INFO8010:
Consistent Video Neural Style Transfer

Florian DELCOUR[1] and **Sacha LEWIN**[2]
[1] *florian.delcour@student.uliege.be (s181063)*
[2] *sacha.lewin@student.uliege.be (s181947)*

I. INTRODUCTION

During the last decade, thanks to the ease of access to a large amount of data and the increase in computational power, we have seen a lot of astonishing applications which would have never existed without deep neural networks. One of these applications is Neural Style Transfer (NST), which consists in extracting the style of an image (usually a painting) and the content of another image, in order to generate a new one (Figure 1).

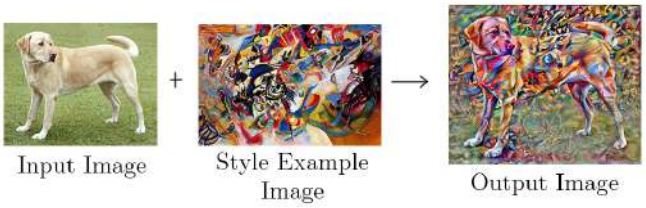

FIG. 1: Example: Painting a dog with Kandinsky's style [1]

This process, which would take enormous amounts of time for a human to produce, especially for videos where painting new frames one by one based on the original ones seems almost impossible in practice (although it was already done in rare cases [2]), is also very hard to produce using a computer, considering the artistic nature of the problem. However, thanks to recent research in neural image processing, it is possible to extract content and style representations of an image in order to produce a new image, blending them together.

First, we implement standard neural style transfer to still images to demonstrate and analyze the original technique that was presented by Gatys et al. [3]. Afterwards, we try to adapt this technique to produce stylized video content. This task is much harder, as simply applying naively the still image algorithm frame by frame produces artifacts due to instability in the optimization with respect to the image. Indeed, small changes in the content image can produce large changes in the final stylized image. We demonstrate various techniques and tuning to try and produce more consistent stylized videos, while staying based on the method by Gatys et al. [3]. These are strongly inspired by Ruder et al. [4].

Afterwards, we present another technique, based on real-time transfer by Johnson et al. [5] in 2016, that was showcased in 2018 by Rainy and de Berker [6], which shows very good performance in real-time but requires long training before being able to produce stylized images or videos.

Our final implementation includes the first part, with many configurable parameters, and a notebook for running the second part with the Fast Style Network.

II. RELATED WORK

In 2015, Gatys et al. [3] present the first real neural style transfer technique, using deep learning. Their technique is only applied to still images, but is the foundation of NST, as their technique influences all the other works done later. They show how one can use intermediate neural representations of images in state-of-the-art image classification networks, such as VGG19, in order to extract content and style metrics from an image. After extracting the style and content of the two given images, they show how the generation of the blended image can be formulated as an optimization problem that can be solved using gradient descent by minimizing some loss based on content and style, which is indeed completely differentiable with respect to the image being generated.

In 2016, Ruder et al. [4] try and adapt the approach in order to produce stylized videos, by modifying the initialization and the loss function. They make use of optical flow, which tries to estimate, from an image to another, where things move. For initialization, they first try with starting optimization from the previous stylized frame, warped using the optical flow, which thus gives a sort of estimation of the next stylized frame. They later develop short-term and long-term temporal consistency losses, which try to penalize deviations from the warped image areas where the optical flow is consistent. Their long-term loss simply takes more frames into account than just the previous one, as does the short-term loss. They obtain great results. However, an issue with these is the time taken to stylize a video, which can be quite long, especially for long videos. Producing a stylized frame can already take some time with the original method, but becomes even longer with this as it needs to estimate optical flows, which is a heavy task.

Also in 2016, Johnson et al. [5] show impressive

results by training a network to solve the optimization problem a lot faster, by about 3 magnitude orders. The great advantage of this method is obviously being able to stylize any image very fast once the network is trained, which can be especially useful for stylizing videos, but the disadvantage is that it requires training one network per style image, and this training requires quite some time, whereas the original technique could take as input any arbitrary content and style and optimize the generated image directly. The method presented by Johnson et al. is therefore especially useful for stylizing a lot of images with the same style, for example for videos.

Their method focused on still images, but as mentioned is very interesting for producing stylized videos. In 2018, Rainy and de Berker [6] combine the methods of Ruder and Johnson in order to produce real-time and consistent video neural style transfer. As they mention, their approach is to consider that "*temporal instability and popping result from the style changing radically when the input changes very little*". They add a small amount of noise to images during training and impose a loss that minimizes the difference between the stylized original images, and stylized noisy images. They thus train a network for stable NST.

