Real-Time Video Stylization Using Neural Cellular Automata

Ali Raed Ben Mustapha Ahmad Jarrar Khan Somesh Mehra

Supervised by Ehsan Pajouheshgar in CS413 at EPFL

Abstract. Real-time video stylization is a challenging task due to the need to produce temporally consistent behavior within very short inference times. Most current methods fall short in one of these criteria, or require a significant amount of compute which is not practical for most applications. Neural Cellular Automata (NCA) meanwhile have shown strong capabilities to generate temporally consistent textures, with extremely efficient and parallelisable implementations that enable real-time synthesis to be performed on everyday devices. In this work, we propose a novel approach for stylizing videos using Neural Cellular Automata. We demonstrate a method to condition NCA generation using a video feed, to ultimately achieve real time video stylization. We exhibit our work using an online, interactive demo which can function on everyday computers and smartphones with a webcam.

1 Introduction

For a relatively simple system where individual cells in a grid all learn the same local rules, Neural Cellular Automata (NCAs) have shown a wide range of powerful capabilities, from generating stable, regenerative images [5] to segmenting high-resolution images [10]. One particular area in which they excel is texture synthesis [6], with recent works developing a framework for real-time and controllable dynamic texture synthesis [8], and even synthesizing these dynamic textures directly on 3D meshes [7].

Whilst these works provide some level of post-training control for the behaviour of the NCAs, to our knowledge there is little work done on training conditional NCAs which can react dynamically to a given input signal. The closest work we found proposed goal-guided NCAs [11], which condition the generation process using an encoding for a set of predefined goals for the NCA. However, this results in NCAs conditioned only on the finite set of goals they were trained on, limiting the generalisability and flexibility of the method.

In this work, we propose a method for training conditional NCAs that can react dynamically to any arbitrary input image. Specifically, we do this with the goal of stylizing videos in real-time, by applying a conditional NCA frame-by-frame on a video. Building upon the DyNCA framework introduced in [8], our method maintains many of the favourable properties of this framework in addition to conditioning on arbitrary input images, all for minimal parameter and performance overheads. Our method is fast to train, and requires very little

training data to learn a generalisable and robust NCA. Once trained, the NCAs can run in real time on a low-end GPU, and enable video stylisation at arbitrary resolution with several video editing controls.

To achieve this, we make some architectural changes to DyNCA to incorporate the conditioning signal when updating cell states. Additionally, whilst the original DyNCA method trains NCAs to have a desired motion and appearance, we introduce an additional objective to mimic the content of the conditioning image. The training objective used here is largely inspired by earlier style transfer works [3].

2 Literature Review

2.1 Video Stylization Models

In recent years, the task of image stylization has been dominated by neural networks, particularly neural style transfer based on GANs [3,4] and Diffusion models [12]. Although such methods can create aesthetically pleasing style transfer whilst preserving the semantic information of the images, they are not directly applicable to stylizing videos because they are not temporally consistent, and are prone to creating flickering artifacts [9]. Optimiation based methods such as [9] use an optical flow based temporal consistency loss to remove these artifacts. However, such methods are very slow, sometimes taking multiple minutes for each frame [1].

Feed-forward network based methods such as [1] speed up video stylization significantly, however they still only manage to achieve 16 frames per second using a GPU. Fast Video Multi-Style transfer [2] improves the speed further by removing the need to compute optical flow at inference time, but this method still requires a reasonable GPU to run the model in real-time at an acceptable resolution.

All of the methods mentioned above utilise deep neural networks, which make them computationally intensive and challenging to deploy in low power portable devices. These networks compute global features on each frame, and then use these features when stylizing the pixels. This prohibits parallel computation for all the pixels until the global features have been computed; computing all hidden features independently for each pixel would result in many redundant computations, and is likely infeasible due to the large number of operations.

