 2024). The data is free and can be downloaded after creating an account. The platform provides data from all Copernicus missions. Users can view and download data specifically from the Sentinel-2 mission, at processing level



a) Height profile map



b) Soil types map



c) Land cover map

Figure 1. Illustrating maps with the characteristics of the Brașov area, including latitude and longitude, legend, and scale.

Table 1. Sentinel-2 bands with the central wavelengths (in nanometers) and their description.

Band	Spatial Resolution (m)	Central Wavelength (nm)	Description
B1	60	443	Aerosol
B2	10	490	Blue
B3	10	560	Green
B4	10	665	Red
B5	20	705	Vegetation Red Edge 1
B6	20	740	Vegetation Red Edge 2
B7	20	783	Vegetation Red Edge 3
B8	10	842	NIR
B8A	20	865	Narrow NIR
B9	60	945	Water Vapor
B10	60	1375	Cirrus Clouds
B11	20	1610	SWIR1
B12	20	2190	SWIR 2

2A. We downloaded the data covering the period from 2020 to 2024. The downloaded images are from the area of the Research and Development Institute of Transilvania University of Brașov (Research and Development Institute of Transilvania University of Brașov, 2024) located at latitude: 45.669410 and longitude: 25.549550. The downloaded products are all at processing level 2A. This processing level is applied to the raw level 1C data with atmospheric correction. During atmospheric correction, the band number 10 (wavelength 1373 nm) is removed (Sentinel-2 processing, n.d.).

In Table 1 we provide an overview of the spectral bands specific to Sentinel-2 images as presented in (Pour et al., 2023). It should be noted that Band 10 was excluded from the level 2A products during the process of atmospheric correction. Since our dataset was generated using level 2A products, the images comprise Bands 1–9 and Bands 11–12. As can be observed in Table 1, the original Sentinel-2 bands have different spatial resolutions. In order to uniformize the bands, they were all sampled to the same spatial resolution of 10×10 m, using the Sentinel Application Platform (SNAP) default Nearest Neighbour algorithm (SNAP – Resampling Methods, n.d).

The image selection process was carried out using the Copernicus Browser platform, which allows filtering Sentinel-2 satellite images based on the maximum cloud coverage parameter (Max. cloud coverage). To ensure a balance between data availability and quality, we set a threshold of 30% cloud coverage, meaning that only images with a cloud cover below this percentage were considered. It is important to note that this threshold applies to the entire satellite product, not exclusively to our area of interest. Regarding the use of the Scene Classification Layer, while the initial filtering was performed using the Max. cloud coverage parameter, an additional manual selection step was necessary. This involved visually inspecting the pre-selected images to ensure that the area of interest was free from cloud shadows and other obstructions that could affect the analysis. The downloaded products were visualized with the Sentinel Application

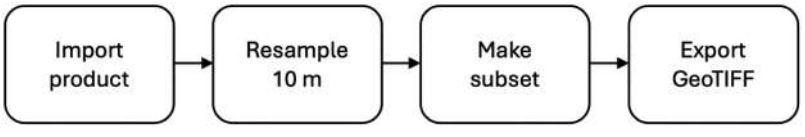


Figure 2. Preprocessing steps in Sentinel Application platform (SNAP) – European Space Agency (ESA).

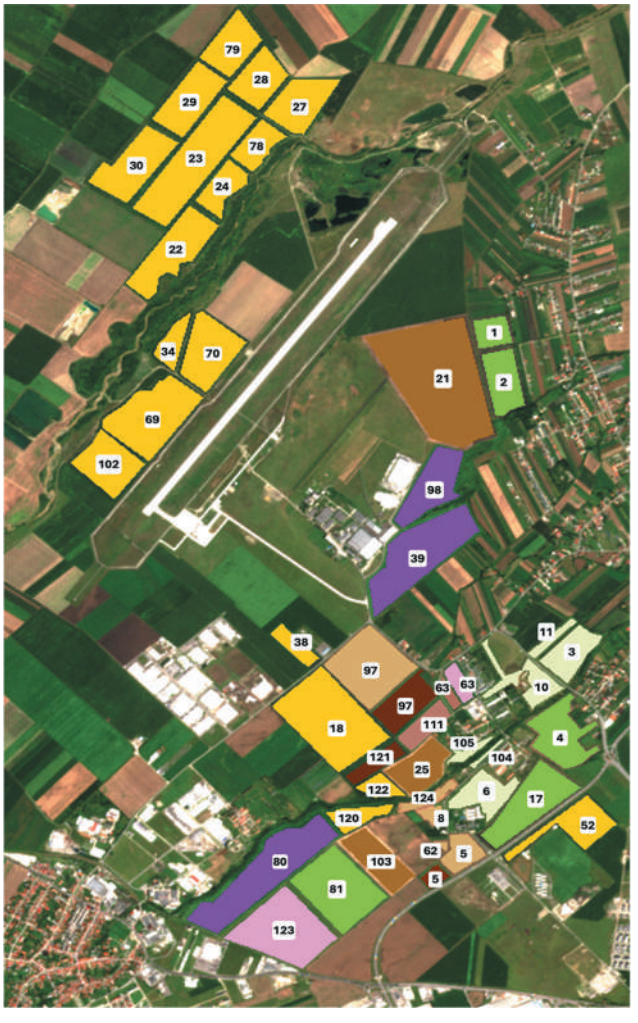


Figure 3. Example of the Potato Institute map of agricultural crops for 2023. The map presents the delimitation of each parcel. The parcels are identified by the corresponding parcel number and color. The colors were assigned using the color palette proposed by the US Department of agriculture (United States Department of Agriculture, 2024).

Platform (SNAP) (Sentinel Application Platform, n.d.) which is a free, open-source platform developed by the European Space Agency (ESA) (European Space Agency, n.d.). Using the ESA SNAP, the Sentinel-2 mission products were visualized, and the steps in Figure 2 were applied in order to obtain images that encompass the area of interest (AOI).

The subset of the initial product was realized using the coordinates inferred after the identification of the Research and Development Institute of Transilvania University of Brașov. The size of the images was chosen to encompass all parcels at the Potato Institute. The AOI was identified using maps from the Potato Institute. An example map can be seen in Figure 3. The maps were used to identify the parcels cultivated by the Potato Institute. Parcel marking was done manually using a software program we developed in MATLAB. The application provides the possibility to read GeoTIFF images from Sentinel-2. To

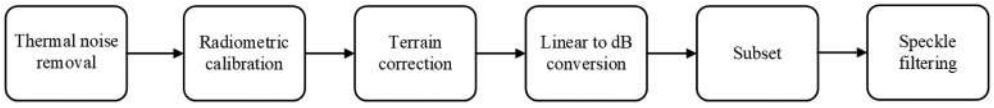


Figure 4. Processing steps for the Sentinel-1 data using the Sentinel-1 Toolbox.

visualize the image, the red, green, and blue (RGB) color channels were selected. In order to produce a color composite image to be displayed, the image values were scaled in the range [0, 255] and a brightness enhancement was performed.

To identify and save the parcel coordinates, an image was chosen where the parcel boundaries were clearly visible. Parcel delineation and coordinate extraction were done with *roipoly* MATLAB function on images acquired in summer, where the parcels can best be observed. Saving the parcel coordinates was done by clicking on the corners of the parcels in the RGB image and saving the coordinates of these points in .txt files. After saving the coordinates of all the parcels that exist on the Potato Institute's maps, masks were generated to cover each year's parcels. These masks were created based on the .txt files, which contained the necessary spatial information. The details regarding the number of these files have been provided in Table 3.

The Sentinel-1 mission, part of the European Space Agency's Copernicus programme, has been widely used in agricultural applications due to its all-weather, day-and-night Synthetic Aperture Radar (SAR) capabilities, and it operates in the C-band at 5.405 GHz. SAR uses the motion of the satellite platform to synthetically increase the aperture of the Radar, which allows high-resolution acquisition of images. SAR backscattering is affected by the structure and the water content of the soil surface and plants. In this dataset, we exclusively used Sentinel-1 images that are temporally close to corresponding Sentinel-2 images in our set. For each Sentinel-2 image, we selected a corresponding Sentinel-1 SAR image acquired within a three-day window, either up to 3 days before or after the Sentinel-2 acquisition date. We retrieved Sentinel-1 SAR images that are acquired in Interferometric Wide (IW) swath mode and included both VV (vertical transmit and vertical receive) and VH (vertical transmit and horizontal receive) polarizations with 10-m spatial resolution from Google Earth Engine's "COPERNICUS/S1_GRD" collection. These images have already been preprocessed using the Sentinel-1 Toolbox (n.d.). This includes detection, multi-looking, projection to ground range using an Earth ellipsoid model (e.g. WGS84), thermal noise removal, radiometric calibration, terrain correction (using SRTM 30 m or ASTER DEM for areas above 60° latitude) where images were also reprojected to a UTM coordinate system, and, finally, conversion of the terrain-corrected values to decibels ($10 \times \log_{10}(x)$). We further processed them after retrieval by applying a Lee Filter (Lee, 1980) to reduce speckle noise. Figure 4 illustrates the processing steps for the Sentinel-1 images.

2.3. Crop description and color scheme

A MATLAB script was used for the generation of masks based on the coordinates saved in .txt files. The files are saved in different directories for each year. Although the parcel

Table 2. The color palette for the masks. The first column lists the crop names, the second column provides the specific APIA identifier for each crop, the third column shows the labels assigned by us, and the final column indicates the crop color as defined by the United States Department of Agriculture.

Name crops	APIA code	Label	Color	Color code (RGB)
Common winter wheat	101	1		(165, 112, 0)
Common spring wheat	1010	2		(217, 181, 107)
Corn	108	3		(255, 211, 0)
Peas	151	4		(84, 255, 0)
Late potatoes	253	5		(112, 38, 1)
Other potato crop	254	6		(125, 46, 7)
Potatoes for seed	255	7		(180, 112, 91)
Potatoes for seed	2557	8		(189, 125, 106)
Sugar beets	3017	9		(167, 0, 228)
Temporal grassland	450	10		(233, 255, 191)
Alfalfa	9748	11		(255, 165, 226)
Permanent grassland	606	12		(233, 255, 191)
Corn silage	131	13		(255, 211, 0)
Soybean	2037	14		(38, 112, 0)
Alfalfa	9747	15		(255, 165, 226)
Winter rapeseed	202	16		(209, 255, 0)
Sunflower	123	17		(255, 255, 0)

coordinates remain largely the same, the crops change from year to year. The masks were refined based on the information received from the Potato Institute. Each year, all the parcels were analyzed and checked on the maps received in order to avoid positioning errors or mistakes in specifying a crop.

All the files used in mask generation follow a naming convention. Each name is of the form "*a_b*" where *a* represents the parcel number and *b* the unique code given to a crop by the Payments and Intervention Agency for Agriculture (APIA) (n.d.) in Romania (see Table 2).

For example, the file with the name "1_151.mat" contains the multispectral data from parcel 1 and the crop with the code 151 corresponding to green peas. For each year, we generated masks for each parcel, with labels corresponding to the specific crops of that year. The visual representation of the masks enables users to easily identify the crops that were cultivated on the parcels by the Potato Institute in the respective year by

Table 3. Overview of acquired Sentinel-1 and Sentinel-2 images from DACIA5 dataset.

Year	Number of images of Sentinel-1	Number of images of Sentinel-2	Number of text files	Number of crops	Number of patches of Sentinel-1	Number of patches Sentinel-2
2020	40	40	51	10	1520	1520
2021	32	32	53	12	1380	1380
2022	25	30	47	12	914	1121
2023	28	30	47	12	1289	1410
2024	34	40	48	11	892	1023

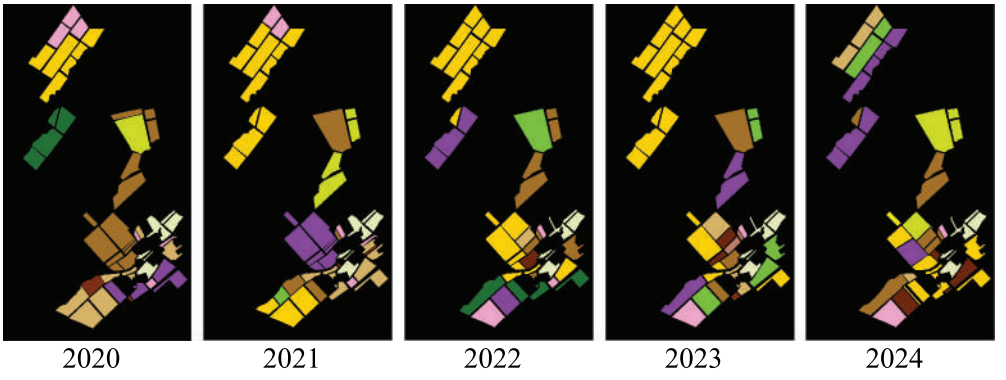


Figure 5. The masks with the parcels and the crops from 2020 to 2024, provided also as shapefiles in the DACIA5 dataset.

their specific colors. For the masks, we used the color palette proposed by the United States Department of Agriculture, which is detailed in Table 2. The colored masks allow us to observe the distribution and variety of the crops. The study of the masks also shows the crop rotation from year to year. The resulting masks are shown in Figure 5. The Potato Institute has dedicated parcels for research studies. These parcels are the same every year, but the crops are changed according to research needs. The APIA corresponding crop codes for the parcels from 2020 to 2024 are provided in the Appendix. It is important to mention that not all 17 crops are present every year.

In Table 3 we present the number of Sentinel-1 and Sentinel-2 images acquired in each year from 2020 to 2024, the number of text files containing the coordinates of the parcels, the number of crops in each year, as well as the number of patches generated from the Sentinel-1 and, respectively, the Sentinel-2 images. The number of text files differs in the different years, as some of the parcels were divided in some years into subparcels.

3. Data records

In this section, we provide detailed information about the data files, format (GeoTIFF), and other aspects that should ease the work with the data. The database contains all the available images from Sentinel-2 MSI and the corresponding Sentinel-1 SAR spanning over a time period of 5 years, from 2020 to 2024, covering the area of interest, as

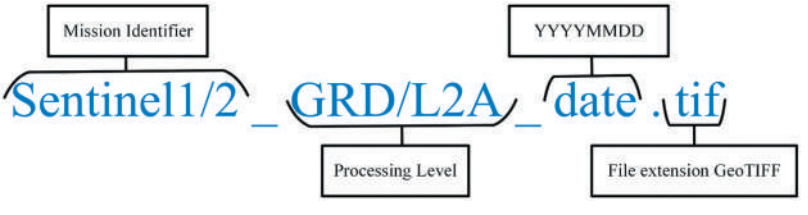


Figure 6. Image naming convention. YYYY is the four-digit year, MM is the two-digit month, and DD is the two-digit day.

described in Section 2.1. All the images respect a naming convention and are saved in GeoTIFF format. This GeoTIFF format can be read in MATLAB and Python. The naming convention is detailed in Figure 6. The Sentinel-2 images have dimensions of $800 \times 450 \times 12$, meaning each image consists of 800 pixels in height, 450 pixels in width, and 12 spectral bands, with a spatial resolution of 10×10 m. Similarly, Sentinel-1 images have dimensions of $800 \times 450 \times 2$, comprising 800 pixels in height, 450 pixels in width, and 2 bands, also with a spatial resolution of 10×10 m.

The Sentinel-1 data is stored in a folder named *Images_Sentinel1_2bands_GeoTIFF*, while folder *Images_Sentinel2_12bands_GeoTIFF* contains Sentinel-2 data. Within each of these folders, the acquired images are organized into subfolders corresponding to the year in which they were captured by the Sentinel-1 SAR sensors and the Sentinel-2 optical sensor, respectively. The folders containing these images have the names *Sentinel1_yyyy* and *Sentinel2_yyyy*, where *yyyy* represents the four-digit year. The directory *Images_Sentinel2_GeoTIFF* contains all the Sentinel-2 images as the previous Sentinel-2 folder, but these images have all the information provided by the Copernicus browser. Additional information is provided in Table 3, which displays the number of images from each year. The file naming convention is depicted in Figure 6.

In addition to the Sentinel-1 SAR data and the Sentinel-2 multispectral images, the database contains the ground truth of agricultural crops as RGB masks in PNG format and the masks with labels corresponding to each agricultural crop in both PNG and MAT formats. These are located in the *Masks_and_legend* directory. This directory also contains the legend for the masks in PDF format and the five subdirectories where the masks for each year are stored. The subdirectories are named *Masks_yyyy*, where *yyyy* represents the four characters for the corresponding year.

Using the aforementioned Sentinel-1 and Sentinel-2 imagery, we generated SAR and multispectral patches with dimensions of $32 \times 32 \times 2$, respectively, $32 \times 32 \times 12$,

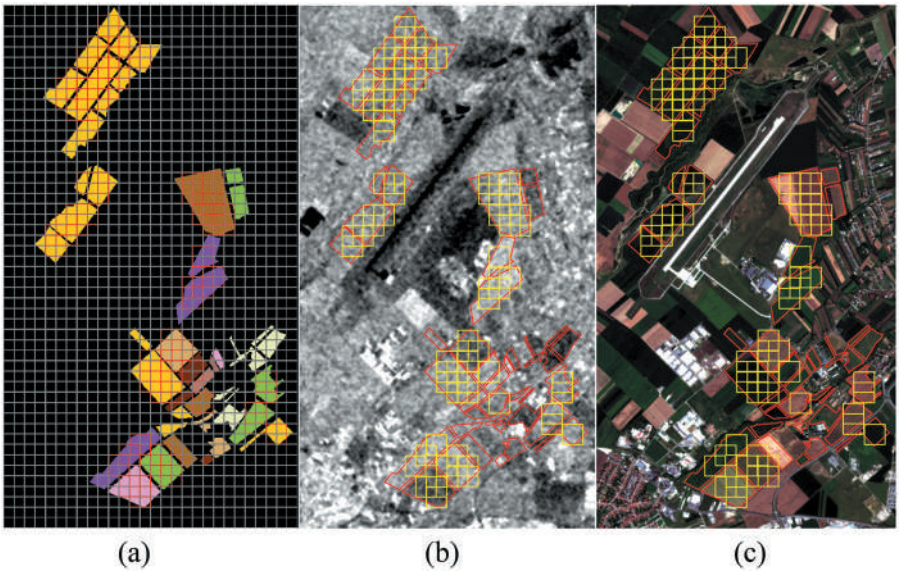


Figure 7. Selection grid for the patches.

which are systematically organized within the database in the *32×32_patches* folder. For automatic patch selection, the chosen rectangular areas of both the multispectral and SAR images were swept from the top-left corner to the bottom-right corner using a 32×32-pixel mask with a step of 16 pixels. Simultaneously, the label mask corresponding to each year was also traversed to count the number of pixels belonging to a specific crop within each identified patch. For each selected patch, a validity check was performed. A patch is considered valid if it contains at least 75% (768 out of 1024) pixels from a specific crop. The patches deemed valid were saved with a specific naming convention (which can be referenced). The position of the patches as a consequence of the traversal process can be observed in Figure 7(a) as well as the position of the resulting patches. In Figure 7(b) and (c), the overlay of patches on a Sentinel-1 image and a Sentinel-2 image, respectively, can be observed. This figure shows that the overlay is identical in both images, with each patch from Sentinel-1 corresponding to a patch from Sentinel-2, covering the same area.

After generating all the patches from the multispectral images, we examined their band selection RGB visualization, in order to filter the patches of interest for the training model, specifically those that contain visible crops (that have sprouted) and not bare soil. To generate the radar patches we used the same method as in case of multispectral patches. To filter them, we used the already examined RGB patches generated from multispectral images. An example of a radar and multispectral patch can be seen in Figure 8.

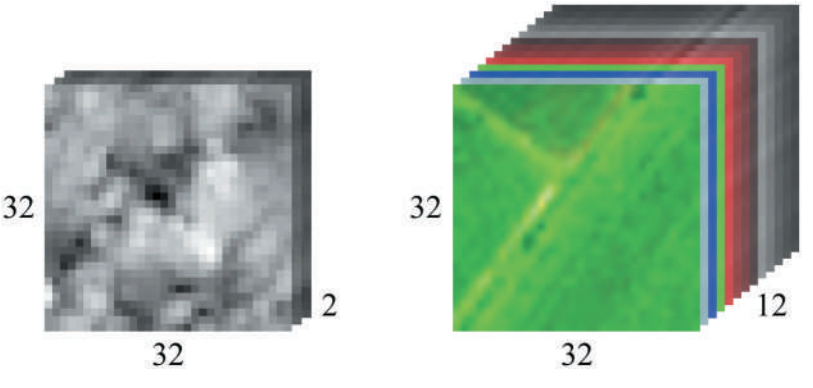


Figure 8. Visualization of a Sentinel-1 patch (left) and an RGB Sentinel-2 patch (right). The patches represent the same area on the ground.

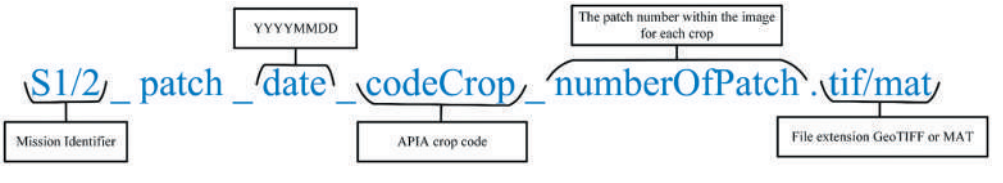


Figure 9. The naming convention for patches.

Each patch is named according to a structured convention, as exemplified in Figure 9. The practical use of these patches is demonstrated in Section 5, where they are employed to address two specific problems.