Some other modern approaches like [7], [8] and [9] perform similar work (and quite fast) using GANs [10]. We did not explore these methods and focused on the other ones, based on the work by Gatys et al.

III. METHODS

A. Original Method and Definitions

First, we implement the original neural style transfer method, using a pre-trained version of VGG-19 [11] (on ImageNet[12]), as done by Gatys et al. The VGG network is composed of a convolutional part, with 5 blocks, then a classification part with dense fully connected layers. We are interested by the first part, which creates a high-level representation of the image. We want to extract style and content metrics from intermediate layers among these.

Content is simply represented by the values of intermediate feature maps of a chosen layer. *Style representation* is more complex, and uses correlations between the filters of a same layer. These are computed using Gram matrices [13], which contain the inner products of two feature maps in a given layer. Both these processes are differentiable.

After defining these, one needs to choose the layers from which these representations are extracted. Based on [3], we choose the following layers, shown in Table I

Layer	Information	Represents
block1_conv1	kernel=3, 64 channels	**Style**
block1_conv2	kernel=3, 64 channels	
block1_pool	Max pooling (by 2)	
block2_conv1	kernel=3, 128 channels	**Style**
block2_conv2	kernel=3, 128 channels	
block2_pool	Max pooling (by 2)	
block3_conv1	kernel=3, 256 channels	**Style**
block3_conv2	kernel=3, 256 channels	
block3_conv3	kernel=3, 256 channels	
block3_conv4	kernel=3, 256 channels	
block3_pool	Max pooling (by 2)	
block4_conv1	kernel=3, 512 channels	**Style**
block4_conv2	kernel=3, 512 channels	**Content**
block4_conv3	kernel=3, 512 channels	
block4_conv4	kernel=3, 512 channels	
block4_pool	Max pooling (by 2)	
block5_conv1	kernel=3, 512 channels	**Style**
block5_conv2	kernel=3, 512 channels	
block5_conv3	kernel=3, 512 channels	
block5_conv4	kernel=3, 512 channels	
block5_pool	Max pooling (by 2)	

TABLE I: VGG-19 Layers used for style and content representations

We thus create a network that contains the *feature* layers of VGG-19, in order to extract the content and style representations. With the image from which to extract the representations as the input, we feed-foward it iteratively through each VGG-19 layer, and each time analyze the name of the layer encountered, if it is one of the layers that we want to extract from, we save the results, and keep feeding forward. Once all the layers have been met, we simply stop feeding forward (therefore it always stops after the *block5_conv1* layer considering our choices).

Now, we can extract the style representation of our style image, and the content representation of the content image. Once this is done, the optimisation problem of the image being generated can be formulated. It simply consists in minimizing a loss function, which is the weighted sum of a style loss and a content loss:

$$\mathcal{L} = w_c \cdot \mathcal{L}_{\text{content}} + w_s \cdot \mathcal{L}_{\text{style}}$$

where w_c and w_s are respectively the content and style weights for trading off content and style in the blended image.

The *content loss* $\mathcal{L}_{\text{content}}$ is the mean-squared error (also called "*perceptual loss*" for images) between the content representation of the content image, and the current content representation of the image being generated.

The *style loss* $\mathcal{L}_{\text{content}}$ is similar, it is also the mean-squared error (also called "*perceptual loss*" for images) between the style representation of the style image, and the current style representation of the image being generated. However, there are several style layers, we therefore take the average of all these MSEs. As in the paper by Gatys et al., all style layers have the same weight in this computation.
Computing this total loss is a differentiable process, with respect to the image being generated. Indeed, the convolutional network is naturally differentiable. The computation of the Gram matrices also is. Same goes for the losses, which are MSEs, also naturally differentiable. This means we can optimize, with gradient descent, the image in order to minimize the distance of its style with the style of the style image, and the distance of its content with the content of the content image.

B. Initialization

For *initialization*, Gatys et al. start from a white noise image and perform gradient descent. Although this gives good results, a way to obtain visually pleasing results with less iterations is starting from the content image. The process therefore tries to "paint" it with the style without changing the content too much.

Let us note that images are downsized before doing anything, in order to speed up computations, as large images can take a very long time to run. We do not make them too small either, but we put an upper limit on the larger dimension of the content and style images, and rescale accordingly while keeping the original scale ratio.

C. Stopping Criterion

The *stopping criterion* we use is a user-defined amount of iterations, because it can highly affect the final image quality, and depends on the image resolution, the style, etc. Therefore, one should adjust this when optimizing an image, depending on the visual result they seek.