2.2 NCAs for Texture Synthesis

[6] first proposed a method to leverage NCAs for texture synthesis. They did this by training NCAs to reproduce the general appearance of a reference texture, and showed that NCAs are capable of producing highly robust and consistent textures. Since then, other works have extended this framework to train texture synthesis NCAs with even more complex behaviours. For example, although NCA generation is inherently unstructured and uncontrollable, [8] was able to

achieve post-training control over the motion of the generated textures, thus demonstrating the potential of NCAs for real-time, dynamic texture synthesis. To our knowledge however, no existing works have achieved an NCA which can dynamically generate textures conditioned on an input image.

NCAs by design evolve their states in small increments, and the next states are computed autoregressively which makes the changes temporally consistent. Furthermore, each cell is updated based only on its own state and the states of its neighbours, which can be computed independently for each cell. Thus, NCAs can be effectively parallelised for deployment on any hardware that supports graphics shading libraries such as OpenGL, Vulkan, CUDA and DirectX.

3 Implementation

3.1 Conditional NCA Architecture

Basic NCA: NCAs consist of a grid of cells, each with a C-dimensional vector representation. Thus, at time T, the NCA state is represented by $S^T \in \mathbb{R}^{H \times W \times C}$, where $H \times W$ is the grid size. The NCA state evolves at each timestep in two steps: a neighbourhood *perception*, and a stochastic *update*.

In the neighbourhood perception step, a set of fixed, depthwise 3×3 convolutions are applied to the NCA state, such that each cell aggregates information from its immediate neighbours. In order to preserve spatial dimensions, a padding scheme is applied. Let $s^T_{i,j} \in \mathbb{R}^C$ represent the state of the cell at the location (i, j) in the grid. The perception vector $z^T_{i,j}$ for a cell is determined by:

$$z^T_{i,j} = Concat(s_{i,j}, \nabla_x \mathbf{S}^T|_{ij}, \nabla_y \mathbf{S}^T|_{ij}, \nabla^2 \mathbf{S}^T|_{ij}) \tag{1}$$

where $z^T_{i,j} \in \mathbb{R}^{4C}$. These values correspond to the outputs of the identity, Sobel-x, Sobel-y and Laplacian filters. The update for each cell is then determined by first passing this perception vector through a small multilayer perceptron (MLP), before multiplying by a random binary variable M to add stochasticity. The MLP consists of two linear layers with a ReLU activation in between. The first layer projects the $4C$-dimensional perception vector into a hidden dimension h, whilst the second layer projects this back to a C-dimensional representation. The update is then added to the original state s^T to obtain the next state s^{T+1}. More formally, the next cell state is determined by:

$$s^{T+1}_{i,j} = s^T_{i,j} + \text{MLP}(z^T_{i,j}) \odot M \tag{2}$$

Figure 1 pictorially represents the *perception* and *update* steps, shown in the red and blue boxes respectively.

The initial NCA state S^0 is filled with zeroes, and updated iteratively according to the steps outlined above. The first three dimensions of the NCA state correspond to the RGB values for each cell, whilst the remaining dimensions represent hidden states for the MLP to encode additional information.

DyNCA: DyNCA makes two main modifications to the basic NCA architecture described above, namely implementing multi-scale perception, and adding a positional encoding for each cell.

The purpose of multi-scale perception is to capture longer range communication between cells, because with single-scale perception, it takes many steps for information to flow between far away cells. This is achieved by applying the same perception described above on a pyramid of downsampled NCA states, upsampling the results to the original dimension, and then aggregating the perception at various scales for each cell. The authors show that this can help preserve appearance fidelity, improve stability, and make the training less sensitive to hyperparameters [8].

The positional encoding meanwhile is introduced to make each cell aware of its global position in the grid. This is implemented by calculating an x and y position encoding for each cell, and concatenating these with the perception vector z, yielding a $(4C+2)$-dimensional vector. They show that this positional encoding is effective in maintaining motion consistency and accuracy.