The *32×32_patches* directory is subdivided into two subdirectories: *32×32_SAR +MSI_patches* and *32×32_RGB_patches*, with the latter intended solely for visualization purposes. The first subdirectory contains radar and multispectral data utilized for addressing the two problems: *problem1* corresponds to the agricultural crop identification (ACI) with temporal generalization, while *problem2* focuses on the agricultural crop early identification (ACEI). For each problem, the data is further divided into training and test sets. The patches in these directories are georeferenced and are saved in both GEOTIFF and MAT formats, as indicated by the names of the subdirectories where they are stored: *sentinel1_patches_mat, sentinel1_patches_tiff, sentinel2_patches_mat,* and *sentinel2_-pacthes_tiff*. For the fused data scenario discussed in Section 5, we included a CSV file for training and testing, which maps the correspondence between Sentinel-1 patches and Sentinel-2 patches.

The directory *32×32_RGB_patches* contains the patches generated from the RGB masks, as a ground truth for the labels at pixel level. Based on these, we generated the patches from each image. Each patch contains an agricultural crop, which must occupy at least 75% of the pixels of the patch for it to be considered. For each year, there is a subdirectory where the patches from the masks are saved, named *patches_yyyy*, where *yyyy* represents the four characters corresponding to the respective year.

In our dataset, we also included the geographic information about the area and the crop field labels. *RoI_and_labels* folder has three sub-folders: *RoI* sub-folder includes files to locate the region of interest (*RoI*) as rectangle polygon, while *WGS84* and *UTM* sub-folders include polygons for crop field labels in geographic (WGS84) and projected coordinate system (UTM). They are further separated based on the year.

File names inside *RoI* and *WGS84* sub-folders which include "wgs84" represent the RoI and label polygons in the geographic coordinate reference system, specifically, World Geodetic System 1984 which is also known as EPSG:4326. This system represents Earth as a three-dimensional ellipsoid. File names that include "utm" in the RoI and UTM sub-folders represent the RoI and label polygons in a projected coordinate system, specifically the Universal Transverse Mercator (UTM), which provides a flat, two-dimensional representation of the Earth. We included both coordinate systems for convenience: WGS84 is more commonly used with Google Earth, Copernicus Browser, and similar applications, while the dataset itself is in the UTM coordinate system for more precise analysis. The primary files have the ".shp" suffix, which stores geospatial vector data. This type of file, known as a shapefile, contains the X and Y coordinates for the RoI and label polygons. Additional files with the suffixes ".shx", ".dbf", and ".prj" accompany the shapefile. The ".shx" file serves as an index for the shapefile, while the ".prj" file contains coordinate system and projection information for the RoI and labels. The ".dbf" file is a dBase database file that complements the shapefile by storing additional attribute data. For the RoI, the dBase file contains only the "NAME" field, while for labels, it includes the following fields: *crop_name, apia_code, label, rgb_r, rgb_g, rgb_b*, and *hex_color*. The *label* field holds non-zero positive integer values, *rgb_r, rgb_g*, and *rgb_b* store color values, and *hex_color* provides the HEX code equivalents of the RGB values. This color information can

be used to overlay crop field labels with specific colors for better visualization and inspection, consistent with the details provided in Table 2.

4. Technical validation

In the context of smart agriculture, where monitoring and automation are increasingly driven by machine learning models, the integrity and accuracy of data are critical. Machine learning algorithms rely heavily on large-size, high-quality data to make reliable predictions, optimize processes, and support decision-making. Any errors in labeling or inconsistencies in the dataset can lead to flawed models, deceiving accuracy, and sub-optimal outcomes, which is especially detrimental in agriculture, where precision is key to resource management, crop monitoring, and sustainability.

In Romania, APIA (https://apia.org.ro/) is responsible for the administration of agricultural subsidies and direct payments to Romanian farmers, thereby playing a pivotal role in the implementation of the European Union's Common Agricultural Policy (CAP) (ECA - European Court of Auditors, 2020), that is implemented with various tools, including the Single Area Payment Scheme (SAPS). This provides direct payments based on land size, along with other initiatives designed to advance environmental protection, rural development, and the modernization of agricultural practices. The agency has to ensure rigorous quality control measures that guarantee the effective and transparent utilization of funds.

The information provided by APIA ensures that our dataset is meticulously curated and free from labeling errors. As a national authority responsible for managing agricultural subsidies and interventions, APIA plays a crucial role in confirming and verifying the accuracy of information, ensuring that data used in our processes adheres to strict quality standards. The partnership with the Potato Institute has allowed us to create a clean, error-free dataset that meets the highest standards of data integrity. Through close collaboration with the Potato Institute, the dataset provided undergoes continuous validation and refinement, ensuring that the data used in the models is accurate, up-to-date, and highly relevant.

Furthermore, considering data collected over a period of 5 years, with measurements taken throughout the entire crop development cycle, from sprouting to harvest, allows for addressing significant issues such as the spectral overlap between different crops and the heterogeneity within a single crop (Rana et al., 2025), thus enhancing the generality of features learned by AI models. For the image patches, the applied filtering to produce clean data ensures that only the patches containing the agricultural crop, specifically, those in which the crop has sprouted but has not yet been harvested, are included.

5. Data set value

The integration of new datasets into precision agriculture offers valuable opportunities to enhance the monitoring, analysis, and optimization of farming operations. In this section, we present a series of practical use cases that demonstrate the applicability of the dataset within the context of precision agriculture. These use cases highlight scenarios where data-driven decision-making can lead to improved crop management, resource efficiency, and environmental sustainability.

By exploring these use cases, we aim to showcase the versatility and value of the dataset, providing insights into how it can address real-world challenges. Each use case is discussed in detail, outlining the experimental setups, methodologies employed, and the results obtained. Through these examples, we aim to emphasize the practical impact and innovative possibilities enabled by our dataset.

5.1. Crop identification: past vs present (problem 1)

For the first problem, we consider as being the "present" moment, "*the beginning of 2024*", and therefore all data before (2020, 2021, 2022, 2023) is viewed as available for training. All the data from 2024 is in the "present", thus not available for training the model and is consequently placed in the testing set. In this way, we can study how relevant is the data that *was* acquired in the past, to identify the data that *will be* presently acquired.

To facilitate the AI-based identification of agricultural crops, we used the patches corresponding to the known agricultural crops within the DACIA5 dataset, generated as described in Section 3. The multispectral patches of $32 \times 32 \times 12$ pixels from Sentinel-2, as well as the Sentinel-1 patches of $32 \times 32 \times 2$ were selected from the Potato Institute parcels. The resolution 32×32 has been popularized by the CIFAR 10 dataset (Krizhevsky & Hinton, 2009), which is one of the most used benchmarks in image classification (Brigato et al., 2022) (this report shows that it is by far most used, but data is limited to 2022).

All patches from the years 2020 to 2023 from Sentinel-2 result in a training set of 5431 images, each with a size of $32 \times 32 \times 12$. The test set, based on patches from 2024, comprises 1023 images of the same dimensions. For Sentinel-1, the number of train patches from the years 2020 to 2023 is 5103, from timestamps slightly different than those of the Sentinel-2 patches, while the number of samples in the test set from year 2024 is 892. As previously explained, a matching between the timestamps of the Sentinel-2 to the closest ones from Sentinel-1 data was performed, to allow the fusion of both types of satellite data.

Although the dataset comprises 17 types of crops (in total), some of the parcels were too small to provide patches of the required size, and as a result, these crops are not included. This leaves us with 12 crops, corresponding to 12 classes for our models. The patches were carefully filtered to ensure they predominantly contained pixels representing the target crop and were captured during periods of the year when the crop had sprouted and was clearly visible on the respective parcel. It is important to note that, while the dataset encompasses 12 distinct crop classes, only 8 of these classes were represented in the test set, reflecting variations in crop availability during the selected time period. This refined dataset enables precise and consistent input for our models, ensuring that they receive high-quality, relevant data for crop identification and classification tasks.

Two types of classification experiments were conducted. The first experiment utilized only the Sentinel-2 patches, while the second experiment employed fused data, where each $32 \times 32 \times 14$ image contains Sentinel-1 information in the first two channels and Sentinel-2 data from the corresponding patch in the remaining 12 channels.

5.1.1. Deep learning approach: identification using ResNet18

One of the most prevalent challenges in the literature is the classification and identification of agricultural crops and our dataset can be seen as addressing this issue. Initial experiments that will form a baseline indicating the problem difficulty and exploring this application were conducted using ResNet18 (He et al., 2016). ResNet18 was selected due to its ability to handle complex and diverse visual patterns efficiently. The architecture uses residual connections to mitigate the challenges of the vanishing gradient problem, making it particularly suited for datasets with nuanced agricultural features.

5.1.1.1. Model description and training procedure. For crop identification, we started a pretrained ResNet18 model provided by the PyTorch library. This architecture is well suited for handling image classification tasks involving small-sized inputs, such as the patches in our dataset. Since the Sentinel-2 patches were derived from multispectral images with 12 spectral bands, we modified the input layer of the original ResNet18 model to accommodate the new input dimensions; this was done by replicating the green plane initial weights. Additionally, the network's final layer was replaced with a fully connected layer containing 12 neurons, each corresponding to one of the crop classes. For the experiment with the fused data, the number of input channels was set to 14.

For optimization, we employed the AdamW optimizer (Loshchilov & Hutter, 2019), a variation of the Adam optimizer that effectively handles weight decay, leading to improved generalization and reduced overfitting. The loss function used was the classical cross-entropy loss, as implemented in the PyTorch library.

During training, the model processed the training set in batches of 128 images. However, the dataset was inherently imbalanced, as some crop classes were significantly more represented than others. Such class imbalances can bias the model towards the overrepresented classes, reducing its ability to accurately classify the underrepresented ones. Addressing this issue was critical to ensure the model achieved balanced performance across all crop classes.

To mitigate the effects of class imbalance, we employed the *Imbalanced Data Sampler* from the *torch.utils.data* module in PyTorch, see Imbalanced Dataset Sampler (2022). This sampling strategy dynamically adjusts the composition of batches, ensuring that underrepresented classes are sampled more frequently during training. By doing so, the sampler effectively balances the contribution of each class to the training process, helping the model to learn features from all classes more equitably. This approach is particularly beneficial for datasets where the minority classes are crucial yet insufficiently represented, as it avoids the need for data duplication or excessive augmentation, which could otherwise lead to overfitting. For further details on such techniques, see Buda et al. (2018), which discusses strategies for dealing with imbalanced datasets in deep learning.

The learning rate was set to a standard value of 10^{-3}, which is commonly used for deep learning models in similar tasks. To further prevent overfitting, a weight decay of 10^{-3} was applied. This regularization technique penalized excessively large weights, encouraging the model to rely on more robust features and improving its generalization capability.

5.1.1.2. Results and discussion. For both types of experiments, we ran the training process several times. For the Sentinel-2 set, the accuracy achieved on the test set varied between 60% and 65%, with an average of 62.67%, while for the training set, we

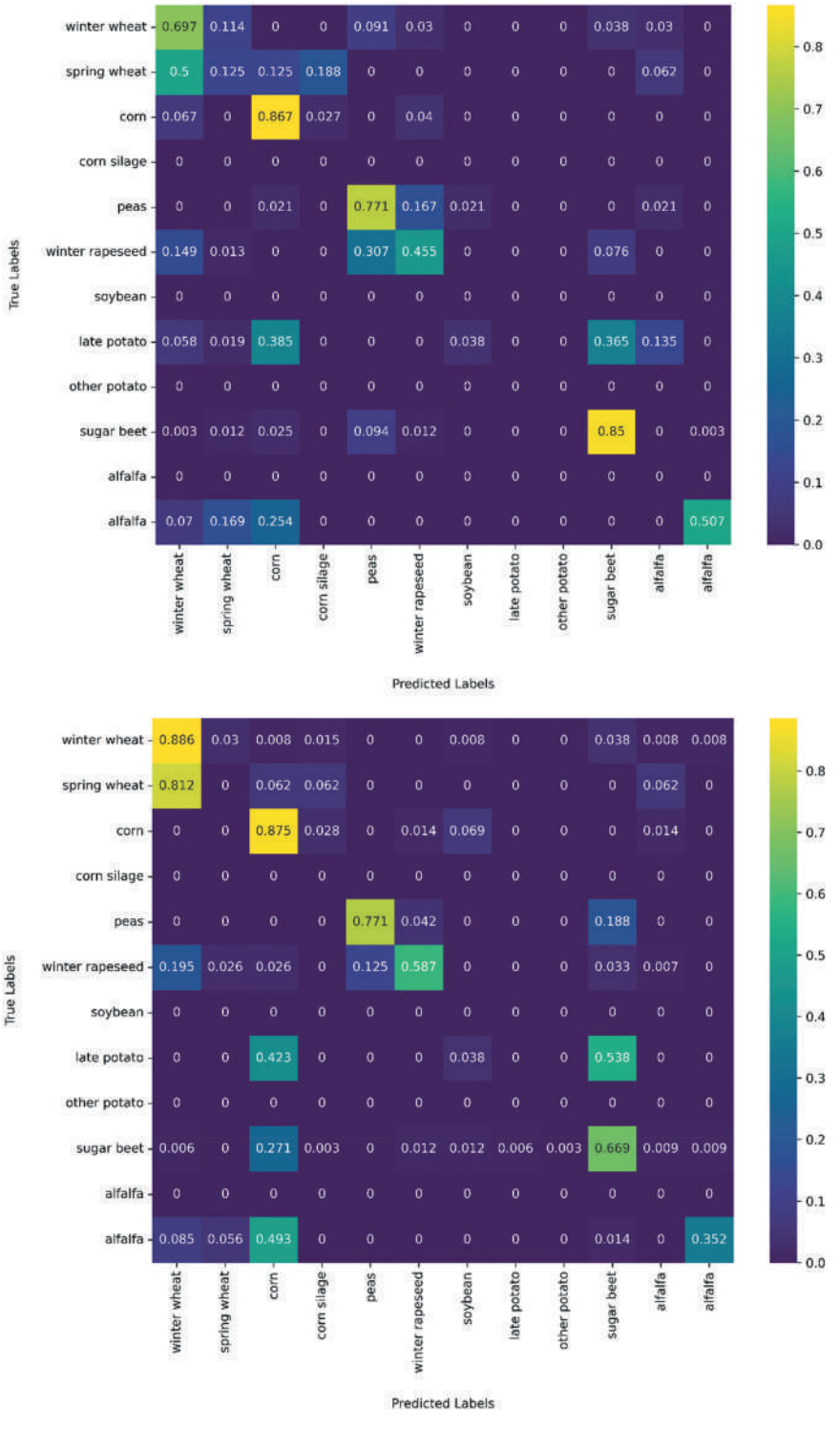


Figure 10. The confusion matrices obtained for two cases are detailed. For the Sentinel-2, the overall accuracy was 62.13% (top), while for the fused Sentinel-1 and Sentinel-2, it was 62.56% (bottom).

consistently obtained accuracies above 90%. To illustrate the model's performance, we selected one example for detailed analysis. For this example, the model achieved an accuracy of 91.72% on the training set and 62.13% on the test set.

For the experiment using the fused data, the best accuracy on the test set was around 55%, obtaining for the train set also a large accuracy above 91%. In this experiment, the training was run several times. The performance in this case is illustrated by one example, in which the accuracy on the train set was 97.33% and, on the test set 62.56%.

The confusion matrices for these two cases are presented in Figure 10. Each cell *(i, j)* of the matrix represents the fraction of images from class *i* that were classified as class *j*. This allows for a detailed evaluation of the model's performance across the different crop classes, revealing potential areas for improvement.

The results from the two datasets – one using Sentinel-2 data and the other combining Sentinel-1 and Sentinel-2 – are similar. The confusion matrices for both tests show nearly identical distributions of true positives, false positives, true negatives, and false negatives. Additionally, the accuracy values are almost the same, indicating that the model performs consistently across different data sources. While the overall accuracy is moderate, the consistency of the results suggests that the model's performance is not significantly influenced by the choice of the dataset. To enhance performance, further model adjustments or data preprocessing may be necessary.

One can notice that several agricultural crops are better identified, such as corn and peas, with true positive rates ≥0.75. Other crops, like wheat and sugar beets, are acceptably-well identified, with rates between 0.5 and 0.7. While for the rest of the crops, the identification exhibits very low accuracy rates.

However, it is worth mentioning that some crops are better classified by adding the Sentinel-1 bands. For example, wheat, potato, and rapeseed present a performance gain, whereas sugar beet and alfalfa show a lower performance. Radar (Sentinel-1) is sensitive to crop structure, and water content, which helps distinguish crops with unique structural characteristics. Optical (Sentinel-2) focuses on leaf reflectance and chlorophyll content. Adding the two bands can make a difference for plants with a more different structure. If the crops are sufficiently different at the spectral level, but similar at the structural level, the use of radar information can downgrade the results.

The classification results can be influenced by several factors, including hyperparameters (such as *learning rate*, *batch size*, *weight decay*, etc.), but also by the inherent imbalance in the training set, where certain classes are underrepresented. Another challenge arises from the presence of similar-looking crop types, which can make accurate classification more difficult.

To address these issues, one potential solution could be to cluster similar crops together, which might help the model focus on distinguishing between the most distinct features of each class. This approach could potentially reduce classification errors and improve the model's overall performance by mitigating the effects of both class imbalance and visual similarity between certain crop types.

There is potential for further improving the performance of the trained model, as well as exploring the use of other models for classification.

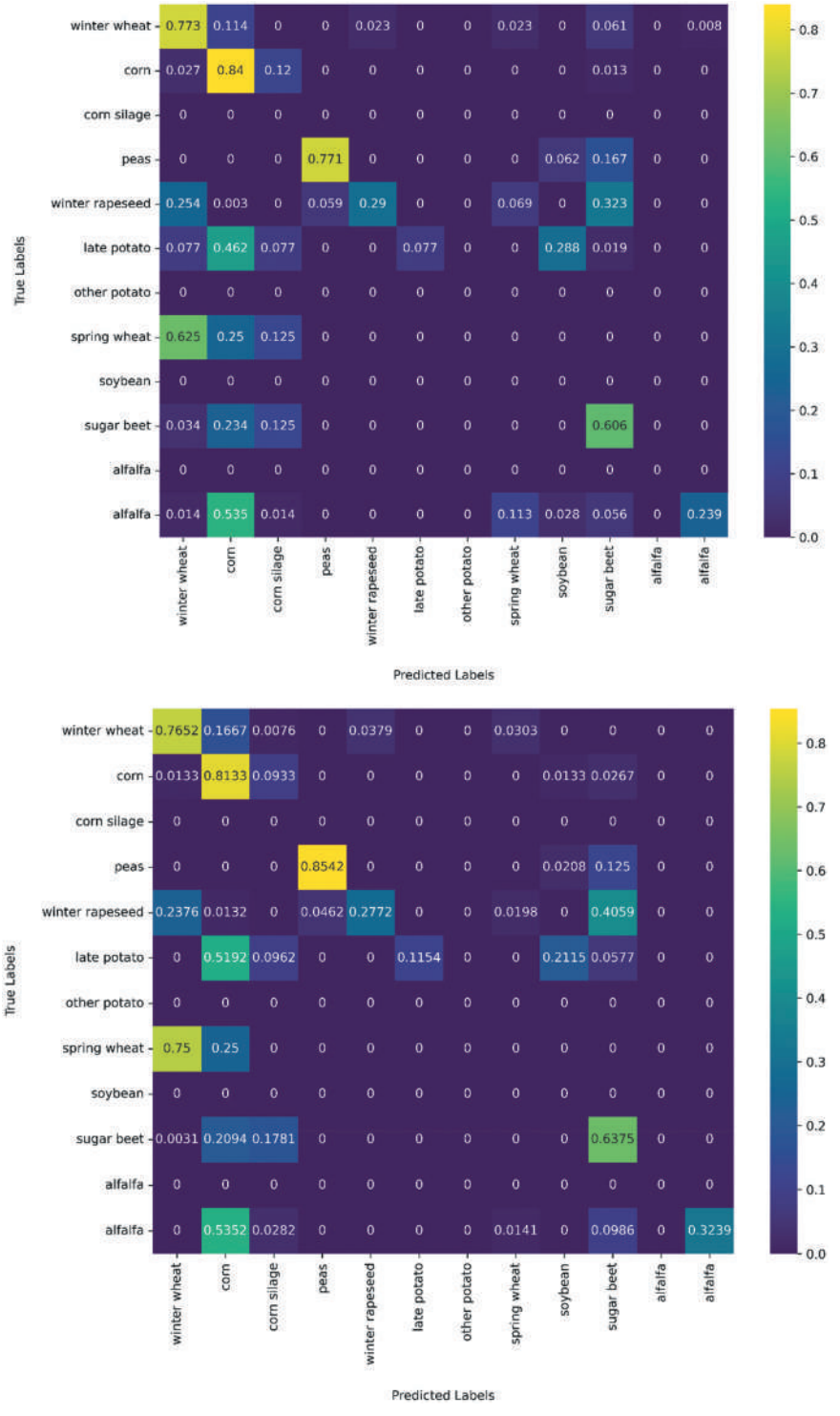


Figure 11. The confusion matrices obtained for the two experiments are detailed. For Sentinel-2 (top), the overall accuracy was 49.6%, while for the fused Sentinel-1 and Sentinel-2 (bottom), it was 54.09%.

5.1.2. Classical approach: identification with Random Forests

Random Forests (RF) (Breiman, 2001) are widely used for classification tasks due to their robust and versatile nature. They consist of an ensemble of decision trees, each trained on a random subset of the data and features. To prevent overfitting and improve generalization, the trees are regularized with injection of randomness, which came two-fold: trees are trained within bootstrapping framework (i.e. on randomly selected subsets) and each node selects a random subset of the input dimension for the split. As a result, Random Forests strike a good balance between bias and variance and can model non-linear relationships in the data without requiring prior assumptions about the data's distribution.

For a classification problem like the one here, the final result is determined by the majority (plurality) vote of the ensemble trees. Considering all these advantages, we conducted a classification experiment for our dataset, training a Random Forest to classify crops.