D. Optimizer

We try two different optimizers. L-BFGS [14] and Adam [15] (learning rate of 0.02, $\beta_1 = 0.99$, $\beta_2 = 0.999$, $\epsilon = 0.1$) which are two popular choices [16], [17], [4] and [5] for Neural Style transfer. L-BFGS being what seems to be the best choice, from literature [16]. These are compared in our qualitative analysis. As a note, everything in this project is implemented using PyTorch with CUDA.

E. First Approach to Video Consistency

Afterwards, for applying neural style transfer to videos, we can obviously first naively apply our still-image implementation to each frame independently, but this causes artifacts that vary fast (sort of flickering) between each frame, even though frames are almost no different. This gives most of the time an unpleasant result, as it looks like a mask filter was simply put onto the video.

Our first idea we implemented is using the previous (stylized) image as the style image, instead of the original style image, starting from the second frame. Following disappointing results, it was very quickly dropped for temporal initialization, explained below.

F. Temporal Consistency with Optical Flow

As done by Ruder et al., we implement a short-term consistency loss term that penalizes deviations from the optical flow of the video. The optical flow is an estimation of "where each thing has moved between a frame and the next one". They used optical flow estimations using deep learning methods such as DeepFlow [18]. We do not use these, but rather compute the optical flow between two frames using OpenCV [19].

1. Temporal Loss

Starting from the second frame, a third term is added to the loss: $w_T \mathcal{L}_{\text{temp}}$, with w_T the associated weight. The previous stylized frame is kept, and is *warped* using the optical flow computed from the two unstylized frames. The loss $\mathcal{L}_{\text{temp}}$ is then the MSE between the warped image, and the image being generated. Let us note that Ruder et al. only consider some pixels for this temporal loss, depending on disocclusions and motion boundaries that are detected using the optical flow estimated. We did not implement such things and considered the optical flow everywhere for the sake of simplicity.

2. Initialization with Optical Flow

Another method [4] to stabilize the video is to start the optimization of a frame (starting from the second one) from the warped previous stylized image. This means the same warped image used for the temporal loss. This makes these two techniques similar, as we make the image deviate less from this warped image. It is used here as the new initialization instead of the content image. We call this *"temporal initialization"*.

G. Further Details

All weights are also "user-defined", and are not unique for all images. We made them vary a lot for each example. Indeed, depending on the style, and on the resolution, different weights should be considered and may give completely different results, that might be less or more visually pleasant.

We also adapt our *stopping criterion* to allow for stopping earlier if the loss has not changed by more than 0.01% for 50 iterations, inspired by Ruder et al.

H. Fast Style Transfer for Video

As previously introduced, the original method proposed by Gatys et al. requires a lot of computational resources and thus time in order to stylize even a single frame, making it intractable for videos. To solve this problem, we based our second implementation on the technique presented by Johnson et al. Instead of explicitly solving the optimization problem for each frame, we train a feed-forward network (one per style image) that learns to solve this problem using perceptual loss functions. This network will directly output the stylized image from the original image as input. Once the network is trained, we can perform neural style transfer on any content image in quasi real-time, depending on the hardware and also the video resolution. However, Johnson et al. limited themselves to still images. We manage to achieve consistency by adding a temporal loss following Rainy and de Berker's idea [6].

1. *Differences with the Original Method*

As in the original method, we use high-level representation of image obtained from pre-trained VGG-19 to compute perceptual losses $\mathcal{L}_{\text{style}}$ and $\mathcal{L}_{\text{content}}$, this time from different layers (`relu1_2, relu2_2, relu3_3, relu4_3` for style loss, and `relu3_3` for content loss). The VGG-19 network is always frozen during training. Compared to the original method, we include to the loss term the total variation loss $\mathcal{L}_{\text{TV}}$ [20] with weight w_{TV} following Johnson et al. This term allows to penalize high variations (along spatial dimensions) in order to make the image smoother and avoid too much variation in the style. The temporal loss $\mathcal{L}_{\text{temp}}$, using optical flow, was abandoned because of the lack of good results and the computational resources it requires. Instead, we use the key idea provided by Rainy and de Berker, that is to create a copy of our content image and add some noise to it. Then we minimize the distance between the stylized versions of original and noisy images to reduce the temporal instability that produces popping effects on stylized video.