Our Implementation: Since our aim with this work is to stylize videos in real time, having an efficient NCA is of high importance. Considering that DyNCA is able to effectively synthesis textures in real time, using a similar architecture without introducing too many additional overheads would allow us to achieve our goal. Thus, our architecture makes only minor modifications to the DyNCA setup.

Firstly, for simplicity and performance reasons, we discard the multi-scale perception and opt for single-scale perception, as is done in the basic NCA. Secondly, instead of a 2-dimensional positional encoding, we concatenate a 3-dimensional conditioning vector to the perception vector z for each cell. This conditioning vector is calculated by applying the same filters as in the perception step outlined above – barring the identity filter – on the grayscale version of the input image we want to condition on, before applying a non-linearity. Thus, since the filters are edge detection filters, we are essentially conditioning the NCA on the transformed edge map of the input image. More formally, at time T we have:

$$Z^{'T} = Concat(Z^T, \sigma(\nabla_x \mathbf{F}^T), \sigma(\nabla_y \mathbf{F}^T), \sigma(\nabla^2 \mathbf{F}^T)) \tag{3}$$

where $Z^{'T} \in \mathbb{R}^{H \times W \times (4C+3)}$ is the new input to the MLP, $Z^T \in \mathbb{R}^{H \times W \times 4C}$ is the original perception vector, $\mathbf{F}^T \in \mathbb{R}^{H \times W}$ is the current frame to condition on in grasycale format, and σ is a non-linear function applied elementwise. In practice, we simply take the mean of the RGB channels to obtain the grayscale image, and use $\sigma = \tanh$ to scale the edges between $[-1, 1]$. A pictorial depiction of our NCA setup is shown in Figure 1, with the conditioning step and edge filters shown in the purple and green boxes respectively.

We make the active choice to exclude the grayscale information from the conditioning vector for a couple of reasons, largely informed by the preliminary results we observed when doing this. Firstly, training with grayscale information can make the NCA more sensitive to the intensity of the conditioning image. For

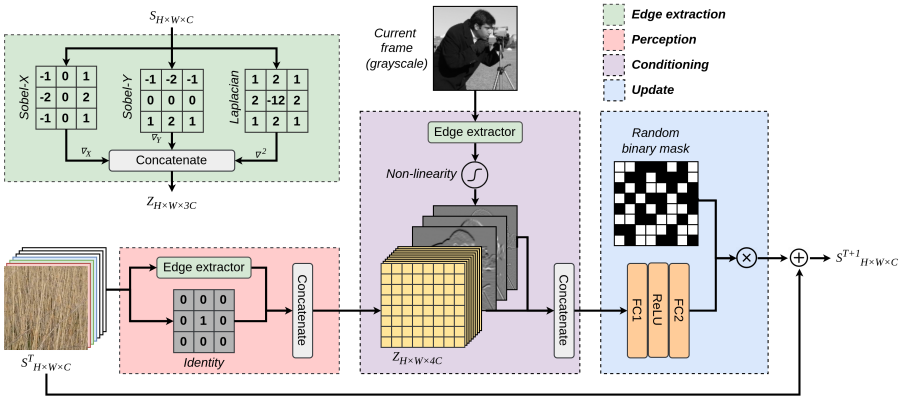

Fig. 1: Illustration of a single step of our conditioned NCA. Given an input state $S^T \in \mathbb{R}^{H \times W \times C}$ at time step T, we apply a perception layer consisting of 4 depthwise convolutions, where each cell perceives gradient information in its local neighbourhood. The edge maps of the current frame (or any arbitrary conditioning image) are passed through a non-linearity and concatenated to this result. Each cell then calculates an update based on its perception vector and conditioning, as parameterized by a small MLP, which is stochastically added to the input state to produce the next state S^{T+1}.

example, if the NCA learns to stylize based on the gray levels, a simple change in the lighting of a scene can result in drastically different NCA outputs. For our purposes however, this would be undesirable in most cases, since we mainly want to preserve the overall content and structure of the input, which is more directly informed by the edges rather than intensities. Furthermore, it could also hurt the generalisability of the NCA. As outlined in the next section, we only use a handful of images during training time, yet the NCA is able to generalise well to arbitrary unseen images. If however we include grayscale information, if there is a certain bias in the the intensity distribution of our training images, the NCA may not generalise as well to out of distribution intensity levels. We also decided to discard the positional encoding for similar reasons, since it makes the NCA more prone to learning artefacts present in the training images and reproducing them on unseen images.