5.1.2.1. Model description and training procedure. To train the model, we used in the first experiment all Sentinel-2 patches (i.e. 5431) having the full size $32 \times 32 \times 12$. In the second experiment, the fused Sentinel-1 and Sentinel-2 patches (i.e. 5103) of size $32 \times 32 \times 14$ were considered. The test data consisted of all the respective patches from 2024. The input to the model consists of multispectral pixels, with each pixel assigned a label corresponding to the patch from which it was derived. The selected hyperparameters were as follows: 40 decision trees and an InBagFraction of 0.8. The model produced 12 output classes, each representing a crop type present in the training set.

5.1.2.2. Results and discussion. The confusion matrix constructed from the results obtained on the Sentinel-2 test set is shown in Figure 11 (top). Overall accuracy is 49.6%. With respect to individual crops, the best obtained classification score is 84% for corn (code 108), followed by the result for wheat at 77.3%, peas at 77.1% and sugar beet at 60.6%.

The results obtained for the second experiment with the fused test (Sentinel-1 + Sentinel-2) set are presented in Figure 11 (bottom). Overall accuracy is 54.09%. In comparison with the first experiment, we can observe that some classes are better classified, like peas, with an accuracy of 85.42%, while for other classes, like winter wheat (76.52%), the results are slightly worse.

Comparing the results obtained by RF (pixel-based) with those achieved by ResNet18 (patch-based) reveals certain similarities in classification, but at the same time certain discrepancies. Some crops are well identified by the deep model and poorly by the RF and the other way around. Other crops, such as winter wheat, are well separated by both approaches.

The two identification solutions presented here are aimed at offering baseline perspectives for future tests. This use case demonstrates the potential of our dataset in training a CNN model to identify different types of agricultural crops based on past data, a task that is critical for applications in precision agriculture, crop monitoring, and resource management.

5.2. Early crop identification (problem 2)

A second experiment focuses on the early identification of agricultural crops. To achieve this, we selected patches of interest from the refined dataset. We chose 20 May as the date for splitting the dataset into train and test. The date is motivated by the fact that at that moment the sowing process ended and the accumulation of data with respect to areas cultivated by various crops can be done. For Romania, it is also the date when APIA asks for self-reports about crops cultivated and it will start the verification process.

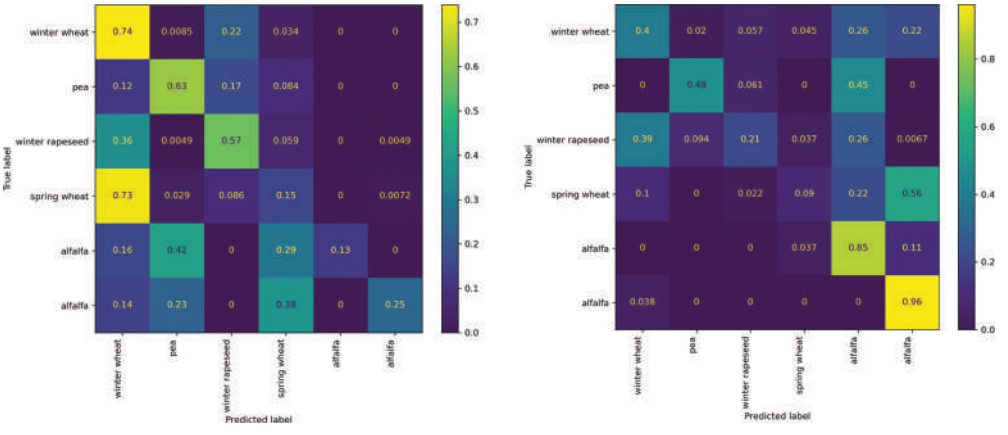
Given this date, we explored two working scenarios. We selected $32 \times 32 \times 12$ patches from the period 2020–2024 up to 20 May only for the crops that had already sprouted, resulting in 1176 patches. Based on the crops present before 20 May, we selected patches from after 20 May in the same period, yielding 1073 patches. Six crops were selected that appeared in both the training and test sets.

In the first working scenario, the model was trained on the patches from before 20 May and tested on those from after 20 May. That would correspond to the standard problem of after harvest identification: recognize older crops from young ones.

In the second scenario, the training and test sets from the first scenario were swapped: the training is done on mature plants, and the testing is on young plants. As mentioned in a previous section, experiments were also performed on the merged data between Sentinel-1 and Sentinel-2, with 1162 patches before 20 May and 1025 after this date. This is motivated from a machine learning perspective to study the dataset compactness and is the standard early crop identification. For the implementation of both scenarios, we used the same model to ensure a fair comparison between the two approaches.

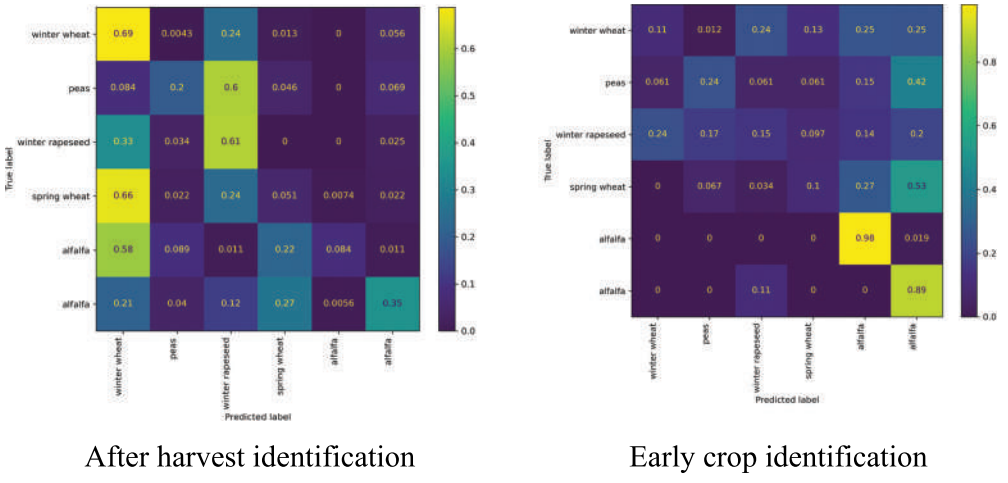
5.2.1. Deep learning approach with ResNet18

5.2.1.1. Model description and training procedure. The ResNet18 model, provided by the PyTorch library preparation, is similar to the Past vs Present crop identification: the input layer of the pretrained is changed to accommodate the input dimensions of $32 \times 32 \times 12$ of Sentinel-2 patches. For the experiment with the Sentinel-1 and Sentinel-2 data, the input dimensions were set to $32 \times 32 \times 14$. The network's final layer was replaced with a fully



After harvest identification Early crop identification

Figure 12. The confusion matrices obtained for the two scenarios, considering only Sentinel-2 data.



After harvest identification Early crop identification

Figure 13. The confusion matrices obtained for the two scenarios with fused data.

connected layer containing six neurons, each corresponding to one of the crop classes. The optimizer used was Adam, and the loss function employed was cross-entropy loss.

During training, the model processed the training set in batches of four images. The dataset was quite unbalanced, some crops were better represented than others. This aspect influenced the model to learn certain agricultural crops better than others, with those that were in a higher proportion achieving better results in identification on the test set.

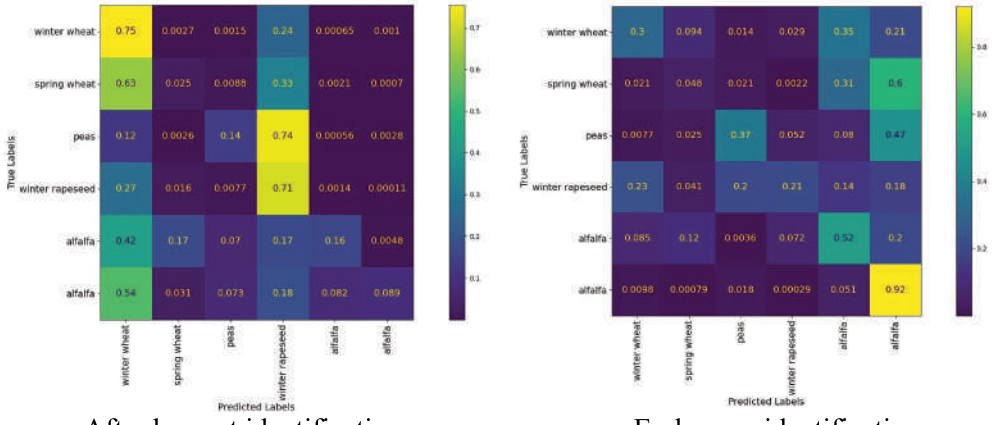
The model was trained for 40 epochs. The learning rate was set to a standard value of 3×10^{-4}. To further prevent overfitting, a weight decay of 3×10^{-4} was applied. The smaller learning rate was chosen in parallel with the smaller batch and to ensure enough iterations as to allow the model to be able to learn.

5.2.1.2. Results and discussion. For the experiment using Sentinel-2 data, in the first working scenario (mature crop identification), the overall accuracy on the test set was 41.01%, while on the training set, it was 91.79%. In the second working scenario (early crop identification), the overall accuracy on the test set was 49.96%, while on the training set, it was 94.25%. The confusion matrices of the two working scenarios can be seen in Figure 12.

For the experiment using the fused data, in the mature crop identification, the overall accuracy on the test set was 33.01%, while on the training set, it was 90.31%. For early crop identification, the overall accuracy on the test set was 41.16%, while on the training set, it was 95.17%. The confusion matrices of the two working scenarios with fused data can be seen in Figure 13.

Through the experiments and scenarios addressed, we found that the model is influenced by the unbalanced amount of data of each agricultural crop, for example in the first scenario we have a higher proportion of wheat crops and in the second scenario the proportion of alfalfa is higher. These aspects are also observed in the confusion matrices, by high accuracy of wheat and of alfalfa, respectively.

When we use Sentinel-2 data, in the after-harvest identification, we observe good identification of wheat, winter rapeseed, and peas. In the early crop identification, the resulting accuracies are different: the model identifies well the alfalfa and peas. Peas are



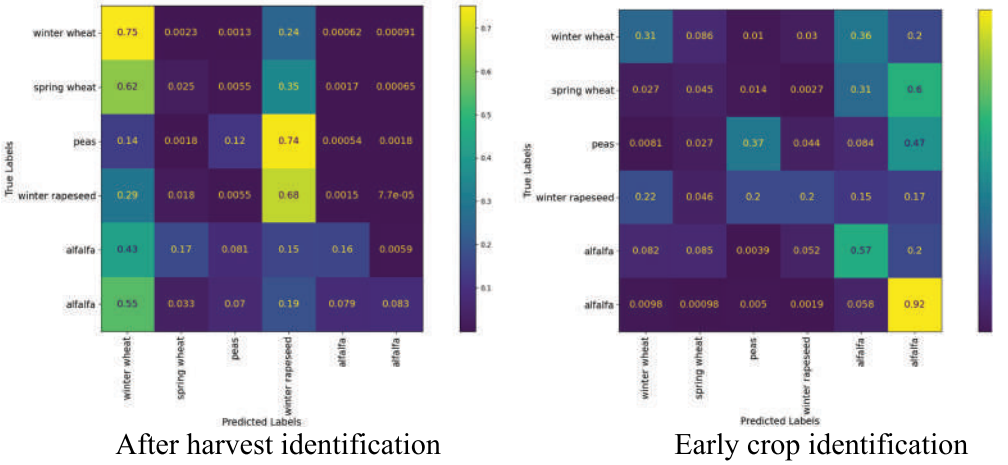
After harvest identification Early crop identification

Figure 14. The confusion matrices were obtained with random forest for the two scenarios.

identified quite well in both scenarios. A major impact on the early identification of crops is also the choice of the date we use to split the dataset; this has to take into account the growing season of the agricultural crops. The use of fused data does not greatly influence the obtained results, the confusion matrices do not change significantly by adding Sentinel-1 data. In the first work scenario wheat and rapeseed are well identified, and in the second work scenario alfalfa is best identified. A difference occurs in the peas crop identification, where a decrease is observed.

5.2.2. Early crop identification using Random Forest

As in the Past vs Present crop identification problem, where ResNet18 and Random Forests were compared, we conducted a similar analysis in this second experiment focused on early crop identification, thus complementing the baseline result with Random Forests.



After harvest identification Early crop identification

Figure 15. The confusion matrices were obtained with random forest for the two scenarios with fused data.

5.2.2.1. Model description and data. The considered dataset was identical to that presented in the previous section, while the Random Forest ensemble is the same as in the previous problem.

5.2.2.2. Results and discussion. In the mature crop identification, where the training set contains images before May 20, and the test set contains images after this date, the total accuracy was 30.64%. Among the very well-identified crops are winter wheat and rapeseed with a percentage of 70–75%. The highest confusions are between peas and rapeseed and the two types of wheat. The confusion matrix for this scenario is represented in Figure 14, left subfigure.

In the second scenario, where the training set contains images after May 20, and the test set contains images before this date, the total accuracy was 35.65%. In this case, the crops correctly identified are two types of alfalfa. The confusions are also present for most of the crops with alfalfa classes. The confusion matrix for the second case can be seen in Figure 14, right subfigure.

As mentioned in Section 5.2.1, a major impact on the results is the uneven distribution of classes. Also, the division of the data set containing crops in different vegetation phases can be an explanation. For example, in the first case (Figure 14 -left) many confusions are made with winter wheat because it is the crop with a more advanced vegetation phase in spring. The test images, being selected from 20 May onwards, contain crops with more vegetation, which is why they can be confused with the most developed crop that was present in the training set.

For the two fused data experiments, the results are quite similar in terms of overall performance and confusion matrices, as illustrated in Figure 15. The accuracies are 30.4% for after harvest identification and 35.28% for early identification, respectively. It can be observed that in the case of early identification there is an increase for the alfalfa crop, but otherwise the results are similar. The results for Random Forest are more stable (there are no major differences between the Sentinel-2 data only and the merged data) because it treats each band separately, avoids overfitting, and is less sensitive to texture differences. CNNs have high capacity and might overfit to radar features for some crops while failing to generalize well for others.

5.2.2.3. Comparison between deep learning and Random Forest approaches. When the two approaches are compared, as expected, the deep learning model presents better results. Having a convolutional layer and more parameters, the ResNet18 model recognizes better local structure of each crop (such as plant spatial organization, and texture) and is therefore able to outperform the Random Forest. The latter, due to its nature, does only local and linear approximations of the crop texture and structure.

6. Conclusions

In the context of smart agriculture, the proposed DACIA5 dataset can contribute to the development of AI-based tools for agricultural crop identification. The dataset presented in this paper is publicly available on Zenodo, to further help the community in developing algorithms and methods for crop identification and monitoring. The provided dataset comprises Sentinel-1 SAR data and Sentinel-2 multispectral images. This ensures a balanced representation of different observational modalities, making the dataset more versatile and

adaptable to various scenarios and use cases within precision agriculture. By integrating both radar (Sentinel-1) and optical (Sentinel-2) imagery, we aim to provide a comprehensive dataset that facilitates model generalization across diverse agricultural crops, improving its applicability in real-world settings. This complementarity is a key strength of our dataset, allowing it to be leveraged in different methodological approaches and machine learning frameworks. Furthermore, the dataset is highly reliable, with accurate labels verified in-situ by accredited institutions like the National Institute of Research and Development for Potato and Sugar Beet, Brașov, Romania. The data set is spanning over 5 years (2020–2024) and comprising 17 types of agricultural crops on 47 parcels. With its 172 Sentinel-2 multispectral images (each of them 800 × 450 pixels), 159 Sentinel-1 radar images, together with its 6,454 Sentinel-2 and its 5,995 Sentinel-1 rectangular patches (of size 32 × 32 pixels) the dataset comprises over 6 M pixels. Ground truth data is essential, particularly for machine learning approaches, as it helps reduce errors in crop identification applications. The 32 × 32-pixel size patches allow for addressing two problems (use cases). We introduce two use cases for our dataset: past versus present agricultural crop identification and late versus early crop identification using machine learning models. Our dataset can be further used for new models and results, thereby contributing to the global efforts towards improved resource and land management.

Additionally, the dataset can be integrated or concatenated with other datasets through recently developed spectral image data aggregation methods (Luca et al., 2025). Current research has demonstrated how combining datasets with interpolation techniques can enhance machine learning models' generalization capabilities across different geographic regions. Such methodologies allow researchers to effectively utilize geographically constrained datasets while preserving the benefits of highly accurate labeling.

However, there are several limitations to consider. Although the dataset includes a substantial amount of multispectral pixel data, its size is still relatively small for training deep neural networks. Cloudy satellite images were excluded during preprocessing, and future work could explore including such cases to improve model robustness. The five-year timespan, while valuable, may not fully capture longer-term climatic variations. Furthermore, the dataset lacks complementary contextual information such as weather conditions, disease occurrences, or agricultural interventions, which could impact plant development. While the data was collected from a limited geographic region and may reflect local-specific patterns, the data aggregation approaches mentioned above can help researchers address this spatial limitation by combining our highly accurate dataset with others that offer greater geographic diversity. By making this dataset publicly available, we hope to support the field of precision agriculture and the development of more accurate, efficient, and sustainable farming practices through improved crop identification and monitoring technologies.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

Funded by the European Union. The AI4AGRI project entitled "Romanian Excellence Center on Artificial Intelligence on Earth Observation Data for Agriculture" received funding from the European Union's Horizon Europe research and innovation program under grant agreement

no. 101079136. Views and opinions expressed are, however, those of the authors only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them; AI4AGRI project received funding from the European Union's Horizon Europe research and innovation programme [101079136].

Notes on contributors

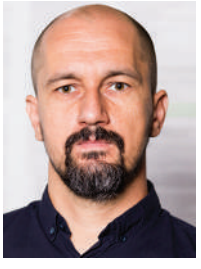
Alexandra Baicoianu holds a PhD in Computer Science from Babeș-Bolyai University, Cluj-Napoca. She currently serves as an Associate Professor in the Department of Mathematics and Computer Science at Transilvania University of Brașov, where she teaches various courses and seminars on Algorithms, Formal Languages and Compilers, Data Mining, Data Warehousing, etc. With a strong academic background, she has published over 30 peer-reviewed scientific papers in prominent journals and conferences. She has also contributed to the academic literature as a co-author of six books, reflecting her commitment to advancing research and education in the field.

Ioana Cristina Plajer received her PhD in Computer Science from Transilvania University of Brasov, Romania, in 2011 and is currently a Lecturer with the Faculty of Mathematics and Computer Sciences of the Transilvania University. She is also a member of the Artificial Intelligence and Earth Observation for Romania's agriculture (AI4AGRI) European project. Her research interests include machine learning, image processing, spectral imaging and remote sensing.

Matei Debu received a bachelor's degree in computer science from the Transilvania University of Brașov in 2023 and is currently pursuing a master's degree in Modern Technologies in Software Systems Engineering at the same university. He is actively involved as a research assistant in several projects, including notable initiatives such as AI4AGRI, which focuses on the integration of artificial intelligence in agriculture, and other projects like AI4RiSK, IMINT and SEEN all of them focus on using artificial intelligence in different scenarios. His research interests span artificial intelligence, software systems design, and the application of modern technologies to solve complex, real-world problems.

Floriana Maria Ștefan is the Head of Potato Genetic Breeding and Selection Lab at NIRDPSB Brasov and also serves as an Associate Professor at the Faculty of Food and Tourism of Transilvania University Brasov. As a horticultural engineer, her expertise lies in potato genotyping and phenotyping. Her research interests further extend to the development of temporal and spatial data.

Mihai Ivanovici received his PhD from the Politehnica University of Bucharest, Romania, in 2006. He is currently a full professor at Transilvania University of Brasov, Romania. His research interests include signal and image processing and analysis, as well as remote sensing and Earth Observation data analysis.



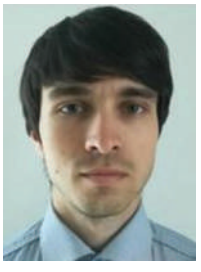
Corneliu Florea earned his master's degree from the National University of Science and Technology Politehnica of Bucharest, Romania, in 2004 and a PhD from the same university in 2009. After a stint with digital still camera software industry, he currently serves as a professor at the university's Image Processing and Analysis group. There, he teaches courses on machine learning, computer vision, and introductory statistical information processing. His research interests include statistical approaches to machine learning and computer vision, yielding over 80 peer-reviewed papers and of more than 25 U.S. patents.

Angel Cațaron obtained his PhD in 2004 from the Politehnica University of Bucharest. He is currently affiliated with Transilvania University of Brașov. His research interests focus on data analysis, machine learning and their applications in addressing complex real-world challenges, with an emphasis on the practical use of predictive modeling and data-driven decision-making.

Radu-Mihai Coliban is an Associate Professor in the Department of Electronics and Computers at Transilvania University of Brasov, Romania. His research interests include hyperspectral image processing, signal processing and digital electronics.

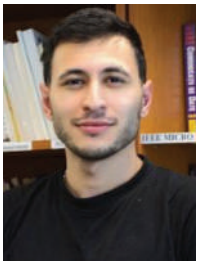
Ștefan Popa received his PhD in Electronics, Telecommunications and Information Technology in 2021 from the Transilvania University of Brasov, Romania. Currently, a researcher in the university's Electronics and Computers Department, his work spans FPGA and ASIC design, data structures and algorithms, artificial intelligence, signal and image processing.

Serban Oprişescu got his PhD in Electronics and Telecommunications from the Politehnica University of Bucharest, Romania, in 2007. He is currently a lecturer with Transilvania University of Braşov, Romania. He holds a B.S. degree in Electrical Engineering and Computer Science (2002) and an M.S. degree in Biomedical Engineering (2003) from the University Politehnica of Bucureşti. His research focuses on image and video processing, ToF cameras, biomedical image processing and analysis and computer science. He has authored two ISI journal papers and more than 23 conference papers.



Andrei Racoviţeanu is a lecturer at the National University of Science and Technology Politehnica Bucharest. He is also a researcher within the Image Analysis and Processing Laboratory. His research interests include image processing and analysis, machine learning and remote sensing.