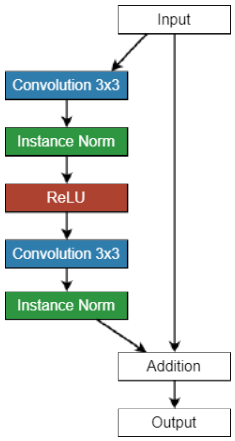

FIG. 2: Residual block diagram

2. *Architecture of the Fast Style Network*

The architecture of our fast style transfer is highly based on the one provided by Jonhson et al. and is shown in Table II. It is composed of 11 blocks: 3 downsampling blocks, 5 residual blocks and 3 upsampling blocks. Downsampling and upsampling blocks are implemented thanks to strided convolutions and transposed convolutions respectively. More precisely, all convolution blocks (resp. transposed convolution blocks) are made of a 2D convolutional layer (resp. transposed conv. layer), a 2D instance normalization [21] and a ReLU activation. Except first and last layers which use 9×9 kernels, all convolutional layers and transposed ones use 3×3 kernels. Despite the use of batch normalization by Johnson et al., instance normalization is preferred as it results in significant improvement, as shown by Ulyanov et al. [21]. The body part of the network are 5 residual blocks taken from Gross and Wilber's work [22] and is shown in Figure 2. Using residual networks allows to train faster compared to non-residual ones [23].

3. *Training details*

To train the Fast Style Transfer network, we use part 1 of the Visual Genome [24] dataset. We also tried on the COCO [25] dataset. Each of the 63,346 images is resized to 512×512, and we train with a batch size of 2. Let us note that, once the feed-forward network is trained, we can apply neural style transfer to any size of image. We use the Adam optimizer with a learning rate $\gamma = 0.001$, MSEs for perceptual losses and temporal loss. These parameters will be tuned in next section, like the different weights attributed to the different losses, the number of pixels that we randomly choose to add noise along with the maximum range of noise added. Training thus consists in passing the unstylized images through the network which tries to stylize them, compute the loss using representations with VGG-19, and update the

Layer	Activation size
Input	$3 \times 512 \times 512$
Conv block $32 \times 9 \times 9$ (stride=1)	$32 \times 512 \times 512$
Conv block $64 \times 3 \times 3$ (stride=2)	$64 \times 256 \times 256$
Conv block $128 \times 3 \times 3$ (stride=2)	$128 \times 128 \times 128$
Residual block, 128 filters	$128 \times 128 \times 128$
Residual block, 128 filters	$128 \times 128 \times 128$
Residual block, 128 filters	$128 \times 128 \times 128$
Residual block, 128 filters	$128 \times 128 \times 128$
Residual block, 128 filters	$128 \times 128 \times 128$
Conv block $128 \times 3 \times 3$ (stride=2)	$64 \times 256 \times 256$
Conv block $32 \times 3 \times 3$ (stride=2)	$32 \times 512 \times 512$
Conv block $3 \times 9 \times 9$ (stride=1)	$3 \times 512 \times 512$

TABLE II: Architecture of Fast Style Transfer Network. `Conv block` $C \times H \times W$ means that it includes one 2D convolution layer of C filters with size $H \times W$.

Fast Style Network with that.

Also, it is important to note that even when applying to videos later, frames are thus stylized one by one! The loss is not computed depending on the previous frame, unlike the previous method. All the temporal consistency is actually ensured by stabilizing the output using the noise added. Using a neural network for stylizing instead of the optimization process previously used also allows for more stability in the results.

IV. RESULTS

A. Qualitative Analysis

Although the results can seem quite subjective, some "features"/qualities of the resulting stylized image/video stay quite objective, and allow for a methodical qualitative analysis.

For a still image, it should well "absorb" the style of the given style image, while keeping the good original shape and content of the content image, without too many artifacts or high-frequency variations of style.

For videos, as it is the goal of this project, the main quality is consistency and stability of the stylized image with respect to the original image, while preserving a pleasant to watch video. Two close images should map to two close stylized images.

We present our results in the order of our studies, by starting with still images, then moving on to videos with no temporal improvements, videos with temporal initialization and temporal loss. Finally, we move on to the fast style network architecture, which seems to be the best choice for a final architecture to use.

1. *Still Images*

Using the standard algorithm as described by Gatys et al. already shows very good result for still images. Let us observe some examples, along with some comparisons depending on the content and style weights, the amount of iterations, initialization, and the optimizer.

First, Figure 3 shows our content image for demonstration.

FIG. 3: Our content image: *"Passerelle La Belle Liégeoise"*

We perform neural style transfer on this image, using our "final" default parameters: L-BFGS, 300 iterations and content image initialization. Note that we made the style weight vary for each image to obtain better results, as it influences the visual quality of the result. The content weight is constant at 1, and we only make the style weight vary here considering there are only two terms. Results are shown on Figure 4.