Ultimately, even when comparing to DyNCA with single-scale perception, our method introduces minimal overhead. In terms of parameters, since we have one additional dimension as input to the first MLP layer, we have an additional h parameters where h is the hidden size of the layer. In terms of compute, the positional encoding calculation is replaced with three convolutional filters and an elementwise non-linearity, which is not significantly more computationally expensive. Thus, we are able to add conditioning to the DyNCA for very little cost.

3.2 Training Setup

For training our conditional NCA, we must define one RGB target style image $Y^s \in \mathbb{R}^{H \times W \times 3}$, as well as a set of N grayscale target content images $\mathcal{Y}^c = \{Y_1^c, Y_2^c ..., Y_n^c\}$ where $Y_n \in \mathbb{R}^{H \times W}$, all cropped and resized to the grid size of the NCA. We also specify a target motion vector field $V^t \in \mathbb{R}^{H \times W \times 2}$, as is done with DyNCA, however only to maintain some motion in the stylized videos for effect rather than for the explicit purpose of controlling the motion during synthesis.

At each iteration of the training, we randomly sample a batch size b of the target content images to condition our NCA generations on. We run K steps of NCA generation before extracting the first three channels of the NCA states to obtain b generated stylized images/frames, $X^g \in \mathbb{R}^{H \times W \times 3}$. We then compare these generations to the target style, content, and motion fields outlined above to calculate an appearance loss $\mathcal{L}_a$, content loss $\mathcal{L}_c$, and motion loss $\mathcal{L}_m$ respectively to optimize the MLP with.

Thus, our training objective is almost the same as in DyNCA, except that we introduce an additional content loss term. To calculate the content loss, we use the formulation originally proposed in [3]. The paper found that whilst the feature maps produced in the earlier layers of a pretrained VGG network capture detailed content representations, feature maps at deeper layers capture the high-level content of the input image. Thus, by comparing the representations of our generated image against the target image at deeper layers, we can effectively compute the similarity in their high level content, which is what we aim to preserve. In line with the original proposal, for this we calculate the feature maps for each image in the 'conv4_2' layer, and calculate the mean squared error (MSE) between the representations.

For calculating the appearance loss $\mathcal{L}_a$ and motion loss $\mathcal{L}_c$, we use the exact same schemes described in the DyNCA paper [8]. We also use an overflow loss $\mathcal{L}_o = |X^g - \max(-1, \min(X^g, 1))|$ to encourage the NCA states to remain between $[-1, 1]$ and prevent divergence.

Our final loss is then determined by the weighted sum of these four loss terms: $\mathcal{L} = \omega_a \mathcal{L}_a + \omega_c \mathcal{L}_c + \omega_m \mathcal{L}_m + \omega_o \mathcal{L}_o$. A pictorial representation of our training objective is shown in Figure 2.

4 Results

Unless otherwise specified, we use the following hyperparameters for all our experiments: cell state dimension $C = 12$, MLP hidden size $h = 96$, circular padding, batch size $b = 2$, $K = 24$ NCA steps, and learning rate of 0.001. The NCAs are trained at 256×256 resolution for 2000 iterations each. We find that using lower resolutions for training loses too much detail in the target content and style images for the NCA to effectively learn. For example, finer-grained textures in the style image may no longer be perceptible at lower resolutions, meaning the NCA will not be able to learn the relevant textures. Thus, we train