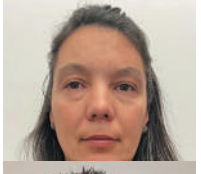
Gheorghe Olteanu is an agronomy expert at Transilvania University of Braşov. His research focuses on in-situ measurements to monitor vegetation status, contributing to advancements in precision agriculture and sustainable land management.



Kamal Marandskiy received a B.Sc. in Electronics, Telecommunication and Radio Engineering from the National Aviation Academy of Azerbaijan in 2020, and his M.Sc. in Advanced Electrical Systems from Transilvania University of Brasov in 2022. He is currently pursuing a PhD in Electronics and Telecommunications. His research focuses on machine learning techniques for Earth observation data analysis.

Adrian Ghinea is a network engineer at NIRDPSB Brasov. He holds a master's degree in Computer Science, specializing in Algorithms and Software Products from the Faculty of Mathematics and Computer Science at Transilvania University of Brasov. His research focuses on remote sensing.

Artur Kazak is a PhD student in Electronics, Telecommunications, and Information Technologies at Transilvania University of Brașov. His research focuses on AI-based analysis of Earth Observation data, aiming to advance techniques for environmental monitoring and decision-making.



Luciana Majercsik received the B.S. degree in Mathematics from the University of Bucharest, Romania, and completed her PhD in Computer Science at the Transilvania University of Brasov, Romania. Her research interests include machine learning, multi- and hyperspectral image analysis and visualization, as well as graph-based methods in remote sensing.

Adrian Constantin Manea earned a PhD in Computers and Information Technology from Transilvania University. His research focuses on the legal and security aspects of Earth Observation, exploring the intersection of technology, law, and data protection.

Liviu Doru Dogar is a PhD student in Computer and Information Technologies at Transilvania University. His research focuses on surveillance camera systems and data acquisition techniques for monitoring applications.

ORCID

A. Băicoianu http://orcid.org/0000-0002-1264-3404

Data availability statement

Data is available on Zenodo at https://doi.org/10.5281/zenodo.14915950.

References

Akkem, Y., Biswas, S. K., & Varanasi, A. (2023). Smart farming using artificial intelligence: A review. *Engineering Applications of Artificial Intelligence*, *120*, 105899. https://doi.org/10.1016/j.engappai.2023.105899

Alzubaidi, L., Bai, J., Al-Sabaawi, A., Santamaría, J., Albahri, A. S., Al-Dabbagh, B. S. N., Fadhel, M. A., Manoufali, M., Zhang, J., Al-Timemy, A. H., Duan, Y., Abdullah, A., Farhan, L., Lu, Y., Gupta, A., Albu, F., Abbosh, A., & Gu, Y. (2023). A survey on deep learning tools dealing with data scarcity: Definitions, challenges, solutions, tips, and applications. *Journal of Big Data*, *10*(1), 46. https://doi.org/10.1186/s40537-023-00727-2

Anderson, R., Bayer, P. E., & Edwards, D. (2020). Climate change and the need for agricultural adaptation. *Current Opinion in Plant Biology*, *56*, 197–202. https://doi.org/10.1016/j.pbi.2019.12.006

Aschbacher, J. (2017). ESA's earth observation strategy and Copernicus. In *Satellite earth observations and their impact on society and policy* (pp. 81–86). Springer Open. https://doi.org/10.1007/978-981-10-3713-9_5

Beck, H. E., McVicar, T. R., Vergopolan, N., Pehrsson, R., Huang, X., Nijssen, B., Zhang, Y., Ukkola, A. M., Wood, E. F., Schumacher, D. L., Weerasinghe, I., & Sheffield, J. (2023). High-resolution (1 km) Köppen-Geiger maps for 1901–2099 based on constrained CMIP6 projections. *Scientific Data*, *10* (1), 724. https://doi.org/10.1038/s41597-023-02549-6

Benami, E., Jin, Z., Carter, M. R., Ghosh, A., Hijmans, R. J., Hobbs, A., Kenduiywo, B., & Lobell, D. B. (2021). Uniting remote sensing, crop modelling and economics for agricultural risk management. *Nature Reviews Earth and Environment*, *2*(2), 140–159. https://doi.org/10.1038/s43017-020-00122-y

Borisov, V., Leemann, T., Seßler, K., Haug, J., Pawelczyk, M., & Kasneci, G. (2024). Deep neural networks and tabular data: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, *35*(6), 7499–7519. https://doi.org/10.1109/TNNLS.2022.3229161

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32. https://doi.org/10.1023/A:1010933404324

Brigato, L., Barz, B., Iocchi, L., & Denzler, J. (2022). Image classification with small datasets: Overview and benchmark. *Institute of Electrical and Electronics Engineers Access*, *10*, 49233–49250. https://doi.org/10.1109/ACCESS.2022.3172939

Buda, M., Maki, A., & Maki, D. (2018). A systematic study of the class imbalance problem in convolutional neural networks. *Computers in Biology and Medicine*, *98*, 107–118. https://doi.org/10.1016/j.neunet.2018.07.011

Cabrera, G. F., Miller, C. J., & Schneider, J. (2014). Systematic labeling bias: De-biasing where everyone is wrong. *2014 22nd International Conference on Pattern Recognition* (pp. 4417–4422). IEEE. https://doi.org/10.1109/ICPR.2014.756

Copernicus Browser. (2024). Retrieved July 10, 2024, from https://dataspace.copernicus.eu/explore-data

Drusch, M., Del Bello, U., Carlier, S., Colin, O., Fernandez, V., Gascon, F., Hoersch, B., Isola, C., Laberinti, P., Martimort, P., Meygret, A., Spoto, F., Sy, O., Marchese, F., & Bargellini, P. (2012). Sentinel-2: ESA's optical high-resolution mission for GMES operational services. *Remote Sensing of Environment*, *120*, 25–36. https://doi.org/10.1016/j.rse.2011.11.026

ECA - European Court of Auditors. (2020). *Special report: Using new imaging technologies to monitor the common agricultural policy: Steady progress overall, but slower for climate and environment monitoring*, https://www.eca.europa.eu/en/publications/SR20_04

European Space Agency. (n.d.). Retrieved July 11, 2024, from https://www.esa.int/

Frona, D., Szenderák, J., & Harangi-Rákos, M. (2019). The challenge of feeding the world. *Sustainability*, *11*(20), 5816. https://doi.org/10.3390/su11205816

Gascon, F., Bouzinac, C., Thépaut, O., Jung, M., Francesconi, B., Louis, J., Lonjou, V., Lafrance, B., Massera, S., Gaudel-Vacaresse, A., Languille, F., Alhammoud, B., Viallefont, F., Pflug, B., Bieniarz, J., Clerc, S., Pessiot, L., Trémas, T. … Martimort, P. (2017). Copernicus Sentinel-2A calibration and products validation status. *Remote Sensing*, *9*(6), 584. https://doi.org/10.3390/rs9060584

Gascon, F., Cadau, E., Colin, O., Hoersch, B., Isola, C., Fernández, B. L., & Martimort, P. (2014). Copernicus Sentinel-2 mission: Products, algorithms and Cal/Val. In Prasad S. Thenkabail (ed), *Earth observing systems XIX* (Vol. 9218, pp. 455–463). SPIE. https://doi.org/10.1117/12.2062260

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 770–778). https://doi.org/10.1109/CVPR.2016.90

Helber, P., Bischke, B., Dengel, A., & Borth, D. (2019). Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, *12*(7), 2217–2226. https://doi.org/10.1109/JSTARS.2019.2918242

Imbalanced Dataset Sampler. (2022). *GitHub repository*. Retrieved July 11, 2024, from https://github.com/ufoym/imbalanced-dataset-sampler

INCDCSZ. (n.d.). National Institute of Research and*Development for potato and sugarbeet*. Retrieved July 11, 2024, from https://potato.ro/

Ivanovici, M., Baicoianu, A., Plajer, I. C., Debu, M., Ștefan, F.-M., Florea, C., Cațaron, A., Coliban, R.-M., Popa, S., Oprisescu, S., Racoviteanu, A., Olteanu, G., Marandskiy, K., Ghinea, A., Kazak, A., Majercsik, L., Manea, A., & Dogaru, L. (2024). *AI4AGRI Sentinel-2 Brasov area 2020-2024 multi-spectral dataset for crop monitoring and identification [data set]*. Zenodo. https://doi.org/10.5281/zenodo.14283243

Ivanovici, M., Olteanu, G., Florea, C., Coliban, R. M., Ștefan, F., & Marandskiy, K. (2024). Digital transformation in agriculture. https://doi.org/10.1007/978-3-031-63337-9_9

Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

Lee, J. S. (1980). Digital image enhancement and noise filtering by use of local statistics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-*PAMI-2*(2), 165–168. https://doi.org/10.1109/TPAMI.1980.4766994

Liedekerke, M. V., Jones, A., & Panagos, P. (2006). *ESDBv2 raster library - a set of rasters derived from the European soil database distribution v2.0*. European Commission and the European Soil Bureau Network.

Loshchilov, I., & Hutter, F. (2019). Decoupled weight decay regularization. *International Conference on Learning Representations* (pp. (ICLR)). https://doi.org/10.48550/arXiv.1711.05101

Luca, R. I., Baicoianu, A., & Plajer, I. C. (2025). Spectral image data aggregation for multisource data augmentation. *European Journal of Remote Sensing*, *58*(1). https://doi.org/10.1080/22797254.2025.2492295

Maja, M. M., & Ayano, S. F. (2021). The impact of population growth on natural resources and farmers' capacity to adapt to climate change in low-income countries. *Earth Systems and Environment*, *5*(2), 271–283. https://doi.org/10.1007/s41748-021-00209-6

Marandskiy, K., & Ivanovici, M. (2024). Early identification of potato fields using data fusion and artificial neural network. *2024 International Symposium on Electronics and Telecommunications (ISETC)*, Timisoara, Romania (pp. 1–4). IEEE. https://doi.org/10.1109/ISETC63109.2024.10797316

Panagos, P. (2006). The European soil database. *GEO: Connexion*, *5*(7), 32–33.

Panagos, P., Van Liedekerke, M., Borrelli, P., Köninger, J., Ballabio, C., Orgiazzi, A., Lugato, E., Liakos, L., Hervas, J., Jones, A., & Montanarella, L. (2022). European soil data centre 2.0: Soil data and knowledge in support of the EU policies. *European Journal of Soil Science*, *73*(6), e13315. https://doi.org/10.1111/ejss.13315

Payments and Intervention Agency for Agriculture. (n.d.). Retrieved July 11, 2024, from https://apia.org.ro/

Peel, M. C., Finlayson, B. L., & McMahon, T. A. (2007). Updated world map of the Köppen-Geiger climate classification. *Hydrology and Earth System Sciences*, *11*(5), 1633–1644. https://doi.org/10.5194/hess-11-1633-2007

Pour, A. B., Parsa, M., & Eldosouky, A. M. (Eds.). (2023). *Geospatial analysis applied to mineral exploration: Remote sensing, GIS, geochemical, and geophysical applications to mineral resources*. Elsevier.

Rana, S., Gerbino, S., & Carillo, P. (2025). Study of spectral overlap and heterogeneity in agriculture based on soft classification techniques. *MethodsX*, *14*, 103114. https://doi.org/10.1016/j.mex.2024.103114

Research and Development Institute of Transilvania University of Brașov. (2024). Retrieved July 11, 2024, from https://icdt.unitbv.ro/ro/contact.html

Ritchie, H., & Rodes-Guiaro, L. (2024). *Peak global population and other key findings from the 2024 UN world population prospects*. *Our world in data*. Retrieved October 18, 2024, from https://ourworldindata.org/un-population-2024-revision

Schmitt, M., Hughes, L. H., Qiu, C., & Zhu, X. X. (2019). SEN12MS – a curated dataset of georeferenced multi-spectral Sentinel-1/2 imagery for deep learning and data fusion. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *IV-2*(W7), 153–160. https://doi.org/10.5194/isprs-annals-IV-2-W7-153-2019

Sentinel-1 Toolbox. (n.d.). Retrieved February 12, 2025, from https://earth.esa.int/eogateway/tools/sentinel-1-toolbox

Sentinel-2 bands. (n.d.). Retrieved July 17, 2024, from https://custom-scripts.sentinel-hub.com/custom-scripts/sentinel-2/bands/

Sentinel-2 processing. (n.d.). Retrieved July 11, 2024, from https://sentinels.copernicus.eu/web/sentinel/sentinel-data-access/sentinel-products/sentinel-2-data-products/collection-1-level-2a

Sentinel Application Platform. (n.d.). Retrieved July 11, 2024, from https://step.esa.int/main/download/snap-download/

SNAP - Resampling methods. (n.d.). Retrieved February 12, 2024, from https://step.esa.int/main/wp-content/help/versions/11.0.0/snap/org.esa.snap.snap.help/general/overview/ResamplingMethods.html

Song, H., Kim, M., Park, D., Shin, Y., & Lee, J. G. (2022). Learning from noisy labels with deep neural networks: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, *34*(11), 8135–8153. https://doi.org/10.1109/TNNLS.2022.3152527

Sumbul, G., Charfuelan, M., Demir, B., & Markl, V. (2019). BigEarthNet: A large-scale benchmark archive for remote sensing image understanding. In *IGARSS, 2019 - 2019 IEEE International Geoscience and Remote Sensing Symposium* (pp. 5901–5904). https://doi.org/10.1109/IGARSS.2019.8900532

Teaci, D. (1980). *Bonitarea terenurilor agricole (Bonitarea și caracterizarea tehnologică a terenurilor agricole)* (p. 296). Editura Ceres, București.

Torres, R., Snoeij, P., Geudtner, D., Bibby, D., Davidson, M., Attema, E., Potin, P., Rommen, B., Floury, N., Brown, M., Navas Traver, I., Deghaye, P., Duesmann, B., Rosich, B., Miranda, N., Bruno, C., L'Abbate, M., Croci, R. . . . Rostan, F. (2012). GMES Sentinel-1 mission. *Remote Sensing of Environment*, *12*, 0034–4257. https://doi.org/10.1016/j.rse.2011.05.028

Tseng, G., Zvonkov, I., Nakalembe, C. L., & Kerner, H. (2021). CropHarvest: A global dataset for crop-type classification. In *NeurIPS, 2021 datasets and benchmarks track*. https://openreview.net/forum?id=JtjzUXPEaCu

U.S. Department of Agriculture. (2024). Retrieved July 12, 2024, from https://www.nass.usda.gov/Research_and_Science/Cropland/sarsfaqs2.php

Weikmann, G., Paris, C., & Bruzzone, L. (2021). TimeSen2Crop: A million labeled samples dataset of Sentinel-2 image time series for crop-type classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, *14*, 4699–4708. https://doi.org/10.1109/JSTARS.2021.3073965

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 15 September 2023, accepted 26 October 2023, date of publication 2 November 2023, date of current version 8 November 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3329732

SURVEY

An Extended Survey Concerning the Significance of Artificial Intelligence and Machine Learning Techniques for Bug Triage and Management

RAZVAN BOCU[1,2], ALEXANDRA BAICOIANU[1,2], AND ARPAD KERESTELY[2]
[1]Department of Mathematics and Computer Science, Transilvania University of Braşov, 500091 Braşov, Romania
[2]Department of Research and Technology, Siemens Industry Software, 500227 Braşov, Romania

Corresponding author: Razvan Bocu (razvan.bocu@unitbv.ro)

ABSTRACT Bug reports are generated in large numbers during the software development processes in the software industry. The manual processing of these issues is usually time consuming and prone to errors, consequently delaying the entire software development process. Thus, a properly designed bug triage and management process implies that essential operations, such as duplicate detection, bug assignments to proper developers, and determination of the importance level, are sustained by efficient algorithmic models and implementation approaches. Designing and implementing a proper bug triage and management process becomes an essential scientific research topic, as it may significantly optimize the software development and business process in the information technology industry. Consequently, this paper thoroughly surveys the most significant related scientific contributions analytically and constructively, distinguishing it from similar survey papers. The paper proposes optimal algorithmic and software solutions for particular real-world use cases that are analyzed. It concludes by presenting the most important open research questions and challenges. Additionally, the paper provides a valuable scientific literature survey for any researcher or practitioner in software bug triage and management systems based on artificial intelligence and machine learning techniques.

INDEX TERMS Bug report, bug prioritization, bug assignment, bug triaging, classification, machine learning.

I. INTRODUCTION

Large software development projects rely on bug triaging as an important part of software testing. Thus, it supports the software bug management processes, while allowing relevant decisions, which are related to the software bug fixing, to be made. Relevant tasks are represented by properly assigning bugs to adequate developers, prioritizing bugs, and detecting duplicate bugs. Nevertheless, manual bug triaging appears as an essentially time consuming and tedious task, considering that a significant part of software development requires a lot of time and other types of resources. Considering old statistics from August 2009, the Mozilla bug database contained over 500,000 bug reports, and the Eclipse bug database had over 250,000 bug reports. The average number of bug reports created daily amounts to 170 for the Mozilla database and 120 for the Eclipse database between January and July 2009. The dynamics of software systems development have constantly increased during the past fifteen years. Therefore, the problem that is approached in this paper becomes increasingly more relevant.

The process of bug triage involves that a triager makes a decision regarding the bugs entered in the respective bugs repository through an analysis, which involves two variants. Thus, the repository-oriented decisions involve that the reported bug does not represent a duplicate, the person that triages checks it for validity, which means that the bug is assessed whether it is genuine. This mediates the removal of bug reports that do not require a resolution. The remaining bug reports are investigated to support the development-oriented decisions, which involve that the triager assesses the severity and priority levels of the bugs. These levels are modified if inappropriate values are observed, so sufficient resources are allocated to resolve critical bugs. Consequently,

The associate editor coordinating the review of this manuscript and approving it for publication was Xinyu Du.

the person that addresses the manual bug triaging process writes down the required remarks for the bug and assigns this bug report to the suitable developer. Therefore, the need to design and implement an efficient bug triaging and management system becomes clear.

In this context, the task of bugs classification, which implies the determination of priorities, appears as a very important effort package relative to very large software development projects, and also open source projects, considering that the efficiency of the development process is usually quantitatively assessed considering the number of open bug reports, and the average resolution time. The manual determination of bug priorities also introduces inherent human errors to the process, which essentially depends on the bug triager's subjective perspective and experience. Consequently, a significant number of bug reports may have been assigned incorrect priority levels while other bug reports remain unaddressed. The implied economic and operational consequences are easily discernible. Therefore, an automatic bug triaging and management strategy, which uses a certain automated approach, is required. Thus, the related scientific literature includes various machine learning approaches, such as Decision Tree (DT), Support vector machine (SVM) classification algorithms, Naive Bayes (NB) classifiers, Information Retrieval (IR), and Random Forest (RF) models. This paper concentrates on a logically structured survey, which aims to analyze and suggest the optimal bug triaging and management approaches. The scientific objectives of this research are the following.

- To survey the existing literature to determine shortcomings and propose optimal solutions.
- Identification of the most relevant studies in bug classification.
- Prioritizing studies relevant to bug sorting according to various criteria: citations, scientific relevance, objectives achieved, reproducibility of experiments, genericity of solutions, etc.
- Classification by various criteria of frameworks dedicated to the open problem.
- To determine the relevant scientific research trends.
- To identify the relevant research problems.
- To define the scientific relevance of the corresponding content of research.
- To propose the conceptual and practical relevance of automatic bug triaging in the management processes.

The rest of the paper is structured according to the following sections. The next section presents the structured research methodology, which has been considered. Following, the most relevant artificial intelligence models are described and analyzed. Moreover, the relevant performance evaluation methods and related metrics are surveyed and assessed. Furthermore, the most relevant scientific challenges and open research questions are discussed. Consequently, the essential research questions, which determined this extended survey process, are analyzed, and the degree of this paper's accomplishment is objectively assessed. The last section concludes the paper.

TABLE 1. Reference scientific literature databases and academic search engines.

Database	Public URL
Science Direct-Elsevier - DL	http://www.sciencedirect.com/
Scopus - SE	http://www.scopus.com/
IEEE Xplore - DL	http://ieeexplore.ieee.org/
ACM Digital library - DL	http://dl.acm.org/dl.cfm
Web of Science - SE	https://www.webofknowledge.com/
Wiley online library - DL	https://onlinelibrary.wiley.com/
Google Scholar - SE	https://scholar.google.ro/
Sensors - DL	https://www.mdpi.com/
Springer - DL	https://www.springer.com/
ResearchGate - Scientific social networking	https://www.researchgate.net/
Edinburgh library database - DL	https://my.napier.ac.uk/Library/
RefSeek - SE	https://www.refseek.com/
Bielefeld Academic Search Engine - SE	https://www.base-search.net/

II. RESEARCH METHODOLOGY

The survey methodology relates to a systematic review (SR) approach, which is determined by the methodology that is referred to as ''Preferred Reporting Items for Systematic Reviews and Meta-Analysis'' (PRISMA) [1]. More precisely, the scientific methodology is based on the following phases: specification of research questions, identification and survey of proper papers, and specification of the relevant inclusion and exclusion criteria.

A. RESEARCH QUESTIONS

The literature review relates to the following research questions.

- What is the related significant literature, which approaches conceptual problems, and reports adequate solutions?
- What are the relevant scientific research trends?
- What are the determined research questions and shortcomings?
- What is the reviewed research scope's conceptual, scientific, and real-world importance?
- What are the principal algorithmic and machine learning models that specify and implement automatic bug triaging and management approaches?

The following subsection describes the logical structure of the proper research process.

B. RESEARCH PROCESS

The reference sources that were considered in order to collect the proper scientific literature are described in Table 1. Here, *DL* means Digital Library, and *SE* means Search Engine.

The next subsection presents the exclusion and inclusion criteria, which have been used to filter the scientific contributions objectively.

TABLE 2. Inclusion criteria.

Inclusion criteria
Articles should be indexed by at least one of the selected scientific literature databases.
Scientific papers are published from 2010-2023, and significant historical papers are also selected.
Articles should meet at least one of the search terms, relative to this review article's title, abstract, and keywords.
Papers should be published in indexed journals, conference proceedings, mainstream technical journals, or books and chapters issued by top tier publishing houses.
Reviewed articles should clearly analyze and answer specified research questions.
A search that relates to the title, abstract, and the full text is sufficient.

