Received 9 March 2024, accepted 22 March 2024, date of publication 27 March 2024, date of current version 2 April 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3382541

RESEARCH ARTICLE

Learning About Growing Neural Cellular Automata

SORANA CATRINA[1], MIRELA CATRINA[1], ALEXANDRA BĂICOIANU[2], AND IOANA CRISTINA PLAJER[2]

[1]Faculty of Mathematics and Computer Science, Transilvania University of Braşov, 500036 Braşov, Romania
[2]Department of Mathematics and Computer Science, Transilvania University of Braşov, 500036 Braşov, Romania

Corresponding author: Alexandra Băicoianu (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

The associate editor coordinating the review of this manuscript and approving it for publication was Hengyong Yu.

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);

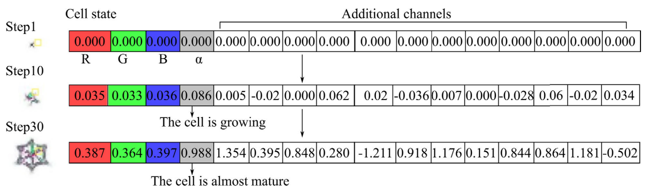

FIGURE 1. Encoding strategy.

- 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 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 $72 \times 72 \times 16$ array. The three arrays obtained thus are concatenated into the network's input feature array of size $72 \times 72 \times 48$. 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.

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

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 $72 \times 72 \times 16$, representing the current state of the NCA, and applies hardcoded 3×3 filters (in this case *Identity*, *Sobel$_x$* and *Sobel$_y$*), outputting a tensor of shape $72 \times 72 \times 48$. 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 $72 \times 72 \times 16$-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.

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.

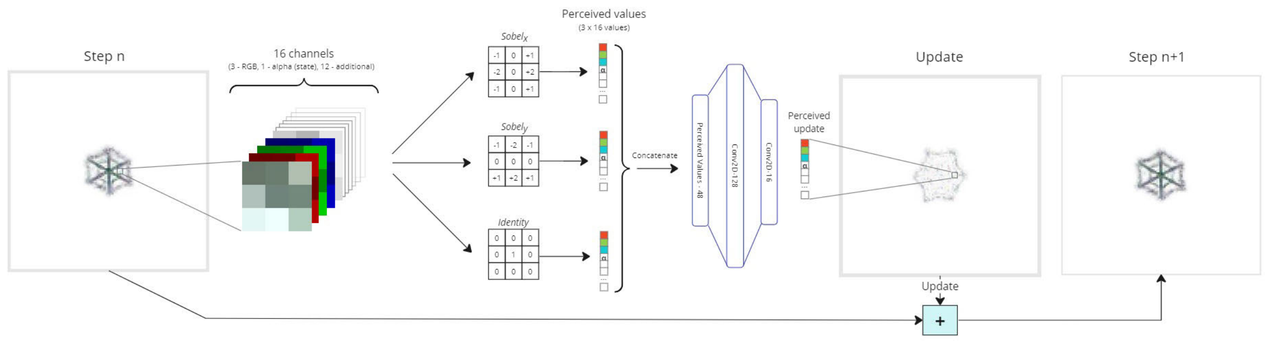

FIGURE 2. **Neural cellular automaton model.**

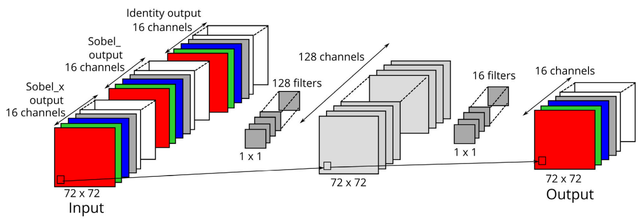

FIGURE 3. **Network architecture.**

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 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 $p \cdot n$.

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

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 $p \cdot n$

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.