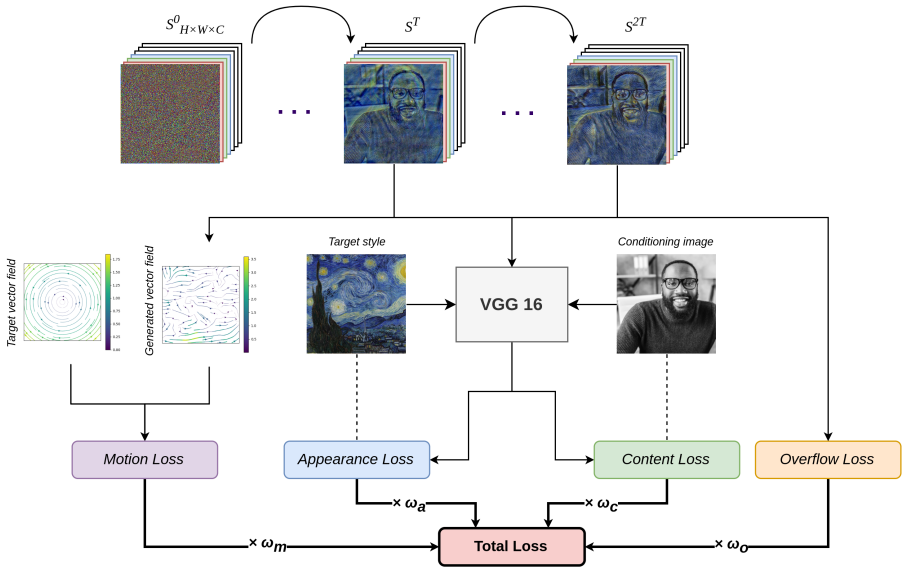

Fig. 2: Losses

at higher resolution to obtain better results, whilst maintaining the ability to generate at lower (or higher) resolutions post-training as required. Due to the high resolution, and the need to track states and gradients over many timesteps for each update, the training process requires a significant amount of memory on a GPU. As such, all training was performed on an NVIDIA V100 GPU with 32GB of VRAM.

Additionally, we use a fixed set of only $N = 6$ target content images for training, as this is already enough for the model to generalise to unseen targets. The images we used can be found in Figure 7 in the Appendix.

4.1 Overall Results

We primarily perform qualitative assessments of our method by using our trained conditional NCAs to stylize various videos. Figure 3 demonstrates the results of an NCA trained on an example style image: Vincent van Gogh's "Starry Night". A frame sampled from the original video is shown alongside the output of the NCA conditioned on this frame.

As we can see, the NCA successfully captures the overall color palette and texture of the "Starry Night" painting, whilst maintaining the structural content of the original video frame. By transferring the swirling, dynamic textures and distinct color scheme of the painting onto the video frame, our model demonstrates its ability to apply a consistent style to previously unseen video frames without the need for a large training set. This result showcases the robustness

Fig. 3: Results of an NCA trained using the Starry Night painting (a) as a style image. (b) shows the original frame, whilst (c) shows the stylized frame. We see that the overall texture and dominant colors of the reference image are well preserved in the stylized image, along with the content from the reference frame. The NCA was trained with the following loss weights: $\omega_a = 8, \omega_c = 0.1, \omega_m = 8, \omega_o = 1000$.

and generalisability of our approach, indicating that NCAs can mimic complex artistic styles while preserving key structural elements from the original content.

However, it is important to note that the stylization captures the overall color and texture, but it is not semantically meaningful. Since the NCA is conditioned only on edge maps, the model does not preserve any semantic information about the input image. This means that while the textures and colors are accurately replicated, specific objects or scenes in the input video may not be consistently represented in the stylized output.

Our NCAs generally perform better on finer-grained textures. For instance, the "Starry Night" painting features intricate patterns and detailed textures, which the NCA effectively captures and replicates. In contrast, when the style image is more blocky or contains larger, less detailed patterns, the NCA struggles to capture the desired effect. This limitation may be due to the single-scale perception, which only considers the immediate neighbors, making it more effective for fine-grained images. Incorporating multi-scale perception could help address this issue by allowing the NCA to better capture broader and less detailed patterns, thereby improving performance with such style images.