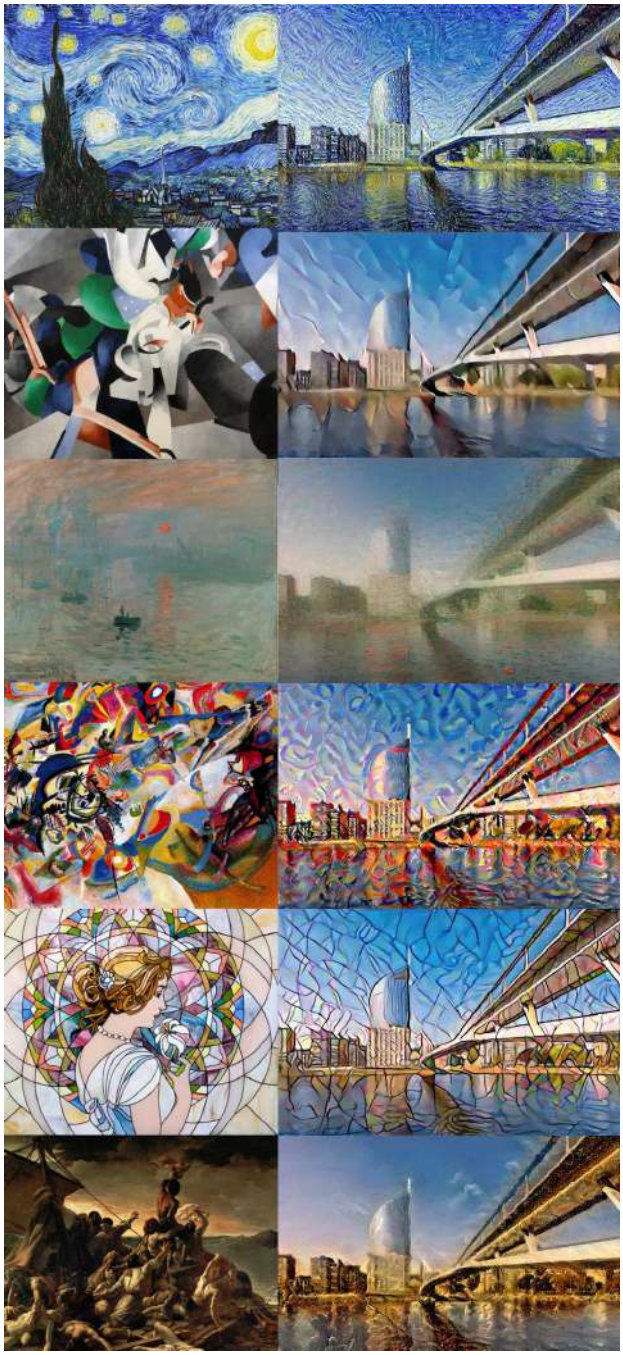

FIG. 4: Style images and the result images, on content image from Figure 3. The styles (and style weight) are in order: "La Nuit Étoilée" by Van Gogh (10^7), "Udnie" by Picabia (10^6), "Impression, soleil levant" by Monet (10^6), "Composition VII" by Kandinsky (10^7), Glass Mosaic (10^7), "Le Radeau de La Méduse" by Géricault (10^6).

As mentioned, the algorithm performs very well, with minimal need for adjusting every hyper-parameter. The style is well absorbed into the content image. Let us note however that some styles perform pretty bad, despite trying various weights. Images with "a lot of content" seem to cause this, as shown on Figure 5. We can see how it does not capture at all the actual style of image, and seems to be based on another style image, even when using a very high style weight (10^{10}). The later shown fast style network performs better on these images, as can be seen in Figure 18.

FIG. 5: Model performs bad when applied to this example japanese anime image.

From now on, let us study how the hyper-parameters (optimizer, iterations, style weight) affect the qualitative result. The same analysis will be conducted quantitatively later, and will focus on the loss convergence. Here, we will only show example results for different values of the hyper-parameters.

First, we compare the optimizers L-BFGS and Adam (learning rate of 0.02, $\beta_1 = 0.99$, $\beta_2 = 0.999$, $\epsilon = 0.1$), along with the number of iterations. We reuse the Van Gogh and glass mosaic images shown in Figure 4 and perform optimization with 20, 100, and 300 iterations with both algorithms. Figure 6 shows the result. As we can see, "stylization" seems to be about the same for the two optimizers for a same number of iterations. However, LBFGS overall gives better and smoother results as can be seen in the third column, especially around the skyscraper. 300 iterations seems to be a good number, and was used in most of our tests successfully.

FIG. 6: Optimizer comparisons. First row: LBFGS, Second row: Adam. Columns are respectively 20, 100, and 300 iterations.

Now, let us analyze how the style weight affects the result. Figure 7 shows for the Passerelle image, 2 styles (Van Gogh, and Hokusai's great wave) for 3 style weights: 10^2, 10^5, 10^{10}. We can see how this weight influences the

stylizing of the image, and how a too big weight can give undesirable results.