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

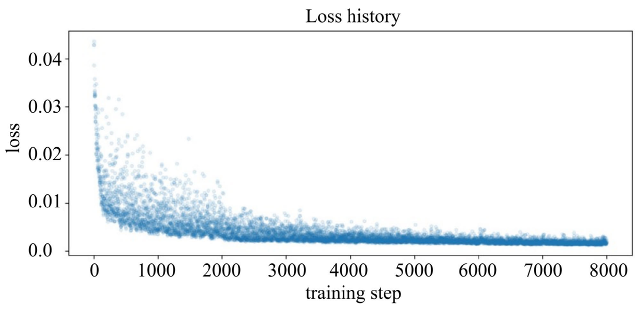

FIGURE 4. Loss history of spiderweb images.

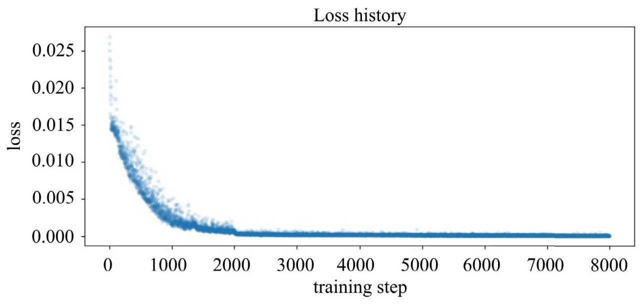

FIGURE 5. Loss history of caterpillar images.

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.

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

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

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 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 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 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 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

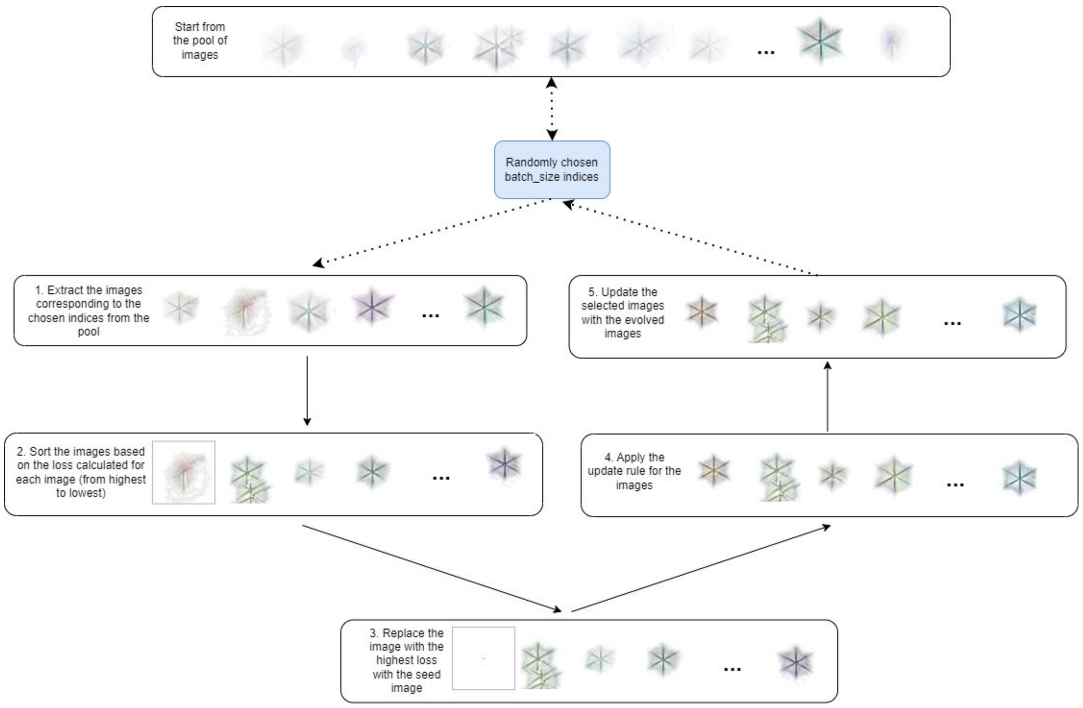
Start from the pool of images
...
Randomly chosen batch_size indices
1. Extract the images corresponding to the chosen indices from the pool
5. Update the selected images with the evolved images
2. Sort the images based on the loss calculated for each image (from highest to lowest)
4. Apply the update rule for the images
3. Replace the image with the highest loss with the seed image

FIGURE 6. Pooling strategy depicted for one epoch.