4.2 Sensitivity to Hyperparameters

With well chosen hyperparameters we are able to produce robust NCAs which can effectively stylize videos whilst maintaining content, however we note that the quality of the NCA is quite sensitive to the choice of hyperparameters, particularly the loss weights. An illustrative example is shown in Figure 4, which shows the results of three NCAs trained using different loss weights to stylize videos with a crumpled-paper texture. We see that depending on the loss weights, the style and/or motion may be too strong resulting in a loss of content, or the

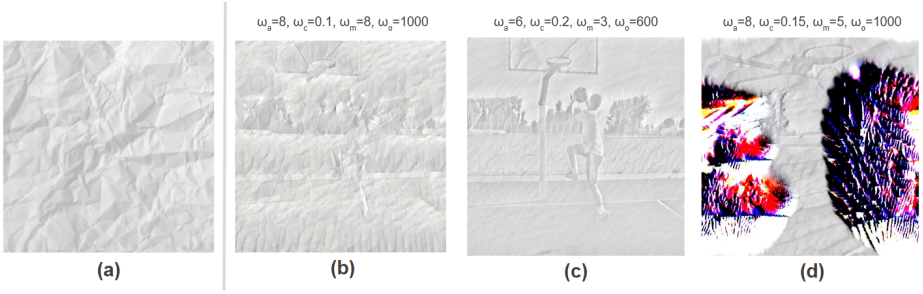

Fig. 4: Sensitivity of the NCA stylization to different hyperparameter configurations using a crumpled paper texture style. (a) The reference style image: crumpled paper. (b) With $\omega_a = 8$, $\omega_c = 0.1$, $\omega_m = 8$, $\omega_o = 1000$, the style and motion are strong but the content is poorly preserved, leading to some divergence. (c) With $\omega_a = 6$, $\omega_c = 0.2$, $\omega_m = 3$, $\omega_o = 600$, the content is overly dominant, resulting in weak stylization and motion. (d) With $\omega_a = 8$, $\omega_c = 0.15$, $\omega_m = 5$, $\omega_o = 1000$, the balance between style and content is initially good but the NCA becomes unstable and diverges over time.

inverse effect where the content is too strong and there is not enough stylization or motion in the NCA. Even when the content and style are balanced well, some configurations can still result in an unstable NCA which eventually diverges.

Additionally, with certain hyperparameter configurations, we observe that the content and style in the NCA seem somewhat disjoint. In other words, the video appears more as if there is some recolored content video in the background, and an independent texture synthesis overlayed, which is an undesired effect when trying to achieve an integrated stylization of the content. Although we are not sure of what exactly causes this effect, we hypothesise it could occur when the content loss is too strong. In general however, we notice that changing the loss weights can help alleviate this issue.

Through our experiment, we also observe that there is no one set of loss weights works well for all style images. Sometimes we can apply our intuition to pick appropriate weights; for example for more coarse textures, often a higher content loss is required, otherwise too much of the details are lost whilst reproducing the style. In many other cases however, the results from tuning the weights can be somewhat unintuitive, and relatively small changes in weights can cause the NCA to become unstable. As such, one of the drawbacks of this sensitivity to hyperparameters is that we must carefully tune the loss weights for each style individually, which can be time and resource consuming.

4.3 Extensions

Controlling Motion for Different Styles Although our main aim is not to control the motion in the NCAs, our training setup allows us to customize the learned motion for different styles. This capability adds an additional layer of versatility to our method, enabling us to produce more coherent and contextually