FIG. 7: Passerelle image, 2 styles (Van Gogh, and Hokusai's great wave) for 3 style weights: 10^2, 10^5, 10^{10}

2. Video Results

From now on, we show results on videos, using the three big different techniques mentioned:

1. Naive approach: Apply frame by frame. Initialize with content.
2. Temporal loss and temporal initialization, using optical flow.
3. Fast Style Network.

A first video shows the results by applying each of these techniques, along with the original video and the style applied, for 5 different short GIFs, with 3 styles showcased per GIF.

This video can be accessed here: `https://www.youtube.com/watch?v=ZqU1A1hIJZw`

3. Naive Video Approach

As expected and mentioned, videos generated with this technique produce instable results. Although it is sometimes acceptable, such as the church with the "June Trees" style, it stays most of the time not really interesting nor pleasant to look at.

4. Temporal Initialization and Loss

Using this approach, we can see big differences compared to the naive approach. The temporal loss affects the result, but temporal initialization seems to be even a greater factor.

As can be seen in the video, the GIFs now achieve greater stability and consistency in the way the style is applied. However, we can quickly notice that it becomes often hard to pleasantly understand and watch the scene. Indeed, a first problem is about parts that appear in the video but were not present before. Indeed, those were then not computed in the optical flow, considering they appear in the scene, and thus cause issues. This is why Ruder et al. tweaked this technique further. We can identify in most videos these dark areas that show up. Lowering the style didn't really yield better results, the "sticky" behaviour it gives is mostly caused by the use of the temporal initialization. It neither disappears with more iterations per frame. Temporal loss only was quickly abandoned because of yielding no results if not coupled with the initialization.

5. Videos: Temporal Noise-based Loss

Before talking about videos, let us note that the appendix A shows in Figure 17 the same images as in Figure 4 with this new network that directly stylizes images instead of solving the entire optimization problem. Results are pretty good, but as we can see, in contrary to the previous model, it seems to apply the style in a "less macroscopic" manner. For example, the mosaic effect creates smaller mosaic panels than before.

Figure 8 shows 3 close (only some seconds apart) frames taken from a stylized video of Liège. We observe that colors and style shapes are similar on the consecutive frames, resulting in a more pleasant video. Unfortunately, we noticed that it is not always the case, especially for large areas with uniform color in the content image, for example the sky.

FIG. 8: 3 close frames from a stylized video

Our implementation seems to well maintain consistency between consecutive frames. The noise-based loss allows to preserve colors and shapes over time, even if there is a timelapse switching to another scene and then coming back to a previous scene. However, the noise-based loss reduces this effect along with popping effect but there is still improvement to make. A video representing our results with fast style network and temporal loss can be accessed here: `https://youtu.be/5pgZ3lq6TkU`.

Further information about the tuning of some parameters can be found in the appendix.

6. *Results conclusion*

The third method really seems to be the best choice by far, performing on average way better than the two first techniques, even though these two other techniques may sometimes produce acceptable or interesting results. We can say that the first method is rarely the best one to use, it takes time and really rarely produces acceptable results. The second one seems to give often more consistent results in some sense than the fast style network (especially almost no flickering at all), but it produces big artifacts and overall strange content that, most of the time, will make the video either nice and pleasant to watch, or completely unwatchable. It also seems to capture the style more sometimes, even though it makes the video content very hard to perceive, for example with Kandinsky's style. Although it still keeps having some flickering, and inconsistency, the third method seems to produce excellent and pleasant results almost all the time. They probably can be even better with more tuning, which is however difficult considering the time it takes to train one network.

B. Quantitative Analysis

1. *Still images*

Let us now focus on the loss convergence of style and content on an image optimization. Let us note that optimization starts from the content image as initial image.

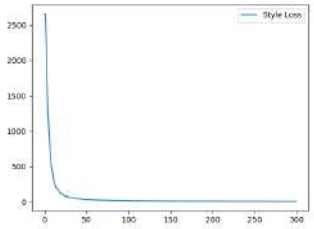

FIG. 9: Style Loss

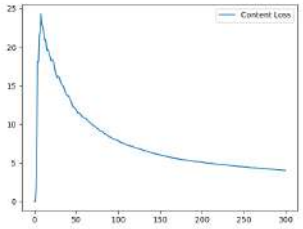

FIG. 10: Content Loss

As expected, both losses converge. We can note that the content loss evolves in a different manner, as it first starts low, then suddenly increases at the beginning of optimization, and finally starts decreasing in an inverse exponential fashion. This is due to the fact that we initialize with the content image. The initial content loss is thus obviously null. Then, it starts increasing as the optimizer deviates from the content, but later converges back to lower values.