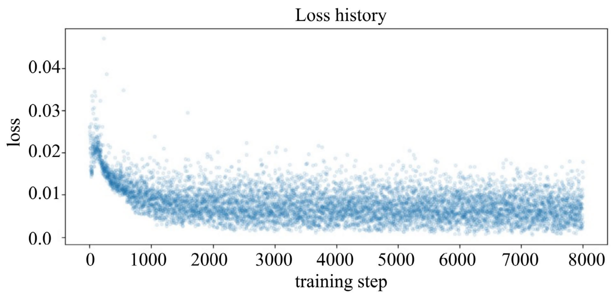

FIGURE 7. Loss history of spiderweb image.

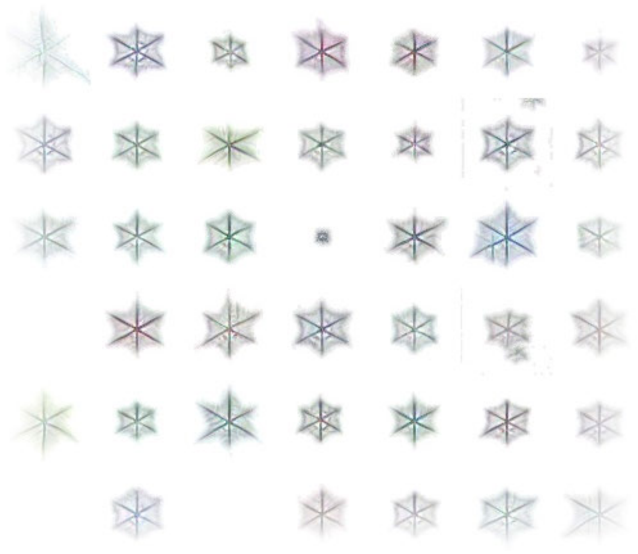

FIGURE 8. Pool of images observed to be in different stages of training.

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.

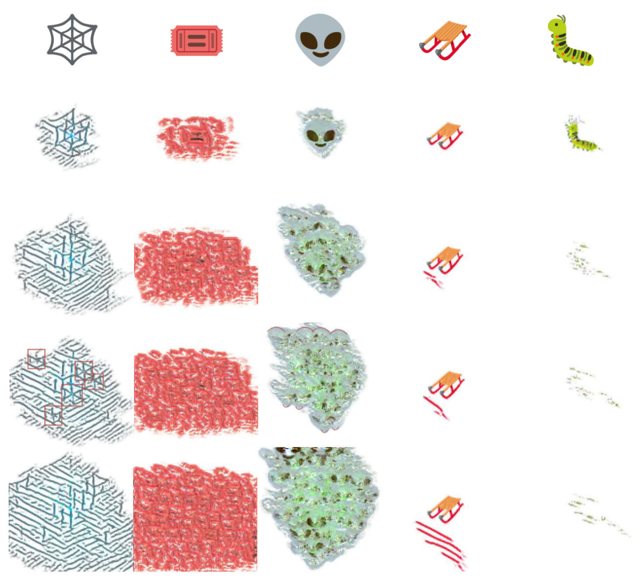

FIGURE 9. Expansion behaviors for figures in the 1st column exhibiting different symmetry properties, after 300, 500, 600 and 800 iterations.

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.

1) 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.

2) 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.

3) 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.

4) 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.

5) 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.

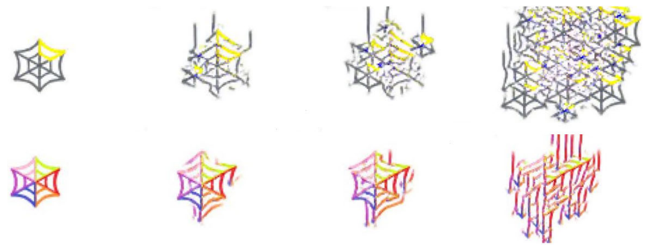
FIGURE 10. Expansion behaviours for the differently colored spiderweb after 100, 250, 300 and 600 iterations.