TABLE 3. Exclusion criteria.

Exclusion criteria
Articles that are not written in English.
Duplicated articles, which are identified using more than one of the specified scientific literature databases.
Articles with full texts that cannot be retrieved.
Articles that are only marginally relevant for studying automatic bug triaging and management, and related artificial intelligence models.

C. EXCLUSION AND INCLUSION CRITERIA

The appropriateness of the surveyed papers, and, consequently, the scientific adequacy of this review paper, are also determined by certain inclusion criteria (IC), and exclusion criteria (EC). More precisely, contributions that do not meet the specified EC are disregarded. The IC-related filtering model relates to a logical process based on the following steps.

- Step 1. Abstract-related filtering: irrelevant articles are disregarded considering the information acquired from the abstract, and also based on the keywords. More precisely, articles that fulfill at least 50% of the relevance threshold are considered.
- Step 2. Full text-related filtering: articles that concern only a small part of the scientific scope, as specified by the abstract and the keywords, are disregarded.
- Step 3. Quality analysis-related filtering: the rest of the papers were additionally filtered considering that at least one of the following conditions are unmet:
  <The paper describes a functional solution concerning the automatic bug triaging and management models.> AND <The article fully presents the implemented technical solution.> AND <The article surveys related relevant contributions.> AND <The article presents and assesses the outcomes of the experimental process.>

Furthermore, the inclusion criteria are presented in table 2. Moreover, the exclusion criteria are described in table 3.

The following sections extensively review the large number of articles, which were selected considering the principles of this scientific survey methodology.

III. ARTIFICIAL INTELLIGENCE MODELS FOR BUGS TRIAGING

The relevant scientific literature presents various approaches concerning software bug triaging (SBT). Thus, SBT models are grouped into six fundamental categories relative to the considered AI technologies. The six technological approaches relate to machine learning (ML), information retrieval (IR), social network analysis (SNA), recommender systems (RS), mathematical modeling and optimization (MMO), and deep learning (DL). The features of these approaches are described in Table 4. This section surveys the most relevant scientific contributions identified in each category.

A. BUG TRIAGING MODELS BASED ON MACHINE LEARNING

Machine Learning (ML) models represent the natural solution to implement an automatic software bug triaging (SBT) system. Proper machine learning-based models [2], [3] are frequently used in this respect (Softmax classifier, Support Vector Machine, Multinomial Naive Bayes, K-Nearest Neighbors, J48, Random Forests, Artificial Neural Networks), along with clustering [4] and association rule mining [5]. SBT is regarded as a multiclass, single-label classification problem [6], which considers the software developer as a class. Thus, it is immediately discernible that the proper classification techniques are frequently used relative to machine learning-based bug triaging techniques. The performance metrics, such as accuracy, precision, recall, and F1-measure, are used in order to assess the described approaches, typically on the top 5 or 10 best outcomes. Thus, the accuracy gets as high as 40% - 50%. It is relevant to note that a comparative review concerning a handful of machine learning models for software bug triaging is presented in article [7], and is also conducted by Goyal and Sardana [8]. While other techniques like information retrieval can beat plain ML-based SBTs, they are relatively simple to model, and there are proper libraries, which implement efficient Application Programming Interface (API) support. Nevertheless, the enhancement of the computational performance [9] represents a goal that continues to motivate scientific research efforts to enhance the existing approaches [10], and consequently assess modern approaches, which are based on artificial intelligence models. The surveyed experimental analysis contributions suggest that plain machine learning-based approaches under-perform deep learning-based models, but they are comparable with information retrieval-based models.

B. BUG TRIAGING MODELS BASED ON INFORMATION RETRIEVAL

The general algorithmic process of software bug triaging may also be perceived as a problem of information retrieval (IR), which presumes that the relevant data determines that a developer is fetched from the set of software developers relative to the newly created bug reports. Thus, IR-related models, such as LSA (Latent Semantic Analysis), and LSI (Latent Semantic Indexing) [11], [12], are also used together with other relevant models. The accuracy scores generated using IR-related SBT models range from 63.2%

TABLE 4. Features of various bug triaging techniques.

Bug triaging class	Features
ML	Training data and classification
IR	Text similarity and topic modeling
SNA	Developers network and cross repository analysis
RS	An ordered list of developers
MMO	Optimization and computational mathematics
DL	Layered stack of training and classification

to 96%. Moreover, a recall value of 95% is reported by the work described in article [13]. Additionally, TF-IDF (Term Frequency-Inverse Document Frequency) represents the most usual algorithmic model relative to IR-related models for software bug triaging. Several articles describe relevant changes to TF-IDF-related solutions related to the scientific efforts presented in [14] and [15]. These resources consider bug location information, term weighting in TF-IDF, and time metadata concerning the TF-IDF presentation. Additionally, software bug triaging uses both text mining, which is presented in articles [16], [17], [18], [19], and also text similarity models, which are described in articles [20], [20], [21], [22], [23], [24], [25]. The topic is also approached in other related studies.

Moreover, large software repositories determine an interesting scope of scientific research, which is interestingly approached in [26]. Also, the concept of topic modeling is approached in several articles, such as [12], [27], [28], [29], and [30], with a clear emphasis on software bug triaging. Moreover, Latent Dirichlet Allocation (LDA) determines an important probability-related algorithmic approach, which is described in paper [31]. Other similar approaches consider this algorithmic model for automatic software bug triaging, presented in [32] and [33]. Certain papers regard specialized variants concerning topic modeling in connection to automatic software bug triaging. Thus, the multi-feature topic model (MTM) is proposed in article [34]. At the same time, the Entropy Optimized Latent Dirichlet Allocation is described in article [35], along with the Multiple LDA concept proposed in paper [29]. Notably, the maximum generated accuracy through the utilization of topic modeling-based bugs triaging approaches is 98.31%, which is suggested by the work presented in article [36].

The extensive scientific literature that was surveyed suggests that, in a similar fashion to machine learning-based models, information retrieval-based models provide acceptable application programming interface (API) features. Furthermore, the implied algorithmic models are easy to model and implement in a suitable programming language. Despite the obvious conceptual and practical advantages, the relevant approaches related to information retrieval and topic modeling present computational performance issues in certain real-world scenarios, and they may also have difficulties generating proper terms relative to the topics produced by the respective topic modeling approaches. Thus, the topics that are produced by related topic modeling approaches may provide a certain degree of randomness, which may impact the overall data analysis process, as it is suggested by the work that is reported in article [37]. Nevertheless, it is relevant to state that the surveyed scientific literature suggests that the main problem, which should be addressed, is represented by the insufficient level of computational performance that manifests in certain real-world scenarios that are described in article [32]. This may affect the real-time processing of relevant software bug data.

There is a clear similarity between ML-related and IR-related automatic bug triage and management techniques. The fundamental difference is determined by the fact that IR-related approaches essentially relate to the textual data that are stored by the software bug repositories. Similarly to ML-related approaches, algorithmic and software modeling is also relatively easy with IR-related models, which require the least computational and data storage resources to implement automatic software bug triaging systems. The surveyed literature also suggests that these IR-related approaches benefit from consistent support relative to all modern programming languages and application programming interfaces. Some drawbacks, such as computational scalability and real-time software bug triaging, are shared between IR-related and ML-related approaches.

C. BUG TRIAGING MODELS BASED ON SOCIAL NETWORK ANALYSIS

Relative to software bug triaging, social networks designate the developers' network, which is used by the enrolled software developers in order to sustain the implied software systems development processes. The activity of bug resolution implies the existence of particular skills. Nevertheless, developers use third-party or external support sources, such as StackOverflow, or GitHub, which may provide useful technical information. The extraction of useful information from several sources in order to enhance the resolution of software bugs is referred to as Crowdsourcing. Thus, this idea is explored in certain papers, such as [38], [39], and [40].

The relevant technical information, as it is fetched from several repositories, may be integrated together with the identifying data of the software bug repository, which may help identify the software developers with expert skills. This approach is generally designated as cross-repository analysis and is approached in an interesting manner in article [41]. The synergistic combination of crowdsourcing and social networks-related analysis models creates a functional advantage regarding determining the relationship between developers and their technical skills. This generally supports an enhanced bug assignment process to the proper developers, which may support the implied automatic bug management systems. The surveyed scientific literature suggests that this type of approach determines a problematic aggregation and integration of the acquired and existing data, which is particularly derived from the consideration of multiple data sources. The implied problem is studied and reported in

articles [38], [39], and [40], which also analyze the generated and existing relationship graphs. Furthermore, the surveyed literature suggests that automatic software bug triage and management approaches, which are based on software network analysis, are relatively difficult to design due to the implied processing of the graph data structures that model the developer-bug relationships. Consequently, the implied data processing routines require more computational time and resources. Nevertheless, the surveyed contributions claim that this is compensated by the overall enhancement of the bug assignment or reassignment, while further algorithmic and implementational improvements are likely to assure the required computational scalability.

D. BUG TRIAGING MODELS BASED ON RECOMMENDER SYSTEMS

Recommender systems represent a fundamental concept in the scope of machine learning scientific research. Thus, the contributions that are described in articles [23], [43], [44], and [45] suggest that proper developers may be efficiently recommended and assigned to the newly created bug reports. The consideration of recommender systems (RS) mediates the creation of a list of software developers, and a list of the most suitable k developers is generated, according to their assessed technical skills. Thus, certain historical contributions are reported in articles [46], [47], and [48]. The approach that is described in article [48] involves that a ranking of the software developers is created considering using the mechanism of developer prioritization. It is interesting to note that certain performance metrics are described in the surveyed literature. Thus, in article [48], the authors propose Accuracy@K (Acc@K), Precision@K (P@K), and Recall@K (R@K). Here, K designates the most suitable K software developers relative to the list, which was created using the implied recommender system. There is a majority of the articles that were surveyed, which consider the Eclipse and Mozilla Firefox bug trackers, and Recall@10 is obtained at a level of up to 90%.

Including an RS-related algorithmic module mediates the generation of a list of ranked developers, which are sent over to the automatic bug triaging and management components. Consequently, the system generates a higher-quality software developer suggestion, who is available and adequate to fix the respective bugs. Nevertheless, it is important to note that there are certain conceptual and practical problems, which are reported in the surveyed literature. Thus, the issue of cold start is analyzed in article [49], while the performance and data sparsity are approached in paper [50]. Thus, the issue of cold start and data sparsity are determined by insufficient data relative to a certain data item or category, which provokes an overall degradation of the recommender system's computational performance. More precisely, an RS-related bug triaging system may not properly identify the software developer, if sufficient data are not available regarding the potentially suitable developers.

E. BUG TRIAGING MODELS BASED ON MATHEMATICAL MODELING AND OPTIMIZATION

Several mathematical models are relevant for general bug triaging and management processes. Thus, fuzzy sets are assessed and experimentally analyzed in articles [51], [52], [53], [54], and [55]. Moreover, the Knapsack programming was analyzed in paper [55], in connection to bug triaging. Additionally, a bug triaging process, which is based on genetic algorithms models, is proposed in article [56]. It is relevant to mention the probabilistic optimization models that are based on the behaviour of ants, which are approached in paper [57]. Most surveyed contributions pertain to the Eclipse bug tracker, and the implemented models generate a maximum accuracy of 86%. In this case, the researcher focuses on the development of the mathematical models, and also on the specification of the objective functions, which are used to process the bug reports data.

As an example, relative to the fuzzy modeling-related techniques, a fuzzy model of software bugs is necessary, along with the membership functions that are specified between software bug terms and respective developers for bug triaging purposes. Considering the optimization models for software bug triaging, the relevant optimization constraints are specified relative to the number of bugs that are resolved by a certain developer during a given period of time. This is particularly important, as time is an important parameter, especially relative to high priority and security bugs. Consequently, the specified and implemented mathematical model mediates the selection of the proper developer to resolve the processed bug reports.

The software bug triage techniques (SBT), which are based on the optimization concept, imply the adequate mathematical modeling of the SBT problems relative to the terms of the implied optimization model. As an example, an SBT software system that is based on the mechanism of Ant Colony Optimization (ACO) [58], presumes that the bug tossing graphs are generated using the historical bug resolution data. Furthermore, the ants are allowed to circulate through these tossing graphs, with the goal to detect the optimum paths, which determine the suggested developers for the resolution of the reported bugs. Additionally, an ACO-related model is based on certain calibration parameters, such as the number of ants, the number of iterations, the configuration of the developer's network, and the intensity of the ants' pheromone. Several knapsack optimization-related SBT techniques [59], involve that the bug fixing and developer metadata are transformed into the respective knapsacks. The capacity of knapsacks is determined by the time limit, which is allocated for the resolution of the software bugs. Thus, the items and knapsacks determine the number of bugs and developers, respectively.

Genetic algorithms-related optimization techniques [60] for software bug triaging imply that fitness functions are specified relative to the list of words that label and describe bugs. The calculation of similarity scores is performed for clustered centers, and the fitness functions pertain to the

developers' data. The maximum value of the similarity suggests a higher-grade membership in the cluster of developers, which resolve the provided software bugs. Moreover, greedy search-related optimization models [61] for software bug triaging are based on the creation of a search space by considering the data of particular developers, which are available for a specific timeframe in order to resolve the existing bugs. Moreover, concerning the bugs triaging process, the distance functions are utilized to compute the distance to all the available developers, and consequently assign the open bugs to the developers that are featured by the shortest distance.

The surveyed articles suggest that the mathematical-based models mediate efficiently identifying the necessary constraints. The computational performance of bug triaging represents the main issues of these techniques, and the relevant problematic is approached in the articles [41], [62], and [63]. The efficient real-time behaviour of the implemented models may be affected by particular situations, such as the removal of the developer from the respective project, or the possibility for the developer to leave the company.

The fuzzy logic-related bug triaging models imply that membership functions are specified to represent relationships between the newly reported software bugs, and the proper software developers. The relevant problematic is approached in articles [60], [61], and [63]. The creation of new software bugs implies that the similarity between the terms of newly reported bugs relative to the existing bug terms is computed. Following, fuzzy logic allows for the membership values to be computed in connection to the newly created bugs, to assign the proper developers. Thus, a greater value of the membership indicates a higher possibility of properly resolving the bugs. It is relevant to note that several recent studies, such as [60], [62], and [63], propose fuzzy logic models relative to multi-criteria decision-making and analysis. The surveyed contributions suggest that these are used in the realm of efficient software bug triaging systems.

The main advantage of fuzzy logic models is the implied simplicity, and also the reasonable amount of computation resources that are required [63]. Considering a varied set of skilled software developers, the consideration of fuzzy membership mediates the efficient selection of the optimal developer. The disadvantage of SBT approaches that are based on fuzzy logic models is represented by the relatively difficult development of the most relevant and logical membership function. Moreover, the extensive scientific survey that was conducted suggests that only a few relevant fuzzy logic-related models are reported. Consequently, the degree of scientific generality is reduced, in this case, and the possibility of considering this type of mathematical and algorithmic model in all real-world use cases is problematic.

F. BUG TRIAGING MODELS BASED ON DEEP LEARNING

Deep Learning represents a machine learning technique, which considers the natural "learn by example" strategy. Therefore, its algorithmic and computational apparatus may also be applied to software bug triaging (SBT). Thus, deep learning techniques, which are based on Convolutional Neural Networks (CNN) are described in articles [2], [64], and [65]. Moreover, models that are based on Recurrent Neural Networks (RNN) are discussed in articles [2] and [5]. These papers report contributions that are consistently used relative to software bug triaging. It is relevant to note that Convolutional Neural Networks and Recurrent Neural Networks represent popular approaches relative to deep learning models, which are considered to implement software bug triaging systems. The essential difference between CNN and RNN is determined by the densely connected feed-forward network, which determines CNN, while RNN considers a feed-backward network, which processes the data from the previous iteration in order to improve the weights. Thus, bug summary and description represent significant attributes, which are usually processed in general bug triaging processes. These are text-based attributes. Consequently, word embedding models, such as Word2Vec and Glove, are frequently used to implement software bug triagers. The surveyed articles report experiments that are conducted on various datasets, and the obtained accuracy values belong to the range 57% to 87% relative to the assessed DL-related models. The Word2Vec approach [66] is a word embedding technique that is frequently considered in the related scientific literature, as compared to the Glove model [67].

The considered CNN models are based on convolutional layers and pooling layers. The rationale behind these layers is represented by feature extraction. Classification is conducted during the last stage. Thus, relative to SBT, each software developer determines a category. Generally, DL-related models offer consistent computational performance, scalability, and learning rates, which is the time required for the model's training, relative to other AI-related models. Nevertheless, computational time and the mandatory features of the computing infrastructure represent key challenges. Thus, DL-related models require greater training time relative to traditional machine learning or information retrieval-related approaches [68]. This computational behaviour is determined by the multiple data processing layers, which have to be visited [69]. Therefore, organizations that cannot afford the more expensive computational infrastructures, may be forced to consider alternative solutions.

Regular multilayered CNN architectures, such as the one that is presented in article [64], consider DL-related software architectures for SBT, which are structured according to word vector representation layers, convolutional layer, pooling layer, and activation functions, which are necessary in order to aggregate the generated output values. The convolution layer conducts the training of the data samples, and also the training of the input data samples. The functional relevance of the pooling layer is to determine and extract the data samples from the feature space, which are relevant to the processed task. Moreover, it is relevant to note that Max pooling techniques, which are sample-based discretization processes, are

considered and proposed in the surveyed studies, considering that they provide consistent computational performance.

The surveyed papers also describe personalized deep learning models, such as graph recurrent convolutional neural networks (GRCNN) models [68], and reinforcement deep learning (RDL) models [69]. The graph recurrent convolutional neural network (GRCNN) model generates F1 scores of 86.74%, and 75.64% relative to the Eclipse and Mozilla projects bug trackers. Furthermore, the surveyed deep reinforcement learning approaches generate 52%, 54%, 68%, and 78% top-5 accuracy values considering the OpenOffice, NetBeans, Mozilla, and Eclipse bug trackers.

IV. PERFORMANCE EVALUATIONS

Performance determines a significant problematic for the assessment of any bug triaging approach, which was naturally approached by this extended study. Consequently, the relevant performance metrics suggest the real-world appropriateness of the proposed algorithmic approaches. As an example, relative to a classification-related bug management and triaging model, accuracy (ACC), precision (P), recall (R), and F-measure (F1) are defined through a confusion matrix [70].

The number of correct predictions divided by the total number of predictions makes up the classification accuracy measure, which is possibly the most straightforward to use and implement. The formula for the accuracy is presented in equation (1), where TP, TN, FP, and FN represent the true positives, true negatives, false positives, and false negatives, respectively.

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

A multitude of studies consider this metric to validate the model [31], [94], [95], [96], [97], [98]. However, some of the mentioned studies also use other metrics for parallel comparisons, and/or validations, precisely because the accuracy itself presents some downfalls. Sometimes, it may be preferable to choose a model with lower accuracy, if it has a stronger ability to anticipate the outcome of the situation. For instance, if the real-world use case presents a significant class imbalance, a model can predict the value of the majority class for all predictions, and obtain a high level of classification accuracy. Nevertheless, the model may not be applicable to the problem at hand.

Therefore, the next time a classification problem is presented, one should be careful not to merely choose accuracy as a metric, and start creating the model straight away. Naturally, experimenting with the model is enticing, but it's necessary to take some time to understand the types of issues that should be solved, and the most suitable metrics. After clearing that up, one may be confident that the model that is created will be appropriate for the task at hand. This paradox is known as the Accuracy Paradox, and for such issues, extra measurements are needed to evaluate a classifier.

Further on, precision represents the ratio of relevant software developers relative to all the developers determined by the respective machine learning algorithm. Continued, the recall represents the ratio of relevant software developers relative to all the relevant software developers determined by the machine learning algorithm. The formulas for these metrics are presented in equation (2).

$$P = \frac{TP}{TP + FP}, R = \frac{TP}{TP + FN} \tag{2}$$

Both of these values should ideally be as high as they can be. That might not be possible, though. Precision will fall off when recall rises, and vice versa. Therefore, one must choose which indicators are particularly relevant while training the machine learning model. One crucial and time-consuming aspect of software maintenance is bug triaging. The model may be required not to provide a false result if it was used for an organization that wanted to determine whether the programmer is a good fit or not for an urgent bug. As a result, one would rather label suitable programmers as inappropriate programmers (false negative) than suitable programmers as inappropriate (false positive). In other words, a false negative is preferable to a false positive (recall).

Additionally, F1-measure (3) is a metric that combines precision and recall. The F1 measure is the best option if it is required to choose a model based on a balance between precision and recall.

$$F1 = \frac{2 \times P \times R}{P + R} \tag{3}$$

It is interesting to note that these metrics may be computed directly, or they can be computed using a confusion matrix [99].

In addition, other metrics have proved valuable for measuring the quality of results, metrics that are perhaps less well-known but of certain interest. For example, the authors of article [71] conduct a relatively consistent analysis of existing bug triaging techniques. Thus, the article surveys 74 studies, which are related to software bug triaging, and it includes a presentation of the metrics that were used in order to evaluate the analyzed software bug triaging models. Thus, the article emphasizes the metrics, which are considered in order to rank the available software developers. These metrics are top-K accuracy, Precision@K (P@K), Recall@K (R@K), mean reciprocal rank (MRR), and mean average precision (mAP). Here, K designates the number of recommended or available software developers. The top-K accuracy relates to the number of effective developers that have been tasked with the resolution of existing bugs, out of the K developers that are recommended. This is computed as the ratio of recommended developers divided by the total number of developers. Although there are numerous studies that use this metric, such as [101] and [102], the article [100] presents an experimental analysis concerning the trade-offs in top-k classification accuracies, and the losses related to the problematic of deep learning.