We can also notice, that after only 50 iterations, the style has converged. However, it is not sufficient for the content loss which takes more iterations to come back to a low enough level.

2. *Temporal loss*

Let us now plot the evolution of our temporal loss over the optimization of 2 middle frames of a GIF.

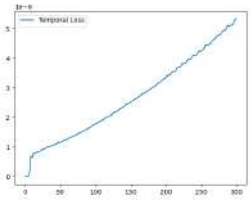

FIG. 11: Temporal Loss (300 iterations)

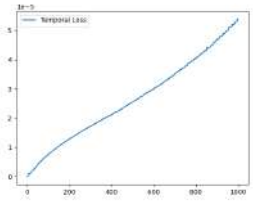

FIG. 12: Temporal Loss (750 iterations)

As can be seen pretty easily, the loss does not really converge at all. It increases all along the optimization process. We did not manage to make it converge better. This is probably why it does not have that much effect, and probably also confirms why temporal initialization has much more impact.

3. *Fast Style Network*

Let's do the same analysis but for the fast style network. Figure 13 and 14 show that the content and style losses again converge.

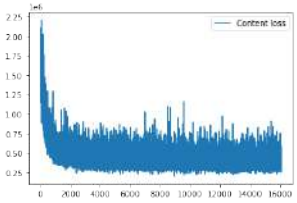

FIG. 13: Content Loss, Fast net.

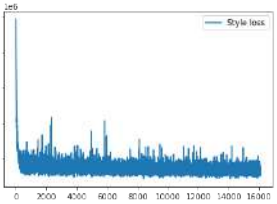

FIG. 14: Style Loss, Fast net.

We see that the style loss is lower than the content one as we are feeding the network with always new content images whereas the style image never changes. Figure 15 and 16 show that the temporal remains quite constant while the total variation loss increases linearly over training iterations.

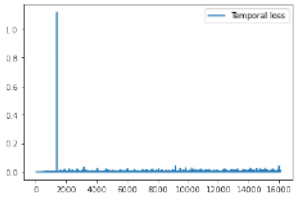

FIG. 15: Temporal Loss, Fast net.

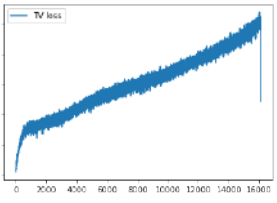

FIG. 16: TV Loss, Fast net.

However, we tested to delete this temporal loss and resulting images were completely different from the ones with the temporal loss. About the total variation loss, it seems that the weight given to this loss is way too low and thus does not have a big impact on training. It is important to mention that it actually extremely depends on the style image we used to train the model.

4. *Computation Time Analysis*

Here, we compare the computation times of our different methods, on both images and videos (time per frame).

Everything is run on Ubuntu 22.04 with PyTorch and CUDA. The hardware used is an Intel i7-10700k CPU, an NVIDIA RTX 3080 GPU, and 32GB of RAM.

Computation time is highly dependent on the image size chosen, the number of iterations, and also, for videos, on whether or not temporal loss (using optical flow) is used. Content and style weight don't seem to influence that much the computation time.

Using the previous hyper-parameters: LBFGS, and 300 iterations, a 512x342 image (downsampled from 1920x1281) is optimized in about 10 to 15 seconds. Likewise, a 512x374 GIF of 62 frames is stylized in 9 minutes, which means about 9 seconds per frame, using no temporal loss. When using optical flow for computing the temporal loss/initialization, the computation increases by a lot, due to the constant calls to OpenCV and the heavy computation of this flow. Indeed, the same GIF is now stylized in 20 to 30 minutes.

For the Fast Style Transfer network, considering the values mentioned in the training section (batch size of 2, ...), it takes about 38 minutes to fully train a network with 256x256 images for one given style, including the use of the temporal noise-based loss. We increased to 512x512 for better results, which resulted in 2-hour-15-minute-long trainings. Afterwards, we were able to stylize a 4'30"-long 1280x720 video at 30fps (8093 frames) in only 12 minutes, which means approximately 11 images stylized per second. For a 500x364 GIF of 62 frames, it was stylized in only 2 seconds (27 fps). We can really see how faster it is. It takes some time to train, even though training is actually not that long compared to applying the other algorithm on a 512x374 GIF frame by frame, or even less long compared to the optimization with optical flow. Once it is trained though, it runs very fast and can stylize small videos in quasi-real-time. The 720p 4'30"-long video would take about 45 hours to process using the previous algorithm with optical flow temporal loss! This is really unusable.