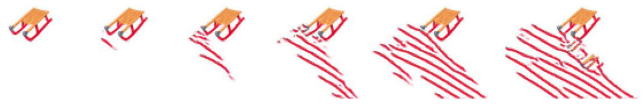
FIGURE 11. Evolution results for the sleigh image after 200, 500, 750, 1000, 1500 and 2000 iterations.

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.

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

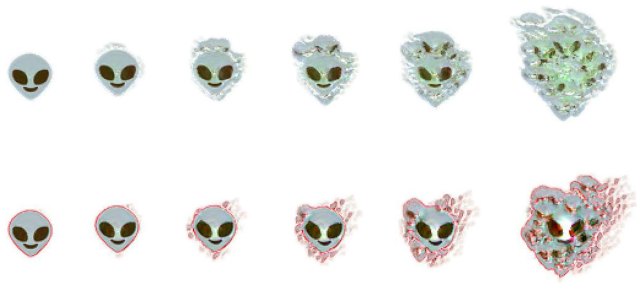
FIGURE 12. Evolution results for the unbordered and of the bordered alien after 100, 150, 200, 250, 300 and 500 iterations.

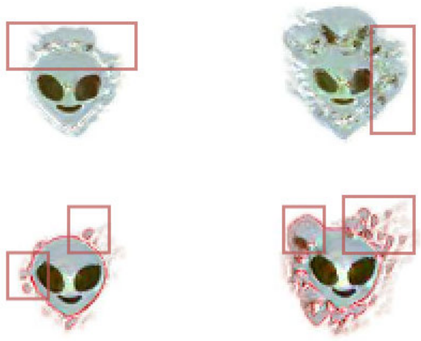
FIGURE 13. Details in the multiplication of the shape during the evolution of the unbordered and of the bordered alien after 200 respectively 300 iterations.

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 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.

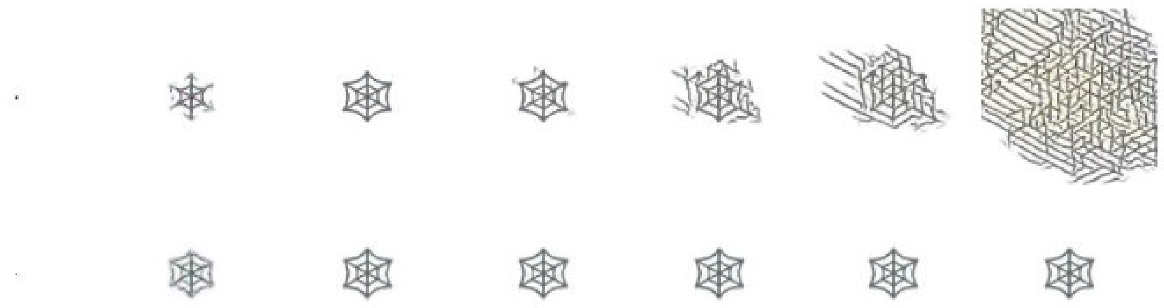

FIGURE 14. Comparison of an organism grown after training with Algorithm 3 (up) and respectively the Algorithm 4 (bottom) steps 0, 50, 100, 150, 250, 350 and 1000.

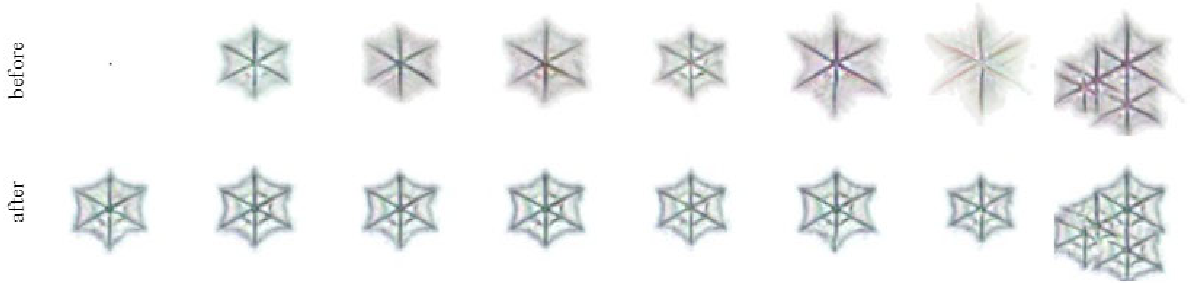

FIGURE 15. Batch 700. Based on the approach presented in Algorithm 4, the randomly selected pool of individuals before and after training of batch number 700.