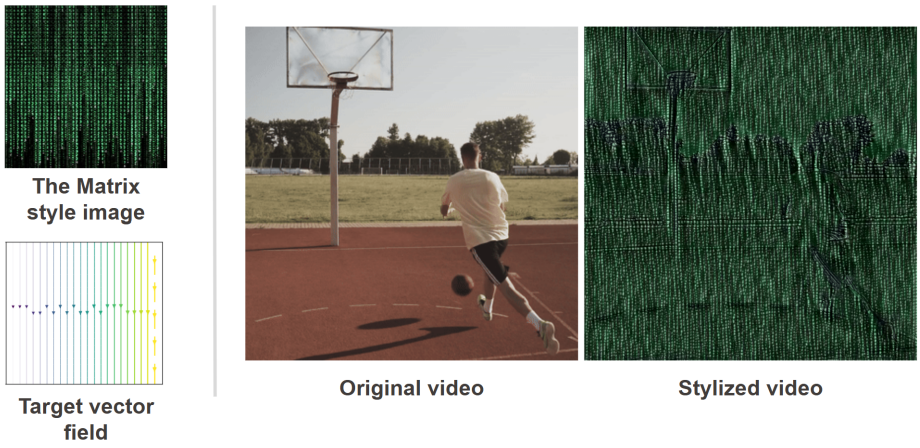

Fig. 5: Stylization of a video frame using the Matrix style image. The image on the left shows the Matrix style reference image characterized by falling green code. The target vector field below it illustrates the downward motion direction. The center image is a frame from the original video. The right image shows the video frame stylized using the NCA trained with the Matrix style and the downward motion vector. This specific motion direction enhances the visual coherence of the stylized output, aligning it more closely with the intended aesthetic of the Matrix style.

appropriate stylizations by tailoring the motion characteristics to match the artistic style of the reference image.

For instance, with a matrix style NCA, we can train the model to follow a downward motion vector rather than a more random motion. This specific motion direction enhances the visual coherence of the stylized output, aligning it more closely with the intended aesthetic of the style. The downward motion can create a more unified and flowing visual effect, which is particularly suitable for styles that inherently suggest a direction or flow. Figure 5 illustrates this concept.

Multi-Scale Perception Another enhancement we can incorporate is the multi-scale perception proposed in DyNCA. Although we do not use it for performance and simplicity reasons, preliminary results indicate that multi-scale perception could be more effective at capturing the style and textures in the reference image, beyond just the low-level texture and overall color information.

As shown in Figure 6, multi-scale perception allows the NCA to consider information at multiple scales, leading to a richer and more detailed representation of the style image. This approach also introduces motion more effectively into the NCA, although in some case it may result in excessive motion if not tuned properly.

Multi-scale perception could be especially useful for offline video stylization, where efficiency is less of a concern, and the highest quality style transfer is

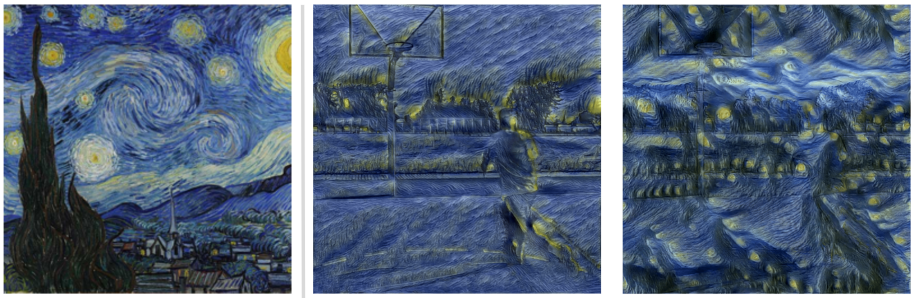

Fig. 6: Comparison of NCA stylization using single-scale and multi-scale perception. (a) The reference style image: Starry Night. (b) Stylization result using single-scale perception, capturing overall color and texture. (c) Stylization result using multi-scale perception, showing enhanced detail and more dynamic textures. Multi-scale perception better captures the intricate patterns of the style image.

desired. By capturing the overall patterns of the style image more accurately, multi-scale perception can enhance the visual quality of the stylized output.

Additionally, since the original paper notes this multi-scale perception makes the NCA training more stable and less sensitive to hyperparameters [8], this could potentially alleviate some of the issues we had with having to carefullt tune our loss weights.