Furthermore, P@K is calculated as the ratio between the relevant (effective) recommended developers and the total number of recommended developers. Moreover, R@K is calculated as the ratio between the relevant (effective) recommended developers and the total number of relevant developers. Thus, P@K and R@K are computed according to the formulae described in equation (4), F1@K is represented in equation (5), and the average precision (AP) is described in equation (6).

$$P@K = \frac{TP@K}{TP@K + FP@K}, \quad R@K = \frac{TP@K}{TP@K + FN@K} \tag{4}$$

$$F1@K = 2 \times \frac{P@K \times R@K}{P@K + R@K} \tag{5}$$

$$AP = \Sigma(Recall@K - Recall@(K-1)) \times Precision@K \tag{6}$$

Furthermore, DCG@K and NDCG@K represent two additional metrics, which are considered in order to evaluate recommender systems. DCG is an acronym for Discounted Cumulative Gain, and NDCG is the acronym for Normalized Discounted Cumulative Gain. These metrics are considered in order to assess the quality of the developers ranking during the software bug triaging process. The calculation of DCG@K and NDCG@K is conducted using the formulae described in equation (7). Here, rel_i represents the relevance of the i^{th} developer recommendation, while the IDCG determines the Discounted Cumulative Gain in ideal conditions.

$$DCG@K = \sum_{i=1}^{K} \frac{2^{rel_i} - 1}{\log_2(i+1)}, \quad NDCG@K = \frac{DCG@K}{IDCG@K} \tag{7}$$

The surveyed articles suggest that, although contributions like [71] imply that mean average precision mAP represents the optimal quantitative assessment metric related to software bug triaging, the quantitative and qualitative analysis is also determined by the type of AI-related model, which is selected for the bug management and triaging process. Thus, if the algorithmic model is based on recommender system techniques, then DCG@K and NDCG@K are the optimal metrics to conduct the performance evaluation [48].

Furthermore, relative to machine learning and information retrieval-oriented models, accuracy, precision, recall, and F-measure are the recommended performance evaluation metrics [70].

V. SCIENTIFIC CHALLENGES AND OPEN RESEARCH QUESTIONS

The types of metrics, algorithmic and numerical approaches, which were surveyed in the previous sections, suggest that although numerous techniques exist for the design and implementation of automatic bug triaging and management techniques, there are also conceptual and practical challenges, which are discussed in this section, along with the proper suggested solutions. This section considers several categories of conceptual and real-world problems, which are relevant for future research efforts. Additionally, the extensive surveyed literature determines significant open research questions, which are discussed.

A. GENERATION OF DEVELOPERS VOCABULARY

The generation of a precise software developers' vocabulary represents a fundamental activity relative to the software bug triaging process. Thus, the design and development of software bug triaging techniques imply that the defining bug data and technical abilities of the software developers are taken into account during the automatic bug assignment process. Nevertheless, apart from the technical abilities of the involved software developers, recent articles, such as [72] and [73], consider other relevant aspects. Thus, there are certain software engineers or developers that can be classified as ''experts'', and may be consequently considered during the optimization of the developer vocabulary generation process, which relates to upcoming software bug triaging techniques.

B. DATA REDUCTION MODELS

The contribution that is described in article [74] reports an approach for filtering and eliminating invalid bugs from the processed bug trackers. This optimizes the amount of bug triaging data, and consequently ameliorates the necessary time and energy, which are necessary for the proper management and effective bug triaging processes. Moreover, the paper defines four feature groups, which are the experience of the bug reporter, the related collaboration network, the degree of technical completeness, and the defining bug text. Consequently, a Random Forest classifier was designed to assign the invalid bug to the proper category. Additionally, the paper has the merit to conduct a rather useful overview on interesting approaches for software bug triaging. It is also relevant to mention the contribution reported in paper [75], which aims to detect and properly manage the non-reproducible bugs, in which identification and handling of respective bugs are conducted relative to the processed bug tracker. This may sensibly reduce the processing time of non-reproducible bugs, which are generally difficult to manage in an adequate manner.

C. MODELS REGARDING BUG PRIORITIZATION

The contribution that is described in article [40] concerns the prioritization of security bugs, which is regarded as an essential process in the scope of software development. Thus, the implementation of security patches determines a mandatory effort in the process of software development, as it implements the required security mechanisms. The reported work is based on reducing the size of the training data, which is logically enriched and further improved with a transfer learning model. The article demonstrates that this type of

approach may be considered relative to the relevant software bug triaging processes.

D. MODELS RELATED TO FEATURES SELECTION AND AGGREGATION

Feature selection, aggregation, and ranking [76] determines a task that is usually performed during the preprocessing phase of the software bug triaging process. Thus, article [77] describes an ontology for an efficient location-based knowledge extraction relative to the implementation phase of software development. The relevant ontology may be considered for bug triaging during the subsequent phases of the software bug triaging process. Furthermore, the generation of user summaries, which are useful to recommend similar projects during the software development is proposed in article [27], which can be aggregated and used together with other algorithmic model to implement the software bug triaging process. Additionally, another contribution was reported in article [78], which described an approach for the generation of a text summary. This is used to assess the opportunity to escalate a ticket, to automatically generate the title and content of the ticket, and also to assign the ticket to a properly available developer. This approach brings obvious advantages to the bug triaging process. Furthermore, aside from the classical feature selection models, feature enhancement is also conducted relative to software bug triaging in article [38], which also presents further useful features and metrics relative to the software bug triaging process.

E. RELEVANT METRICS

The comprehensive contributions that are reported in articles [51], [79], and [80] discuss four quality metrics, which influence the quality and computational efficiency of bug triaging processes. Thus, the relevant quality metrics are complexity, cohesion, inheritance, and coupling, which were analytically assessed in the mentioned articles. The conceptual and real-world analysis that determines this scientific survey implies that future research efforts should concentrate on other software quality features, such as maintainability, reusability, and refactoring.

F. ANALYSIS OF LARGE BUG DATASETS

The majority of surveyed papers [77] consider public bug trackers, such as Mozilla, Apache, Google, and Git, in order to assess the validity and computational efficiency of the proposed software bug triaging models. Consequently, we suggest that the reference performance assessment datasets should be diversified, while the automatic software bug triaging models should be algorithmically enriched with up-to-date techniques, such as big data analytics.

G. MODELS RELATED TO THE STUDY OF NETWORKED AND GRAPH STRUCTURES

The authors of article [38] suggest the consideration of semantic, multiplex, and multimode networks, which are useful to model the relationships between the components of bug items components. This supports the design and development of future enhanced software bug triaging models. Moreover, paper [81] describes the creation of a visual analysis tool that is related to a Directed Acyclic Graph (DAG), which may be considered to model the logical relationship between the involved developers. Article [82] reports a model that is called iTriage, which creates a sequential model that supports the processing of textual features, and also the relevant bugs tossing sequences. The relevant conceptual and practical suggestions should be considered for the creation of future enhanced software bug triaging systems, which properly implement the bug assignment and management routines. It is relevant to note that article [41] proposes a logically related approach that considers the GitHub bugs tracking repository, which presumes that semantically related issues are properly recorded in the analyzed GitHub bugs dataset. The reported scientific developments may be considered to efficiently identify similar bugs, and consequently determine relevant information for the resolution of critical and high-priority bugs. Furthermore, the logical mechanism of bug dependency, and also dependency graphs may also be considered for the enhancement of software bug triaging routines, as it is demonstrated in article [83].

H. MODELS THAT CONSIDER CROSS REPOSITORY ANALYSES

Several papers consider the experimental and performance analysis mechanism of cross-repository analysis. Thus, paper [84] proposes an automatic commit message generation model, which pertains to version control systems. Thus, the text of the commit message was automatically generated considering previous software commit data, which was stored in a comprehensive repository. Moreover, a further relevant model is described in article [85], which proposed a customized recommender system that is related to the processing of open-source repositories. Additionally, an interest measurement mechanism, which is based on the processing of GitHub data, is specified relative to the developers that work on a project that is linked, considering the common technical abilities of relevant software developers. These and similar existing contributions propose conceptually relevant algorithmic models, which should be considered in order to design future software bug triaging models.

I. MODELS THAT CONSIDER DEEP LEARNING TECHNIQUES

Deep Learning (DL) models are created and reported by several contributions that pertain to software bug triaging. Thus, the Deep Learning-related Convolutional Neural Networks (CNN) model is utilized, together with the Word2vec word embedding model, in order to implement the bug triaging processes [64]. Apart from the described algorithmic model, DL determines a consistent algorithmic and functional apparatus, which can be used in future

software bug triaging approaches. Variants of CNN, such as bio-inspired spiking CNN (SCNN), seem to provide superior computational performance than classical CNN networks. Consequently, SCNN networks can be considered in order to implement relevant computational routines for software bug triaging [46]. Moreover, article [44] analyzes the technological profile recommendations relative to various document embedding models, while paper [86] addresses the problematic of the software developers' skills prediction considering a multi-label classification of available resumes data, which is based on the usage of CNN networks with model predictions.

J. CONTEMPORARY MACHINE LEARNING AND BIG DATA APPROACHES

Contemporary machine learning models like ensemble learning and transfer learning are usually coupled with big data analytics methods, which can be considered in order to further improve the software bug triaging approaches. Thus, the model of label prediction in bug repositories is studied in [17], while the automatic identification of technically skilled software engineers are discussed, designed, and implemented in paper [73]. Moreover, various software fault prediction models are explored in article [87], while an automated issue assignment model is proposed by the contribution reported in paper [88]. Furthermore, it is interesting to note that relevant blockchain-related models are also analyzed and described in article [62], with a clear emphasis on the software bug triaging processes.

K. EFFICIENT MANAGEMENT OF IMBALANCED DATASETS

The subject of imbalanced classes represents a theoretical and practical problem in the general field of artificial intelligence. Relative to software bug triaging, class imbalancing manifests when the bug items are not assigned to the developers in a balanced manner. More precisely, the abnormal situation occurs when a few software engineers fix a majority of the reported bugs. This situation may determine a class imbalancing in the training dataset, which is considered relative to the bug triaging process. Thus, several papers explore the relevant problematic from several perspectives, including the defect prediction algorithmic models [89], [90], [91], [92]. The surveyed papers also approach the topics of bug fixability prediction [93], and also the general problem of bugs classification [23], [24]. Nevertheless, there are only a few systematic studies concerning the relevant research aspects and open questions in the scope of automatic bug triaging and management. Consequently, this survey contributes to filling a gap, as it systematically approaches the essential aspects and existing contributions, which are related to software bug triaging and management. The issue of imbalanced datasets may be further approached using deep learning and soft computing techniques, which combine artificial intelligence models, and also natural selection approaches.

VI. RESEARCH QUESTIONS AND DEGREE OF ACCOMPLISHMENT

Artificial intelligence is perceived and practically demonstrated as a novel approach to the problematic of design and implementation of precise and automated software bug triaging systems. The vast array of analyzed scientific contributions suggests that artificial intelligence and machine learning can be considered effective tools in order to design and implement accurate and computationally efficient automated software bug triaging systems. Nevertheless, the continuous technological developments, and also certain conceptual and practical issues, which were addressed in the previous sections, suggest that there is still enough room for further optimizations and functional extensions. Consequently, the following paragraphs discuss the three fundamental research questions, which guided this scientific survey effort and also analyze the degree of scientific achievement of this paper.

A. RQ1: IS AI CAPABLE TO EFFECTIVELY ENHANCE AND AUTOMATE SOFTWARE BUG TRIAGING?

Regarding the thorough scientific review that was conducted, several pertinent remarks can be made concerning the consideration of AI-related models to design and implement software bug triaging approaches. Thus, the discussion considers three perspectives relative to software bug triaging systems, which this section considers. The first perspective pertains to AI-related models for bug triaging. The identified scientific contributions suggest that machine learning models have been a popular choice during the past ten years for software bug triaging. Following, as this paper already demonstrated, information retrieval-related models were considered, chronologically followed by approaches that are based on recommender systems. Furthermore, contemporary studies have started to consistently rely on deep learning-related solutions. The significant changes regarding the conceptual and practical paradigms are easily discernible considering the scientific literature, which this article surveys. Thus, deep learning-related approaches have been demonstrated to enhance related software bug triaging routines, both considering the accuracy, as it is measured by various metrics and also the computational efficiency. Nevertheless, future developments are expected to address the remaining issues, which are also connected to the accuracy and computational efficiency of relevant bug triaging processes. These currently hamper the consideration of some deep learning-related approaches in the context of certain large or structurally complex real-world software bug triaging use cases.

It is immediate to observe that automated software bug triaging will minimize the time that is necessary for software development processes, and concomitantly, the software development costs will be reduced. Consequently, the economical viability of the software development projects and companies will be strengthened. This assertion is comprehensively approached and demonstrated in the surveyed

literature. Therefore, it can be inferred that AI-related models are not only relevant, but essential for the implementation of automated software bug triaging systems.

B. RQ2: IDENTIFICATION OF SIGNIFICANT PERFORMANCE PARAMETERS AND METRICS CONSIDERED BY EXISTING CONTRIBUTIONS

The second perspective relates to the comprehensive analysis of performance evaluation techniques, parameters, and metrics, which are considered for AI-related software bug triaging models. The relevance of the second analysis perspective is determined by the fact that the selection of performance parameters should be carefully conducted, in accordance with the technological model, which is used to design and implement the relevant software bug triaging techniques. More precisely, relative to a bug triaging technique that is based on classification algorithms, accuracy, precision, recall, and F-measure are specified through the consideration of a confusion matrix. Furthermore, developer ranking algorithms consider performance metrics like top-K accuracy, Precision@K, Recall@K, mean reciprocal rank, and mean average precision, which is fully covered in this paper. Here, K designates the number of recommended software developers. Additionally, Discounted Cumulative Gain, and Normalized Discounted Cumulative Gain, which have already been covered, represent significant performance metrics, which are used in order to evaluate software bug triaging approaches, which are based on recommender systems.

Although certain contributions, such as the one that is reported in article [71], suggest that the mean average precision may be the optimal performance evaluation metric relative to the relevant software bug triaging processes, the assessment should consider the specific AI model that is used to implement the bug triaging process. As an example, in the case of bug triagers that are based on recommender systems, then Discounted Cumulative Gain, and Normalized Discounted Cumulative Gain was reported to be the proper performance evaluation metrics. Furthermore, in the case of bug triagers that are based on machine learning and information retrieval models, accuracy, precision, recall, and F-measure were reported to be the suitable metrics, both considering their validity for the intended type of measurement and their computational efficiency. It can be asserted that this scientific survey reached the goal to determine and comparatively analyze the most suitable performance parameters and metrics, in the context of software bug triaging.

C. RQ3: OPEN SCIENTIFIC PROBLEMS AND POSSIBLE FUTURE DEVELOPMENTS TO ENHANCE SOFTWARE BUG TRIAGING PROCESSES

The third perspective is defined by the open scientific research problems, which could define possible future research pathways in the scope of software bug triaging. The open research problems are presented in the previous section, along with clear indications regarding the possible future research pathways. Thus, eleven categories concerning the future possible research subjects are defined. These categories pertain to data reduction techniques, developers' vocabulary generation, feature selection and aggregation, bug prioritization, performance parameters and metrics, exploration of large bug data sets (repositories), networks and graph-related models, advanced deep learning-related models, cross repository analysis, contemporary machine learning and big data-related approaches, and the proper management of imbalanced datasets. The comprehensive scientific literature that was surveyed suggests that, in spite of the existing functional approaches that were proposed for automatic software bug triaging, there are numerous conceptual and practical aspects that need to be further addressed. Considering the semantics and perspectives that are determined by the comprehensive literature that was surveyed, it is possible to objectively assert that the most relevant open scientific problems, research gaps, and future research pathways were properly identified and discussed in this paper.

VII. CONCLUSION

Software bug triaging determines a significant research scope, considering its conceptual and real-world implications. Consequently, the relevant scientific literature encompasses various contributions, which have been thoroughly surveyed and presented. Thus, AI-related models are generally considered in order to implement automatic software bug triaging and management systems. The present article describes an extended survey, which systematically analyzes the most relevant contributions that are related to software bug triaging. The scientific survey is structured according to a systematic approach, which selects the relevant existing articles based on the principles of the PRISMA scientific review system. Consequently, this paper comprehensively surveys, classifies, and analyzes relevant software bug triaging and management approaches, which are reported in the existing literature. The identified papers are analytically and comparatively evaluated, and three fundamental research questions are defined and discussed. Consequently, a three-dimensional evaluation and comparative analysis are conducted relative to the identified software bug triaging contributions, which consider the defined research questions. Consequently, an evaluation of the identified performance parameters and metrics is conducted and included, as a separate section, in this paper.

Furthermore, relevant research questions, which are currently unsatisfactorily approached, are presented and analyzed, and possible future research and practical trends related to software bug triaging are discussed. Considering each surveyed bug triaging approach, the identified advantages and problems are discussed and evaluated, both considering their conceptual relevance and their importance for real-world software development processes. The relevance of the problematic approached in this survey paper is

further justified by the economic implications of the implied bug triaging and management processes relative to the overall software development efforts. Thus, the consideration of algorithmic models that relate to artificial intelligence and machine learning has essentially changed the paradigm of software bug triaging and management. The surveyed approaches significantly enhance the relevant software bug triaging and management processes, and the software development times and costs are consistently reduced. Nevertheless, several conceptual and practical issues remain, which are thoroughly presented. Therefore, future research efforts should carefully approach the remaining problems, in order to fully establish the automatic software bug triaging and management as an accurate and computationally efficient practical solution.

REFERENCES

[1] D. Moher, A. Liberati, J. Tetzlaff, and D. G. Altman, "Preferred reporting items for systematic reviews and meta-analyses: The PRISMA statement," *Ann. Internal Med.*, vol. 151, no. 4, pp. 264–269, 2009.

[2] S. Mani, A. Sankaran, and R. Aralikatte, "DeepTriage: Exploring the effectiveness of deep learning for bug triaging," in *Proc. ACM India Joint Int. Conf. Data Sci. Manage. Data*, Jan. 2019, pp. 171–179.

[3] H. Mohsin and C. Shi, "SPBC: A self-paced learning model for bug classification from historical repositories of open-source software," *Expert Syst. Appl.*, vol. 167, Apr. 2021, Art. no. 113808.

[4] J. A. Nasir, O. S. Khan, and I. Varlamis, "Fake news detection: A hybrid CNN-RNN based deep learning approach," *Int. J. Inf. Manage. Data Insights*, vol. 1, no. 1, Apr. 2021, Art. no. 100007.

[5] U. Koc, S. Wei, J. S. Foster, M. Carpuat, and A. A. Porter, "An empirical assessment of machine learning approaches for triaging reports of a Java static analysis tool," in *Proc. 12th IEEE Conf. Softw. Test., Validation Verification (ICST)*, Apr. 2019, pp. 288–299.

[6] M. Sharma, A. Tandon, M. Kumari, and V. B. Singh, "Reduction of redundant rules in association rule mining-based bug assignment," *Int. J. Rel., Qual. Saf. Eng.*, vol. 24, no. 6, Dec. 2017, Art. no. 1740005.

[7] G. Murphy and D. Cubranic, "Automatic bug triage using text categorization," in *Proc. 16th Int. Conf. Softw. Eng. Knowl. Eng.*, 2004, pp. 1–6.

[8] A. Goyal and N. Sardana, "Machine learning or information retrieval techniques for bug triaging: Which is better?" *E-Informatica Softw. Eng. J.*, vol. 11, no. 1, pp. 117–141, 2017.

[9] S. N. Ahsan, J. Ferzund, and F. Wotawa, "Automatic software bug triage system (BTS) based on latent semantic indexing and support vector machine," in *Proc. 4th Int. Conf. Softw. Eng. Adv.*, Sep. 2009, pp. 216–221.

[10] W. Wu, W. Zhang, Y. Yang, and Q. Wang, "DREX: Developer recommendation with K-nearest-neighbor search and expertise ranking," in *Proc. 18th Asia–Pacific Softw. Eng. Conf.*, Dec. 2011, pp. 389–396.

[11] H. Kagdi, M. Gethers, D. Poshyvanyk, and M. Hammad, "Assigning change requests to software developers," *J. Softw., Evol. Process*, vol. 24, no. 1, pp. 3–33, Jan. 2012.

[12] S. Banerjee, Z. Syed, J. Helmick, M. Culp, K. Ryan, and B. Cukic, "Automated triaging of very large bug repositories," *Inf. Softw. Technol.*, vol. 89, pp. 1–13, Sep. 2017.

[13] R. Shokripour, J. Anvik, Z. M. Kasirun, and S. Zamani, "Why so complicated? Simple term filtering and weighting for location-based bug report assignment recommendation," in *Proc. 10th Work. Conf. Mining Softw. Repositories (MSR)*, May 2013, pp. 2–11.

[14] R. Shokripour, J. Anvik, Z. M. Kasirun, and S. Zamani, "Improving automatic bug assignment using time-metadata in term-weighting," *IET Softw.*, vol. 8, no. 6, pp. 269–278, Dec. 2014.

[15] R. Shokripour, J. Anvik, Z. M. Kasirun, and S. Zamani, "A time-based approach to automatic bug report assignment," *J. Syst. Softw.*, vol. 102, pp. 109–122, Apr. 2015.

[16] M. Alenezi, K. Magel, and S. Banitaan, "Efficient bug triaging using text mining," *J. Softw.*, vol. 8, no. 9, pp. 2185–2190, Sep. 2013.

[17] J. M. Alonso-Abad, C. López-Nozal, J. M. Maudes-Raedo, and R. Marticorena-Sánchez, "Label prediction on issue tracking systems using text mining," *Prog. Artif. Intell.*, vol. 8, no. 3, pp. 325–342, Sep. 2019.

[18] P. Ardimento, N. Boffoli, and C. Mele, "A text-based regression approach to predict bug-fix time," in *Complex Pattern Mining*. 2020, doi: 10.1007/978-3-030-36617-9_5.

[19] A. Hindle, A. Alipour, and E. Stroulia, "A contextual approach towards more accurate duplicate bug report detection and ranking," *Empirical Softw. Eng.*, vol. 21, no. 2, pp. 368–410, Apr. 2016.