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 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

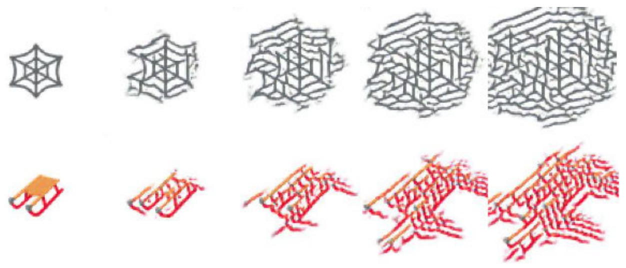

FIGURE 16. Evolution results for NCA using rotated Sobel filters for the spiderweb and the sleight: original and after 300, 500, 600 and 800 iterations.

$Sobel_x$ 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.

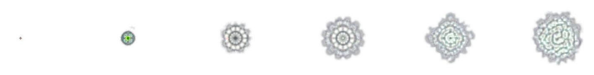

FIGURE 17. Evolution results for NCA using the Laplace filter for the spiderweb after 1, 50, 100, 250, 500 and 1000 iterations.

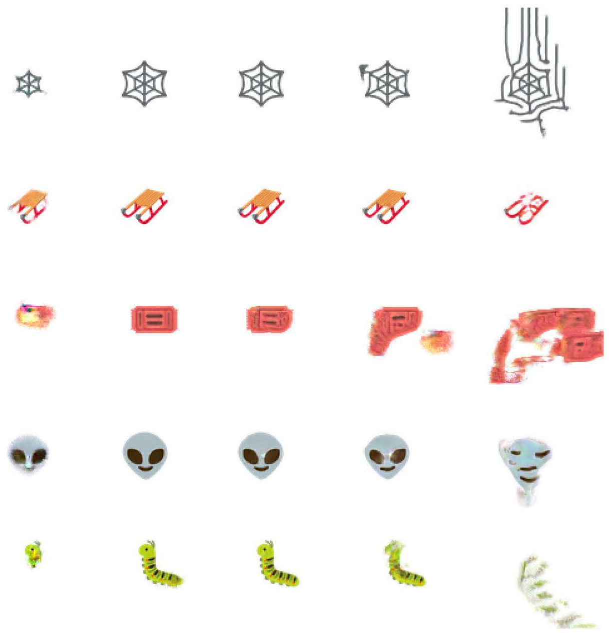

FIGURE 18. Evolution results for NCA using the $Sobel_x$, $Sobel_y$ and the laplace filters after 50, 100, 250, 500 and 1000 iterations.

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**	

IV. CONCLUSION

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," *Theor. Comput. Sci.*, vol. 334, nos. 1–3, pp. 3–33, Apr. 2005.

[2] M. Khanjary, "Cellular learning automata: Review and future trend," in *Computational Vision and Bio-Inspired Computing*. Singapore: Springer, 2022, pp. 229–238.

[3] E. W. Weisstein. (2002). *Cellular Automaton*. [Online]. Available: https://mathworld.wolfram.com/

[4] W. Li and N. Packard, "The structure of the elementary cellular automata rule space," *Complex Syst.*, vol. 4, no. 3, pp. 281–297, 1990.

[5] A. H. Ruiz, A. Vilalta, and F. Moreno-Noguer, "Neural cellular automata manifold," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 10015–10023.

[6] E. W. Weisstein. (2002). *Elementary Cellular Automaton*. [Online]. Available: https://mathworld.wolfram.com/

[7] S. Wolfram, *A New Kind of Science*. Champaign, IL, USA: 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 Comput.*, vol. 19, no. 2, pp. 433–461, Jun. 2020.