5 Conclusion

In this work we introduce a method to train a conditional NCA which can react to a given input image, and demonstrate that this can be used to achieve generalisable and robust video stylization. Building on top of the DyNCA framework, we show that simply concatenating the edge maps of the input image to the perception vectors of the cells, whilst imposing an additional loss term during training, can effectively achieve this objective. At the same time, we are able to maintain many of the desirable properties of the original DyNCA, all with minimal parameter and performance overheads, thus enabling real time and somewhat controllable stylization.

Limitations. Since we are only conditioning on edge maps, the NCA does not receive any color or semantic information about the input image, meaning the stylization is not semantically meaningful; it simply preserves the content of the input image whilst mimicking the general textures and colors of the style image. Moreover, the performance of the NCA is quite sensitive to the choice of hyperparameters, with many configurations being prone to overflow which leads to the NCA states diverging. There is also not a set of hyperparameters which works well across all different textures. Rather, it is necessary to tune the

critical hyperparameters such as loss weights for each style individually in order to achieve a stable and effective NCA.

Future Work. There are many different avenues we plan to explore further with this work. First, we hope to introduce an additional control for the level of style applied to the video, such that post-training we can interpolate between applying no style and fully stylising the input video. Next, we plan to experiment further with using multi-scale perception [8] to improve the style and motion captured by the NCA. Finally, we hope to evaluate our method more formally, both quantitatively and qualitatively. Until now, the evaluation has been highly subjective and qualitative within the development team, however a more systematic approach could allow us to better evaluate our NCAs.

References

1. Chen, D., Liao, J., Yuan, L., Yu, N., Hua, G.: Coherent online video style transfer. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 1105–1114 (2017) 2
2. Gao, W., Li, Y., Yin, Y., Yang, M.H.: Fast video multi-style transfer. In: Proceedings of the IEEE/CVF winter conference on applications of computer vision. pp. 3222–3230 (2020) 2
3. Gatys, L.A., Ecker, A.S., Bethge, M.: Image style transfer using convolutional neural networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 2414–2423 (2016) 2, 6
4. Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., Aila, T.: Analyzing and improving the image quality of stylegan. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2020) 2
5. Mordvintsev, A., Randazzo, E., Niklasson, E., Levin, M.: Growing neural cellular automata. Distill **5**(2), e23 (2020) 1
6. Niklasson, E., Mordvintsev, A., Randazzo, E., Levin, M.: Self-organising textures. Distill **6**(2), e00027–003 (2021) 1, 2
7. Pajouheshgar, E., Xu, Y., Mordvintsev, A., Niklasson, E., Zhang, T., Süsstrunk, S.: Mesh neural cellular automata. arXiv preprint arXiv:2311.02820 (2023) 1
8. Pajouheshgar, E., Xu, Y., Zhang, T., Süsstrunk, S.: Dynca: Real-time dynamic texture synthesis using neural cellular automata. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 20742–20751 (2023) 1, 2, 4, 6, 11, 12
9. Ruder, M., Dosovitskiy, A., Brox, T.: Artistic style transfer for videos. In: Pattern Recognition: 38th German Conference, GCPR 2016, Hannover, Germany, September 12-15, 2016, Proceedings 38. pp. 26–36. Springer (2016) 2
10. Sandler, M., Zhmoginov, A., Luo, L., Mordvintsev, A., Randazzo, E., et al.: Image segmentation via cellular automata. arXiv preprint arXiv:2008.04965 (2020) 1
11. Sudhakaran, S., Najarro, E., Risi, S.: Goal-guided neural cellular automata: Learning to control self-organising systems. arXiv preprint arXiv:2205.06806 (2022) 1
12. Zhang, Y., Huang, N., Tang, F., Huang, H., Ma, C., Dong, W., Xu, C.: Inversion-based style transfer with diffusion models. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 10146–10156 (2023) 2

6 Appendix

Fig. 7: Target content images used for training our conditional NCAs.