[20] A. Kaur and S. G. Jindal, "Text analytics based severity prediction of software bugs for apache projects," *Int. J. Syst. Assurance Eng. Manage.*, vol. 10, no. 4, pp. 765–782, Aug. 2019.

[21] L. Chen, X. Wang, and C. Liu, "Improving bug assignment with bug tossing graphs and bug similarities," in *Proc. Int. Conf. Biomed. Eng. Comput. Sci.*, 2011, pp. 421–427, doi: 10.1109/ICBECS.2010.5462287.

[22] L. Chen, X. Wang, and C. Liu, "An approach to improving bug assignment with bug tossing graphs and bug similarities," *J. Softw.*, vol. 6, no. 3, pp. 421–427, Mar. 2011.

[23] M. Chen, D. Hu, T. Wang, J. Long, G. Yin, Y. Yu, and Y. Zhang, "Using document embedding techniques for similar bug reports recommendation," in *Proc. IEEE 9th Int. Conf. Softw. Eng. Service Sci. (ICSESS)*, Nov. 2018, pp. 811–814, doi: 10.1109/ICSESS.2018.8663849.

[24] R. Chen, S.-K. Guo, X.-Z. Wang, and T.-L. Zhang, "Fusion of multi-RSMOTE with fuzzy integral to classify bug reports with an imbalanced distribution," *IEEE Trans. Fuzzy Syst.*, vol. 27, no. 12, pp. 2406–2420, Dec. 2019.

[25] D. Hu, M. Chen, T. Wang, J. Chang, G. Yin, Y. Yu, and Y. Zhang, "Recommending similar bug reports: A novel approach using document embedding model," in *Proc. 25th Asia–Pacific Softw. Eng. Conf. (APSEC)*, Dec. 2018, pp. 725–726.

[26] J. Jiang, D. Lo, J. Zheng, X. Xia, Y. Yang, and L. Zhang, "Who should make decision on this pull request? Analyzing time-decaying relationships and file similarities for integrator prediction," *J. Syst. Softw.*, vol. 154, pp. 196–210, Aug. 2019.

[27] M. R. Resketi, H. Motameni, H. Nematzadeh, and E. Akbari, "Automatic summarising of user stories in order to be reused in future similar projects," *IET Softw.*, vol. 14, no. 6, pp. 711–723, Dec. 2020.

[28] J.-W. Park, M.-W. Lee, J. Kim, S.-W. Hwang, and S. Kim, "CosTriage: A cost-aware triage algorithm for bug reporting systems," in *Proc. Nat. Conf. Artif. Intell.*, 2011, p. 139.

[29] D.-G. Lee and Y.-S. Seo, "Improving bug report triage performance using artificial intelligence based document generation model," *Hum.-Centric Comput. Inf. Sci.*, vol. 10, no. 1, Dec. 2020, Art. no. 26.

[30] T. S. Roopa, Y. Purna, and C. Krish, "A novel approach for bug triaging with specialized topic model," *Int. J. Innov. Technol. Exploring Eng.*, vol. 8, no. 7, pp. 1032–1038, 2019.

[31] X. Xia, D. Lo, Y. Ding, J. M. Al-Kofahi, T. N. Nguyen, and X. Wang, "Improving automated bug triaging with specialized topic model," *IEEE Trans. Softw. Eng.*, vol. 43, no. 3, pp. 272–297, Mar. 2017, doi: 10.1109/TSE.2016.2576454.

[32] X. Xie, W. Zhang, Y. Yang, and Q. Wang, "DRETOM: Developer recommendation based on topic models for bug resolution," in *Proc. 8th Int. Conf. Predictive Models Softw. Eng.*, Sep. 2012, pp. 19–28.

[33] W. Zhang, S. Wang, and Q. Wang, "BAHA: A novel approach to automatic bug report assignment with topic modeling and heterogeneous network analysis," *Chin. J. Electron.*, vol. 25, no. 6, pp. 1011–1018, Nov. 2016.

[34] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Jan. 2003.

[35] W. Zhang, Y. Cui, and T. Yoshida, "En-LDA: An novel approach to automatic bug report assignment with entropy optimized latent Dirichlet allocation," *Entropy*, vol. 19, no. 5, p. 173, Apr. 2017.

[36] G. Brookes and T. McEnery, "The utility of topic modelling for discourse studies: A critical evaluation," *Discourse Stud.*, vol. 21, no. 1, pp. 3–21, Feb. 2019.

[37] D. Matter, A. Kuhn, and O. Nierstrasz, "Assigning bug reports using a vocabulary-based expertise model of developers," in *Proc. 6th IEEE Int. Work. Conf. Mining Softw. Repositories*, May 2009, pp. 131–140.

[38] I. Alazzam, A. Aleroud, Z. Al Latifah, and G. Karabatis, "Automatic bug triage in software systems using graph neighborhood relations for feature augmentation," *IEEE Trans. Computat. Social Syst.*, vol. 7, no. 5, pp. 1288–1303, Oct. 2020.

[39] J. Xuan, H. Jiang, H. Zhang, and Z. Ren, "Developer recommendation on bug commenting: A ranking approach for the developer crowd," *Sci. China Inf. Sci.*, vol. 60, no. 7, Jul. 2017, Art. no. 072105.

[40] S. Mostafa, B. Findley, N. Meng, and X. Wang, "Sais: Self-adaptive identification of security bug reports," *IEEE Trans. Depend. Sec. Comput.*, vol. 18, no. 4, pp. 1779–1792, Jul. 2021.

[41] W. Zhang, S. Wang, and Q. Wang, "KSAP: An approach to bug report assignment using KNN search and heterogeneous proximity," *Inf. Softw. Technol.*, vol. 70, pp. 68–84, Feb. 2016.

[42] Y. Zhang, Y. Wu, T. Wang, and H. Wang, "A novel approach for recommending semantically linkable issues in GitHub projects," *Sci. China Inf. Sci.*, vol. 62, no. 9, pp. 1–3, Sep. 2019.

[43] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage: Recommenders for development-oriented decisions," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 3, pp. 1–35, Aug. 2011.

[44] P. Chamoso, G. Hernández, A. González-Briones, and F. J. García-Peñalvo, "Recommendation of technological profiles to collaborate in software projects using document embeddings," *Neural Comput. Appl.*, vol. 34, no. 11, pp. 8423–8430, Jun. 2022.

[45] A. Ray, M. H. Kolekar, R. Balasubramanian, and A. Hafiane, "Transfer learning enhanced vision-based human activity recognition: A decade-long analysis," *Int. J. Inf. Manage. Data Insights*, vol. 3, no. 1, Apr. 2023, Art. no. 100142.

[46] S. F. A. Zaidi, F. M. Awan, M. Lee, H. Woo, and C.-G. Lee, "Applying convolutional neural networks with different word representation techniques to recommend bug fixers," *IEEE Access*, vol. 8, pp. 213729–213747, 2020.

[47] J. Xuan, H. Jiang, Z. Ren, and W. Zou, "Developer prioritization in bug repositories," in *Proc. 34th Int. Conf. Softw. Eng. (ICSE)*, Jun. 2012, pp. 25–35.

[48] G. Shani and A. Gunawardana, "Evaluating recommendation systems," in *Recommender Systems Handbook*. Springer, 2011, pp. 257–297, doi: 10.1007/978-0-387-85820-3_8.

[49] N. Mishra, S. Chaturvedi, A. Vij, and S. Tripathi, "Research problems in recommender systems," *J. Phys., Conf. Ser.*, vol. 1717, no. 1, Jan. 2021, Art. no. 012002.

[50] M. K. Najafabadi, A. H. Mohamed, and M. N. Mahrin, "A survey on data mining techniques in recommender systems," *Soft Comput.*, vol. 23, no. 2, pp. 627–654, Jan. 2019.

[51] R. Kumar, A. I. Khan, Y. B. Abushark, M. M. Alam, A. Agrawal, and R. A. Khan, "A knowledge-based integrated system of hesitant fuzzy set, AHP and TOPSIS for evaluating security-durability of web applications," *IEEE Access*, vol. 8, pp. 48870–48885, 2020.

[52] A. Tamrawi, T. T. Nguyen, J. Al-Kofahi, and T. N. Nguyen, "Fuzzy set-based automatic bug triaging (NIER track)," in *Proc. 19th ACM SIGSOFT Symp., 13th Eur. Conf. Found. Softw. Eng.*, 2021, pp. 884–887.

[53] A. Goyal and N. Sardana, "Empirical analysis of ensemble machine learning techniques for bug triaging," in *Proc. 12th Int. Conf. Contemp. Comput. (IC3)*, Aug. 2019, pp. 1–6.

[54] M. Wei, S. Guo, R. Chen, and J. Gao, "Enhancing bug report assignment with an optimized reduction of training set," in *Proc. Int. Conf. Knowl. Sci., Eng. Manag.*, in Lecture Notes in Artificial Intelligence, vol. 11062, 2018, pp. 36–47.

[55] Y. Kashiwa and M. Ohira, "A release-aware bug triaging method considering developers' bug-fixing loads," *IEICE Trans. Inf. Syst.*, vol. E103.D, no. 2, pp. 348–362, 2020.

[56] J. Lee, D. Kim, and W. Jung, "Cost-aware clustering of bug reports by using a genetic algorithm," *J. Inf. Sci. Eng.*, vol. 35, no. 1, pp. 175–200, 2019.

[57] V. Akila, V. Govindasamy, and S. Sharmila, "Bug Triage based on ant system with evaporation factor tuning," *Int. J. Control Theory Appl.*, vol. 9, no. 2, pp. 859–863, 2016.

[58] Md. M. Rahman, G. Ruhe, and T. Zimmermann, "Optimized assignment of developers for fixing bugs an initial evaluation for eclipse projects," in *Proc. 3rd Int. Symp. Empirical Softw. Eng. Meas.*, Oct. 2009, pp. 439–442.

[59] S. G. Jindal and A. Kaur, "Automatic keyword and sentence-based text summarization for software bug reports," *IEEE Access*, vol. 8, pp. 65352–65370, 2020.

[60] R. R. Panda and N. K. Nagwani, "Classification and intuitionistic fuzzy set based software bug triaging techniques," *J. King Saud Univ., Comput. Inf. Sci.*, vol. 34, no. 8, pp. 6303–6323, Sep. 2022.

[61] P. M. Vu, T. T. Nguyen, and T. T. Nguyen, "Fuzzy multi-intent classifier for user generated software documents," in *Proc. ACM Southeast Conf.*, Apr. 2020, pp. 292–295.

[62] C. Gupta and M. M. Freire, "A decentralized blockchain oriented framework for automated bug assignment," *Inf. Softw. Technol.*, vol. 134, Jun. 2021, Art. no. 106540.

[63] R. R. Panda and N. K. Nagwani, "Multi-label software bug categorisation based on fuzzy similarity," *Int. J. Comput. Sci. Eng.*, vol. 24, no. 3, pp. 244–258, 2021.

[64] S. Guo, X. Zhang, X. Yang, R. Chen, C. Guo, H. Li, and T. Li, "Developer activity motivated bug triaging: Via convolutional neural network," *Neural Process. Lett.*, vol. 51, no. 3, pp. 2589–2606, Jun. 2020.

[65] Y. Liu, J. X. Huang, and Y. T. Ma, "An automatic method using hybrid neural networks and attention mechanism for software bug triaging," *J. Comput. Res. Develop.*, vol. 57, no. 3, p. 461, 2020.

[66] C. A. Choquette-Choo, D. Sheldon, J. Proppe, J. Alphonso-Gibbs, and H. Gupta, "A multi-label, dual-output deep neural network for automated bug triaging," in *Proc. 18th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2019, pp. 937–944.

[67] M. A. Wani, F. A. Bhat, S. Afzal, and A. I. Khan, *Advances in Deep Learning*. Springer, 2020.

[68] H. Wu, Y. Ma, Z. Xiang, C. Yang, and K. He, "A spatial–temporal graph neural network framework for automated software bug triaging," *Knowl.-Based Syst.*, vol. 241, Apr. 2022, Art. no. 108308.

[69] Y. Liu, X. Qi, J. Zhang, H. Li, X. Ge, and J. Ai, "Automatic bug triaging via deep reinforcement learning," *Appl. Sci.*, vol. 12, no. 7, p. 3565, Mar. 2022.

[70] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques* (The Morgan Kaufmann Series in Data Management Systems), vol. 5, no. 4, 3rd ed. 2011, pp. 83–124.

[71] A. Sajedi-Badashian and E. Stroulia, "Guidelines for evaluating bug-assignment research," *J. Softw., Evol. Process*, vol. 32, no. 9, Sep. 2020, Art. no. e2250.

[72] E. Kalliamvakou, C. Bird, T. Zimmermann, A. Begel, R. DeLine, and D. M. German, "What makes a great manager of software engineers?" *IEEE Trans. Softw. Eng.*, vol. 45, no. 1, pp. 87–106, Jan. 2019.

[73] P. L. Li, A. J. Ko, and A. Begel, "What distinguishes great software engineers?" *Empirical Softw. Eng.*, vol. 25, no. 1, pp. 322–352, Jan. 2020.

[74] Y. Fan, X. Xia, D. Lo, and A. E. Hassan, "Chaff from the wheat: Characterizing and determining valid bug reports," *IEEE Trans. Softw. Eng.*, vol. 46, no. 5, pp. 495–525, May 2020.

[75] A. Goyal and N. Sardana, "An empirical study of non-reproducible bugs," *Int. J. Syst. Assurance Eng. Manage.*, vol. 10, no. 5, pp. 1186–1220, Oct. 2019.

[76] B. Alkhazi, A. DiStasi, W. Aljedaani, H. Alrubaye, X. Ye, and M. W. Mkaouer, "Learning to rank developers for bug report assignment," *Appl. Soft Comput.*, vol. 95, Oct. 2020, Art. no. 106667.

[77] J. R. Martínez-García, F.-E. Castillo-Barrera, R. R. Palacio, G. Borrego, and J. C. Cuevas-Tello, "Ontology for knowledge condensation to support expertise location in the code phase during software development process," *IET Softw.*, vol. 14, no. 3, pp. 234–241, Jun. 2020.

[78] M. Nayebi, L. Dicke, R. Ittyipe, C. Carlson, and G. Ruhe, "ESSMArT way to manage customer requests," *Empirical Softw. Eng.*, vol. 24, no. 6, pp. 3755–3789, Dec. 2019.

[79] L. Kumar, S. Tummalapalli, and L. B. Murthy, "An empirical framework to investigate the impact of bug fixing on internal quality attributes," *Arabian J. Sci. Eng.*, vol. 46, no. 4, pp. 3189–3211, Apr. 2021, doi: 10.1007/S13369-020-05095-0.

[80] S. Kumar, A. K. Kar, and P. V. Ilavarasan, "Applications of text mining in services management: A systematic literature review," *Int. J. Inf. Manage. Data Insights*, vol. 1, no. 1, Apr. 2021, Art. no. 100008.

[81] Y. Kim, J. Kim, H. Jeon, Y.-H. Kim, H. Song, B. Kim, and J. Seo, "Githru: Visual analytics for understanding software development history through git metadata analysis," *IEEE Trans. Vis. Comput. Graphics*, vol. 27, no. 2, pp. 656–666, Feb. 2021, doi: 10.1109/TVCG.2020.3030414.

[82] S.-Q. Xi, Y. Yao, X.-S. Xiao, F. Xu, and J. Lv, "Bug triaging based on tossing sequence modeling," *J. Comput. Sci. Technol.*, vol. 34, no. 5, pp. 942–956, Sep. 2019.

[83] R. Almhana and M. Kessentini, "Considering dependencies between bug reports to improve bugs triage," *Automated Softw. Eng.*, vol. 28, no. 1, pp. 1–26, May 2021.

[84] Y. Huang, N. Jia, H.-J. Zhou, X.-P. Chen, Z.-B. Zheng, and M.-D. Tang, "Learning human-written commit messages to document code changes," *J. Comput. Sci. Technol.*, vol. 35, no. 6, pp. 1258–1277, Nov. 2020.
[85] C. Yang, Q. Fan, T. Wang, G. Yin, X.-H. Zhang, Y. Yu, and H.-M. Wang, "RepoLike: Amulti-feature-based personalized recommendation approach for open-source repositories," *Frontiers Inf. Technol. Electron. Eng.*, vol. 20, no. 2, pp. 222–237, Feb. 2019.
[86] K. F. F. Jiechieu and N. Tsopze, "Skills prediction based on multi-label resume classification using CNN with model predictions explanation," *Neural Comput. Appl.*, vol. 33, no. 10, pp. 5069–5087, May 2021.
[87] S. S. Rathore and S. Kumar, "A study on software fault prediction techniques," *Artif. Intell. Rev.*, vol. 51, no. 2, pp. 255–327, Feb. 2019.
[88] E. U. Aktas and C. Yilmaz, "Automated issue assignment: Results and insights from an industrial case," *Empirical Softw. Eng.*, vol. 25, no. 5, pp. 3544–3589, Sep. 2020.
[89] K. K. Bejjanki, J. Gyani, and N. Gugulothu, "Class imbalance reduction (CIR): A novel approach to software defect prediction in the presence of class imbalance," *Symmetry*, vol. 12, no. 3, p. 407, Mar. 2020.
[90] L. Gong, S. Jiang, and L. Jiang, "Tackling class imbalance problem in software defect prediction through cluster-based over-sampling with filtering," *IEEE Access*, vol. 7, pp. 145725–145737, 2019.
[91] C. Tantithamthavorn, A. E. Hassan, and K. Matsumoto, "The impact of class rebalancing techniques on the performance and interpretation of defect prediction models," *IEEE Trans. Softw. Eng.*, vol. 46, no. 11, pp. 1200–1219, Nov. 2020.
[92] Z. Yuan, X. Chen, Z. Cui, and Y. Mu, "ALTRA: Cross-project software defect prediction via active learning and TrAdaBoost," *IEEE Access*, vol. 8, pp. 30037–30049, 2020.
[93] A. Goyal and N. Sardana, "NRFixer: Sentiment based model for predicting the fixability of non-reproducible bugs," *E-Informatica Softw. Eng. J.*, vol. 11, no. 1, pp. 109–122, 2017.
[94] H. A. Ahmed, N. Z. Bawany, and J. A. Shamsi, "CaPBug—A framework for automatic bug categorization and prioritization using NLP and machine learning algorithms," *IEEE Access*, vol. 9, pp. 50496–50512, 2021, doi: 10.1109/ACCESS.2021.3069248.
[95] P. Oliveira, R. M. C. Andrade, I. Barreto, T. P. Nogueira, and L. M. Bueno, "Issue auto-assignment in software projects with machine learning techniques," in *Proc. IEEE/ACM 8th Int. Workshop Softw. Eng. Res. Ind. Pract. (SER IP)*, Madrid, Spain, Jun. 2021, pp. 65–72, doi: 10.1109/SER-IP52554.2021.00018.
[96] M. Panda and A. T. Azar, "Hybrid multi-objective Grey Wolf search optimizer and machine learning approach for software bug prediction," in *Handbook of Research on Modeling, Analysis, and Control of Complex Systems*. 2020, doi: 10.4018/978-1-7998-5788-4.
[97] Y. Shi, Y. Mao, T. Barnes, M. Chi, and T. W. Price, "More with less: Exploring how to use deep learning effectively through semi-supervised learning for automatic bug detection in student code," *Int. Educ. Data Mining Soc*. [Online]. Available: https://par.nsf.gov/biblio/10340894
[98] A. Yadav, "Bug assignment-utilization of metadata features along with feature selection and classifiers," in *Applications of Artificial Intelligence and Machine Learning*. 2021, doi: 10.1007/978-981-16-3067-5_7.
[99] Z. J. Szamosvölgyi, E. T. Váradi, Z. Tóth, J. Jász, and R. Ferenc, "Assessing ensemble learning techniques in bug prediction," in *Computational Science and Its Applications—ICCSA 2021* (Lecture Notes in Computer Science), vol. 12955. Springer, 2021, doi: 10.1007/978-3-030-87007-2_26.
[100] A. Sawada, E. Kaneko, and K. Sagi, "Trade-offs in top-k classification accuracies on losses for deep learning," 2020, *arXiv:2007.15359*.
[101] V. Nath, D. Sheldon, and J. Alphonso-Gibbs, "Principal component analysis and entropy-based selection for the improvement of bug triage," in *Proc. 20th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Pasadena, CA, USA, Dec. 2021, pp. 541–546, doi: 10.1109/ICMLA52953.2021.00090.
[102] S. F. A. Zaidi, H. Woo, and C.-G. Lee, "A graph convolution network-based bug triage system to learn heterogeneous graph representation of bug reports," *IEEE Access*, vol. 10, pp. 20677–20689, 2022, doi: 10.1109/ACCESS.2022.3153075.

RAZVAN BOCU received the B.S. degree in computer science, the B.S. degree in sociology, and the M.S. degree in computer science from the Transilvania University of Brasov, Brasov, Romania, in 2005, 2006, and 2007, respectively, and the Ph.D. degree from the National University of Ireland, Cork, in 2010.

He is currently with the Department of Mathematics and Computer Science, Transilvania University of Brasov, where he is also a Research and Teaching Staff Member. He is with Siemens Industry Software, Romania. He is the author or coauthor of more than 60 technical articles, together with six books and book chapters. In this capacity, he supervises research projects with strategic business value. He is an editorial reviewing board member of 28 technical journals in the field of information technology and biotechnology, which includes prestigious journals, such as *Journal of Network and Computer Applications*, IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, *International Journal of Computers Communications and Control*.

ALEXANDRA BAICOIANU received the Ph.D. degree from Babeş-Bolyai University, Cluj-Napoca, in 2016.