[9] D. A. Zaitsev, "k-neighborhood for cellular automata," 2016, *arXiv:1605.08870*.

[10] B. Breckling, G. Pe'er, and Y. Matsinos, *Cellular Automata Ecological Modeling*. Berlin, Germany: Springer, Jan. 2011, pp. 105–117.

[11] A. Mordvintsev and E. Niklasson, "μnca: Texture generation with ultra-compact neural cellular automata," 2021, *arXiv:2111.13545*.

[12] E. Niklasson, A. Mordvintsev, E. Randazzo, and M. Levin, "Self-organising textures," *Distill*, vol. 6, no. 2, Feb. 2021, Art. no. e00027.

[13] K. Horibe, K. Walker, and S. Risi, "Regenerating soft robots through neural cellular automata," 2021, *arXiv:2102.02579*.

[14] A. Mordvintsev, E. Randazzo, E. Niklasson, and M. Levin, "Growing neural cellular automata," *Distill*, vol. 5, no. 2, p. e23, Feb. 2020. [Online]. Available: 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, *arXiv:2205.06806*.

[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," 2021, *arXiv:2103.08737*.

[17] R. Gonzales and R. Woods, *Digital Image Processing*, 4th ed. London, U.K.: Pearson, 2018.

[18] A. Lambora, K. Gupta, and K. Chopra, "Genetic algorithm—A literature review," in *Proc. Int. Conf. Mach. Learn., Big Data, Cloud Parallel Comput. (COMITCon)*, Feb. 2019, pp. 380–384.

[19] J. Stovold, "Neural cellular automata can respond to signals," in *Proc. Conf. Artif. Life*, 2023, pp. 24–32.

[20] E. Pajouheshgar, Y. Xu, T. Zhang, and S. Süsstrunk, "DyNCA: Real-time dynamic texture synthesis using neural cellular automata," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2023, pp. 20742–20751.

[21] R. Castro, R. Gómez, and L. Arancibia, "Fine material migration modelled by cellular automata," *Granular Matter*, vol. 24, no. 1, p. 14, Feb. 2022.

[22] S. Zambrano-Asanza, R. E. Morales, J. A. Montalvan, and J. F. Franco, "Integrating artificial neural networks and cellular automata model for spatial–temporal load forecasting," *Int. J. Electr. Power Energy Syst.*, vol. 148, Jun. 2023, Art. no. 108906.

[23] A. Mordvintsev, E. Randazzo, and C. Fouts, "Growing isotropic neural cellular automata," 2022, *arXiv:2205.01681*.

SORANA CATRINA is currently pursuing the bachelor's degree in computer science with Transilvania University of Brasov. She is passionate about quantum physics and quantum programming, participating in plenty of courses, contests, and Summer Schools on this topic. Alongside quantum, her research interests include algorithms, machine learning, and data structures.

MIRELA CATRINA is currently pursuing the bachelor's degree in computer science with Transilvania University of Brasov. She participates in numerous algorithmic contests, Summer Schools, and projects, affirming an interests include 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 received the Ph.D. degree from Babeş-Bolyai University, Cluj-Napoca, in 2016. She was a member with the Department's Machine Learning Research Group, founded in 2018. She has been a Lecturer with Transilvania University of Brasov, since 2017, teaching various courses and seminars. She is currently a Research Engineer in computer science. She has published more than 30 scientific papers and is the coauthor of seven 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. Her research interests and expertise include the field of machine learning, formal languages and compilers, algorithms, remote sensing and Earth observation data, autonomous driving, and electric and hybrid vehicles.

IOANA CRISTINA PLAJER received the B.E. and M.S. degrees in computer science from the University of Bucharest, Romania, in 1997 and 1998, respectively, and the Ph.D. degree in computer science from Transilvania University of Brasov, Braşov, Romania, in 2011. She was a member with 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). She is currently a Lecturer with the Faculty of Mathematics and Computer Sciences, Transilvania University of Brasov. Her research interests include machine learning, image processing, spectral imaging and remote sensing, formal languages, algorithms, and data structures.