She has been a Lecturer with the Transilvania University of Braşov, Braşov, Romania, since 2017, teaching various courses and seminars, where she is currently with the Department of Mathematics and Computer Science. She is also with Siemens Industry Software, Romania. She is a Research Engineer of informatics. She supervised tens of graduation and dissertations thesis, programming training courses, programming summer schools, and code/tech camps, some of them in collaboration with IT companies. She is a member of the Department's Machine Learning and Quantum Computing Research Group, founded in 2018. She has published more than 30 scientific articles and is the coauthor of six books. She was a part of various scientific projects, among them it is important to mention Advanced Technologies for Intelligent Urban Electric Vehicles, Powerful Advanced N-level Digital Architecture (PANDA), Intelligent Motion Control under Industry4.E (IMOCO4E), Artificial Intelligence and Earth Observation for Romania's agriculture (AI4AGRI), Digital Technologies and Artificial Intelligence (AI) Solutions projects (DiTArtIS), and New Modular Electrical Architecture and Digital platforM to Optimize Large Battery Systems on SHIPs (NEMOSHIP). Her research interests and expertise are in the field of machine learning, formal languages and compilers, algorithms, remote sensing and Earth observation data, autonomous driving, and electric and hybrid vehicles.

ARPAD KERESTELY is currently with Siemens Industry Software, Romania. He is also a Research and Development Engineer. He was a part of various scientific projects, among them it is important to mention Advanced Technologies for Intelligent Urban Electric Vehicles, Powerful Advanced N-level Digital Architecture (PANDA), Intelligent Motion Control under Industry4.E (IMOCO4E), Artificial Intelligence and Earth Observation for Romania's Agriculture (AI4AGRI), Digital Technologies and Artificial Intelligence (AI) Solutions projects (DiTArtIS), and New Modular Electrical Architecture and Digital platforM to Optimize Large Battery Systems on SHIPs (NEMOSHIP). His research interests and expertise are in the field of machine learning, formal languages and compilers, and algorithms.

AI-Based Visualization of Remotely-Sensed Spectral Images

Ioana Cristina Plajer*, Alexandra Baicoianu† and Luciana Majercsik‡
Faculty of Mathematics and Informatics
Transilvania University of Braşov, Romania
* ioana.plajer@unitbv.ro, † a.baicoianu@unitbv.ro, ‡ luciana.carabaneanu@unitbv.ro

***Abstract*— With the increase in multispectral and hyperspectral satellite data availability, the necessity of interpreting and processing such data is also growing. Satellite imagery can be used in a wide range of fields, from military and defence applications to ecology, agriculture and forest management. As multi- and hyperspectral images cannot be directly interpreted either by the human eye or by usual computer displays, a visually-consistent mapping of these images is necessary. In this paper we propose an approach based on an artificial intelligence (AI) model for spectral image visualisation in the RGB color space. The visualization is performed by a fully-connected neural network trained on the popular CAVE dataset which we consider being suitable for visualization, as it has a significant color diversity in the visible domain. The coloring method was applied on a hyperspectral PRISMA image. The study offers a visual interpretation of the results obtained with the proposed architecture. The results are promising and will be further used for the true mapping of agricultural areas.**

Key words: spectral images, satellite data, RGB, neural network, PRISMA satellite, CAVE dataset, linear interpolation

I. Introduction

Each type of material has a unique spectral fingerprint, which means that light is absorbed differently by objects with different properties. Multispectral (MS) and hyperspectral (HS) sensors can capture tens or even hundreds of spectral bands therefore they are much more sensitive to small changes in an object's reflectance or radiance. In the images produced by such sensors, objects are described by additional parameters besides the descriptive geometric data, as each pixel also contains spectral information about the chemical composition of the objects compared to RGB images. For this reason, the former is increasingly used in many remote sensing fields such as agriculture [1], [2], [3], forestry [4], [5], ecology and environmental monitoring [6], and military and industrial applications.

The visualization of MS and HS data as RBG images is extremely important because an RGB image serves as an interface between the human eye and the multidimensional data space, helping the viewer to correlate pixel areas to the surface features they represent. To generate realistic RGB images using spectral images, a number of visualization methods have been proposed in literature, such as band selection [7][8], PCA based methods [9] or, more recently, machine learning [10][11]. However, research in this field is still quite scarce. For this reason, one of the main objectives of our investigation was to obtain meaningful corresponding RGB images for MS and HS images using a fully-connected neural network (FCNN) trained for this task.

The spectral image sensors provide images with a large number of contiguous spectral channels per pixel and the sources from which these images come are diverse, therefore visualizing these massive datasets is not simple and straightforward. The process of visualization has a particular value for users who need to evaluate the importance of data, which is why we have proposed in this study a suitable architecture for this assignment. As the human visual perception is not necessarily a linear process, the use of a linear color space to represent images might result in an unpleasant effect on the human observer, generating the impression of high contrast between the darker and the lighter areas. Furthermore, pairs of colors which in the RGB color cube are at the same Euclidean distance might be perceived totally different by humans, the RGB cube thus being a perceptually-nonlinear space [12]. Taking this into consideration, the nonlinearity of a neural network (NN) could provide a good way of modeling the transform function from multispectral to tridimensional RGB, in order to offer a perceptually proper interpretation for human users.

The aim of this article is to provide answers to some research questions within the context of spectral image visualization. One of the open questions is if an AI model is capable of learning the correspondence between MS or HS reflectance curves and RGB triplets. The results of our experiments confirm that by training a NN to learn the RGB equivalent for spectral pixels, visually consistent RGB images can by generated. Two important considerations should be mentioned here. The first one is that at the moment, the interpretation of the results has been done from a visual point of view and considering some common metrics. The second is that the network testing was done using a single HS satellite image, and this should be extended to test other satellite images. Another open concern in the field of spectral image visualization is to discover what are the main characteristics of a data set suitable for optimal coloring results. As our tests have shown, in order to have a good coloring, it is important to use for the training of the NN a dataset with a significant color diversity in the visible domain. The amount of data on which training is done must be large enough and correct standardization before training is extremely important. A third

issue considered was to feature the advantages of coloring spectral images using a NN over other alternative visualization techniques. We can mention among these advantages the relatively easy adaptability through an interpolation process to inputs having a different spectral resolution. Another advantage of a NN is its non-linearity, the network being thus able to emulate the nonlinear visual perception of human users and so to enhance the coloring output. Furthermore enhancing the training data set could be an option to achieve improved visual results.

II. Datasets

The high-resolution spectral information provided by multi- and hyperspectral imagery has changed the way we think about environmental and ecosystem phenomena. More and more such data are becoming available for scientific and practical purposes, and analysing them is a first step in better understanding the phenomena mentioned above. Various datasets are available and known including CAVE [13], UGR [14], UEA Colour Group datasets [15]. For this study we chose the CAVE dataset, which is well known in literature, because it offers a good diversity of colors, the images were acquired in controlled environment and for each MS image an RGB correspondent is available. For testing we used a HS image provided by the relatively new PRISMA satellite.

A. CAVE Dataset

The CAVE high-resolution MS image dataset [13] contains 32 images of indoor scenes. Each MS image has a resolution of 512×512 pixels and covers a wavelength range in [400 - 700 nm], sampled at 10 nm intervals, providing a total of 31 channels. Each scene has also a unique corresponding color image representation, rendered under a neutral daylight illuminant and displayed using sRGB values. Some samples from CAVE dataset are displayed in Figure 1.

Fig. 1: RGB images from the CAVE database.

B. PRISMA Image

The PRISMA image used for testing is one of the images captured by the Italian Space Agency (ASI)'s PRISMA hyperspectral satellite on 18th of October 2022 in the north of Brasov county, Romania. The hyperspectral sensors of the satellite are able to recall images in a wavelength range of 239 spectral bands between [400 - 2500 nm], 66 in the Visible Near Infra Red [400 - 1010 nm], and 173 in the Short Wave Infra Red [920 - 2500 nm], with a spectral sampling interval smaller than 12 nm. The images have a spatial resolution of 1000×1000 pixels, with a ground sample distance of 30 m [16]. The spectral bands used in the experiments are in the visible domain from 406 nm to 713 nm, roughly 8 nm sampled.

III. The Proposed AI Model Architecture

In practice, a number of classical algorithms are commonly used to visualize MS images, for example by using band selection or linear color formation models. These approaches presume the choice of the appropriate illuminant and the modelling of the nonlinearity of human perception by gamma correction. The results of these algorithms often present low visual quality. As mentioned in the introduction, a NN approach could as well simulate the nonlinear human perception as offer a general solution for MS images acquired by different systems.

A. Model Description

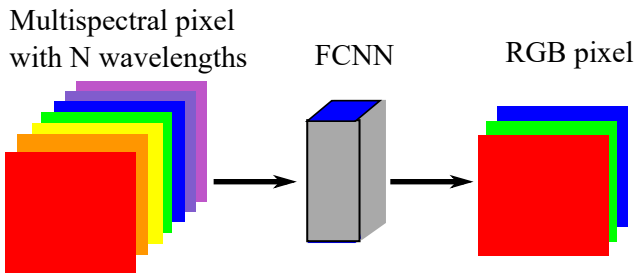


Fig. 2: Model pipeline.

As the coloring for visualization of the MS images can be formulated as a regression problem, it seems appropriate to use for this purpose a FCNN, inspired by the one proposed in [17]. As the number of wavelength present in the CAVE dataset used for training the model are 31, this is the number of neurons in the input layer of the network. The network is constructed with 3 fully connected hidden layers and an output layer containing three neurons, one for each of the RGB color channels. Thus, the model estimates for each MS pixel input an RGB output, see Figure 2.

B. Training of the Model

The FCNN model was trained on all the pixels of all the images in the CAVE dataset, collected into a single randomly shuffled set. This set was partitioned into train and test set and the model was trained until the loss decay on both train and test set was stabilized. During each training epoch randomly selected pixel batches of the training set were passed through the network, using PyTorch data loader. The training stages can be synthesized by the following methodology.

Training Steps

1) Loading of all the pixels from all the images into one dataframe [18].
2) Random shuffling of all the pixels.

3) Partitioning of the set into train set - 75% of the pixels - and test set - 25% of the pixels.
4) Standardization of the training using the PyTorch standard scaler and using the transformation on test dataset [19].
5) Training the model with random batches of 2048 pixels.

IV. Analysis and Interpretation of Results

Following a set of experiments it proved sufficient to train the model on 150 epochs. To validate the correctness of the training method, k-fold cross validation with 10 folds was used. On each fold the average losses on the training and on the test were calculated and plotted. The results on most folds were similar, indicating the correctness of the approach. The loss decay during training on one of the folds is presented in Figure 3. It can be seen, that the loss on the training set has a steep decay, while for similar results on the test set, the model needs more training.

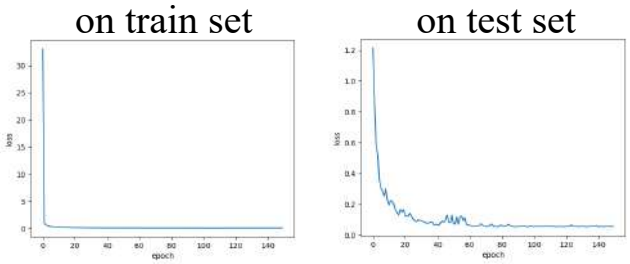


Fig. 3: Loss decay on CAVE dataset.

During the k-fold process all the weights were saved and the best ones were used for coloring all the MS images of the CAVE dataset. The visual outputs were accurate, further validating the chosen model. Figure 4 is presenting the coloring result on two samples of the dataset together with the provided RGB label image. One may notice that the two images are visually identical. The similarity measures for the pairs of input-output images in the train set confirm the visual results. The considered quality metrics were Mean Squared Error (MSE) and Peak Signal-to-Noise Ratio (PSNR). For MSE the values for the pairs are between [0.09-0.17] and the values for PSNR are between [55.71-58.45].

The trained model was also used to visualize HS satellite images obtained from the PRISMA satellite. It must be mentioned, that the spectral bands of the PRISMA images differ slightly from those of the images in the CAVE dataset. We considered from the PRISMA images only the wavelength in the visual range and linearly interpolated them to fit those of the CAVE dataset. The coloring of the HS images was performed by the following procedure.

Inference

1) Load the pixels of the PRISMA image into a dataframe.
2) Linearly interpolate the values as to fit the wavelength to those of the CAVE images. As the spectral range of the PRISMA image is very similar to the one of CAVE images, we considered linear interpolation as accurate enough.
3) Standardize the pixels of the dataframe relative to their mean and variance using the standard PyTorch scaler.
4) Pass each pixel through the model to predict the corresponding (R, G, B) triplet.
5) Construct the RGB-image with respect to the original size of the PRISMA image and save as PNG file.

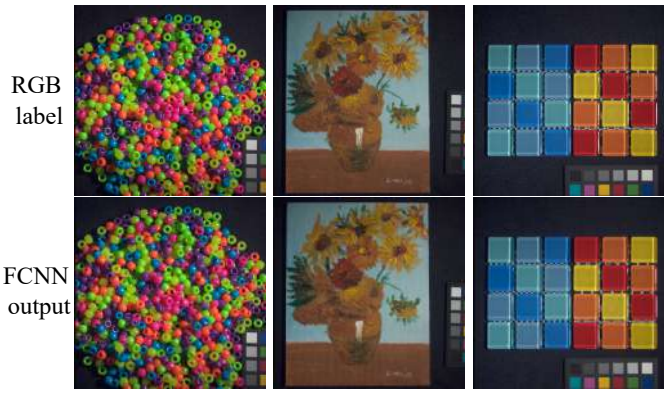
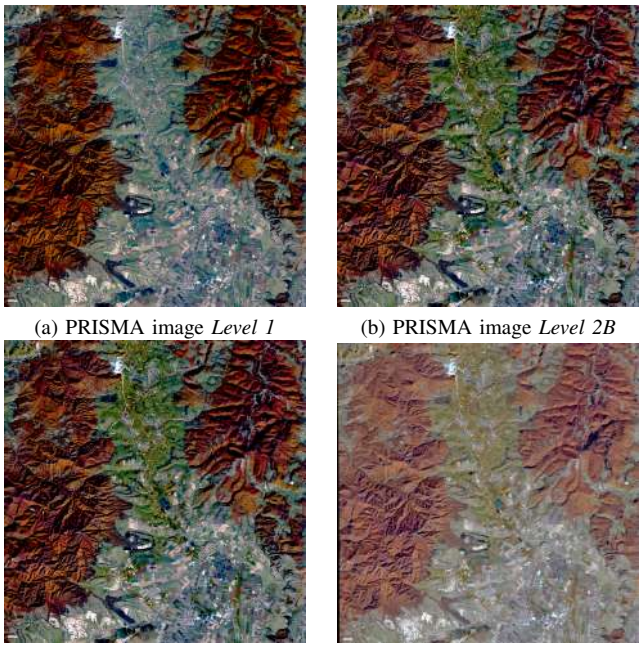


Fig. 4: Coloring of two MS images from the CAVE dataset using the trained FCNN model.

A result of this coloring on a PRISMA image is presented in Figure 5. Four different product levels of the same PRISMA image were visualized by the network, each of them presenting the initial HS image, while the other three present subsequent levels of correction. *Level 1* is a top of the atmosphere radiance imagery. After the atmospheric correction and geolocation of *Level 1*, *Level 2B* image contains the information about the reflected radiance of the Earth's surface, and *Level 2C* has the information about the boundary reflection coefficient, aerosol optical thickness and water vapor map. The last level, *Level 2D*, represents the image after all the transforms plus orthorectification [16], [5]. According to [16], "the orthorectification process foresees the correction of all image distortions caused by the collection geometry (this includes the optical sensor characteristics) and the variable terrain".

As can be seen in Figure 5, the results of the coloring are promising. The expected natural coloring of the image is achieved in all four cases, given that the image was acquired late October. The region of the valley is accurately rendered and the agricultural parcels can be clearly distinguished. There are still some artefacts present, probably due to highly reflective surfaces. This might be due to the fact that the training dataset was acquired in the indoor environment and has a specific spectral signature. Another aspect might concern the interpolation method, opening the research possibility of other interpolation or mapping methods.

It also can be noticed that after corrections in *Level 2*, the slightly blueish coloring of the Level 1 HS image is attenuated, and the green color of the vegetation is enhanced. The most pronounced difference in color to the original sample can be observed after orthorectification. *Level 2C* leads to a higher



(a) PRISMA image *Level 1* (b) PRISMA image *Level 2B*

(c) PRISMA image *Level 2C* (d) PRISMA image Level 2D

Fig. 5: Coloring of four products of PRISMA image using the FCNN model with weights trained on CAVE dataset before and after corrections.

contrast are more vivid colors. *Level 2B* and *Level 2C* images are the most colorful.

V. Conclusions and Future work

This paper presents a novel fully connected model for the task of HS/MS images coloring. The given architecture has yielded encouraging results. This study was developed on two known datasets in the field, CAVE and PRISMA, but we are interested in running additional tests on several datasets with different characteristics, the indoor category which is a controlled environment, and also the outdoor variety which is a natural environment. Such an approach can bring generality to the solution already offered, but also optimal and diversified results. However, the proposed network can further be optimized, for instance, one minor restriction that we have noticed through testing is that the current model still generates errors for highly reflective surfaces. In addition, at this point of the study, visually it can be seen that the results are meaningful. Still, there is necessary to validate the results with other different classical (Euclidean Distance, Mahalanobis Distance, Cosine Similarity, etc) and/or specific (Structural Similarity Index, Universal Quality Image Index, etc) metrics. We also want to improve the performance and results of the model by enhancing the input datasets.

Acknowledgment

This work was funded from the AI4AGRI project entitled "Romanian Excellence Center on Artificial Intelligence on Earth Observation Data for Agriculture". The AI4AGRI project received funding from the European Union's Horizon Europe research and innovation program under the grant agreement no. 101079136. The hyperspectral image from the PRISMA satellite presented in this paper was kindly provided by the Italian Space Agency (ASI).

References

[1] T. Adão et al., "Hyperspectral Imaging: A Review on UAV-Based Sensors, Data Processing and Applications for Agriculture and Forestry", *Remote Sensing*, 2017, Volume 9(11). [Online]. https://doi.org/10.3390/rs9111110.

[2] M. Weiss, F. Jacob, G. Duveiller, "Remote sensing for agricultural applications: A meta-review", *Remote Sensing of Environment,* Volume 236, 2020, pp. 111402, ISSN 0034-4257, [Online]. https://doi.org/10.1016/j.rse.2019.111402.

[3] J. Zhao et al., "Deep-Learning-Based Multispectral Image Reconstruction from Single Natural Color RGB Image—Enhancing UAV-Based Phenotyping", *Remote Sensing*, 2022, Volume 14, pp. 1272. [Online]. https://doi.org/10.3390/rs14051272.

[4] P. Rodríguez-Veiga et al., "Forest biomass retrieval approaches from earth observation in different biomes", *Int. J. Appl. Earth Obs. Geoinf*, 2019, Volume 77, pp. 53–68.

[5] E. Vangi et al., "The New Hyperspectral Satellite PRISMA: Imagery for Forest Types Discrimination," *Sensors*, 2021, Volume 21, pp. 1182, [Online]. https://doi.org/10.3390/s21041182.

[6] L. Du et al., "A comprehensive drought monitoring method integrating MODIS and TRMM data," *Int. J. Appl. Earth Obs. Geoinf*, 2013, Volume 23, pp. 245–253.

[7] B. Demir, A. Celebi, S. A. Erturk, "Low-complexity approach for the color display of hyperspectral remote-sensing images using one-bit-transform-based band selection", *IEEE Trans. Geosci. Remote Sens*, 2008, Volume 47, pp. 97–105.

[8] S. Le Moan, A. Mansouri, Y. Voisin, J.Y. Hardeberg, "A constrained band selection method based on information measures for spectral image color visualization", *IEEE Trans. Geosci. Remote Sens.*, 2011, Volume 49, pp. 5104–5115.

[9] H.A. Khan, M.M. Khan, K, Khurshid, J. Chanussot, "Saliency based visualization of hyper-spectral images", *Proceedings of the 2015 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, Milan, Italy, 2015, pp. 1096–1099.

[10] P. Duan, X. Kang, S. Li, "Convolutional neural network for natural color visualization of hyperspectral images", *Proceedings of the IGARSS 2019–2019 IEEE International Geoscience and Remote Sensing Symposium*, Yokohama, Japan, 2019, pp. 3372-3375, doi: 10.1109/IGARSS.2019.8900359.

[11] R. Tang, H. Liu, J. Wei, W. Tang,W. "Supervised learning with convolutional neural networks for hyperspectral visualization", *Remote Sens. Lett*, 2020, 11, pp. 363–372.

[12] C. A. Poynton, *A Technical Introduction to Digital Video*, John Wiley & Sons, Inc., 1996, ISBN: 047112253X.

[13] F. Yasuma, T. Mitsunaga, D. Iso, and S.K. Nayar, "Generalized Assorted Pixel Camera: Post-Capture Control of Resolution, Dynamic Range and Spectrum", Technical Report, Department of Computer Science, Columbia University CUCS-061-08, 2008.

[14] J. Eckhard, T. Eckhard, E.M. Valero, J.L. Nieves, E. Garrote Contreras, "Outdoor scene reflectance measurements using a Bragg-grating-based hyperspectral imager", *Applied Optics*, Volume 54 (13), pp. D15-D24, 2015.

[15] UEA Colour Group Datasets, [Online]. Available: https://colour.cmp.uea.ac.uk/datasets/multispectral.html

[16] ASI. Prisma Products Specification Document Issue 2.3 Date 12/03/2020. [Online]. Available: http://prisma.asi.it/missionselect/docs/PRISMA%20Product%20Specifications_Is2_3.pdf

[17] R.M. Coliban, M. Marincaş, C. Hatfaludi, M. Ivanovici, "Linear and Non-Linear Models for Remotely-Sensed Hyperspectral Image Visualization", *Remote Sensing*, Volume 12(15), 2020, [Online]. https://doi.org/10.3390/rs12152479.

[18] DataFrame, [Online]. Available: https://pytorch.org/torcharrow/beta/dataframe.html (accessed on 17.12.2022)

[19] StandardScaler, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html (accessed on 10.11.2022)