V. DISCUSSION

A. Performance

For overall visual quality of still images, the standard algorithm as proposed by Gatys et al. produces, as seen, very good results with no modifications. It requires some tuning depending on the style image, but it is possible to achieve very nice results. For videos though, it becomes really quickly impractical to use this method, as computation times become really long even for a short GIF, and it yields pretty disappointing results most of the time. The temporal initialization provides interesting results, but these stay on average pretty deceiving.

However, in that regard, the final method which combines the trained fast style network with noise-based temporal loss, yields very good results and runs in quasi real-time on short videos. It obviously, however, requires long training for good results, and is thus hard to tune (especially for the content and style trade-off).

One big advantage of the original method, especially for images, is thus the fact it does not need to train a network for each new style. Therefore, if one wanted to apply a lot of different styles to one or several images, it would become impossible with the fast style network, which is itself better for applying a small number of styles to a large number of images, like the frames of a video.

B. Limitations

As mentioned then, the main limitations are about *time*. All methods can quickly be problematic about time in some regard. The first method also suffers from big artifacts, bad consistency, and overall bad visual quality on videos. The second method also has huge artifacts, but more consistency. The Fast Style Network has less limitations in that regard, but still has occasional visual artifacts, along with some flickering. In addition to these limitations, tuning the parameters really depends on the style and content images, which again is time consuming.

C. Future Work

There are many possible improvements to explore, including many features mentioned in the papers we read and based our ideas on. Concerning optical flow, it would be best to use another method to estimate them, like DeepFlow [18], and take more frames into account in the temporal loss. Based on [4], we could also implement a multi-pass algorithm, which processes the video in several directions to produce more consistent results.

One could also consider a different loss function than MSE to compare the style and content representations, that would match better closer results as MSE could penalize feature maps that are only different because of some offset for example, while it should not.

For the temporal loss, we should also only consider areas where the optical flow is consistent, as done by Ruder et al. This could probably help a lot for disocclusions. Another way to perhaps fix that strange behaviour when using the temporal (optical flow) loss along with temporal initialization would perhaps be to increase the learning rate, and/or increase the number of iterations. Although we tried doubling the amount of iterations, the loss convergence didn't change at all, and stayed at the same level, and it didn't remove the dark areas and the "sticky" behaviour. Maybe even more iterations are needed, or a different learning rate.

[1] `www.tensorflow.org/tutorials/generative/style\protect_transfer`.

[2] `https://en.wikipedia.org/wiki/Loving_Vincent`.

[3] Gatys et al. A neural algorithm of artistic style. 2015.

[4] Manuel Ruder, Alexey Dosovitskiy, and Thomas Brox. Artistic style transfer for videos. In *Lecture Notes in Computer Science*, pages 26–36. Springer International Publishing, 2016.

[5] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution, 2016.

[6] Jeffrey Rainy and Archy de Berker. Stabilizing neural style-transfer for video. 2018.

[7] Samet Hicsonmez, Nermin Samet, Emre Akbas, and Pinar Duygulu. Ganilla: Generative adversarial networks for image to illustration translation, 2020.

[8] Wenju Xu, Chengjiang Long, Ruisheng Wang, and Guanghui Wang. Drb-gan: A dynamic resblock generative adversarial network for artistic style transfer, 2021.

[9] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2017.

[10] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[11] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[12] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[13] `https://en.wikipedia.org/wiki/Gram\protect_`

matrix.
[14] Dong C. Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. 1989.
[15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.
[16] Slav Ivanov. Picking an optimizer for Style Transfer. 2017.
[17] Haozhi Huang, Hao Wang, Wenhan Luo, Lin Ma, Wenhao Jiang, Xiaolong Zhu, Zhifeng Li, and Wei Liu. Real-time neural style transfer for videos. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7044–7052, 2017.
[18] Lukas Mosser, Olivier Dubrule, and Martin J. Blunt. Deepflow: History matching in the space of deep generative models, 2019.
[19] Itseez. Open source computer vision library. `https://github.com/itseez/opencv`, 2015.
[20] `https://en.wikipedia.org/wiki/Total_variation_denoising`.
[21] Victor Lempitsky Dmitry Ulyanov, Andrea Vedaldi. Instance normalization: The missing ingredient for fast stylization, 2016.
[22] Sam Gross and Michael Wilber. Training and investigating residual nets, 2016.
[23] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution: Supplementary material.
[24] R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L. Li, D. Shamma, M. Bernstein, and L. Fei-Fei. Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations, 2016.
[25] T.Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C.L. Zitnick. Microsoft COCO: Common Objects in Context, 2014.

Appendix A: Still images with Fast Style Network

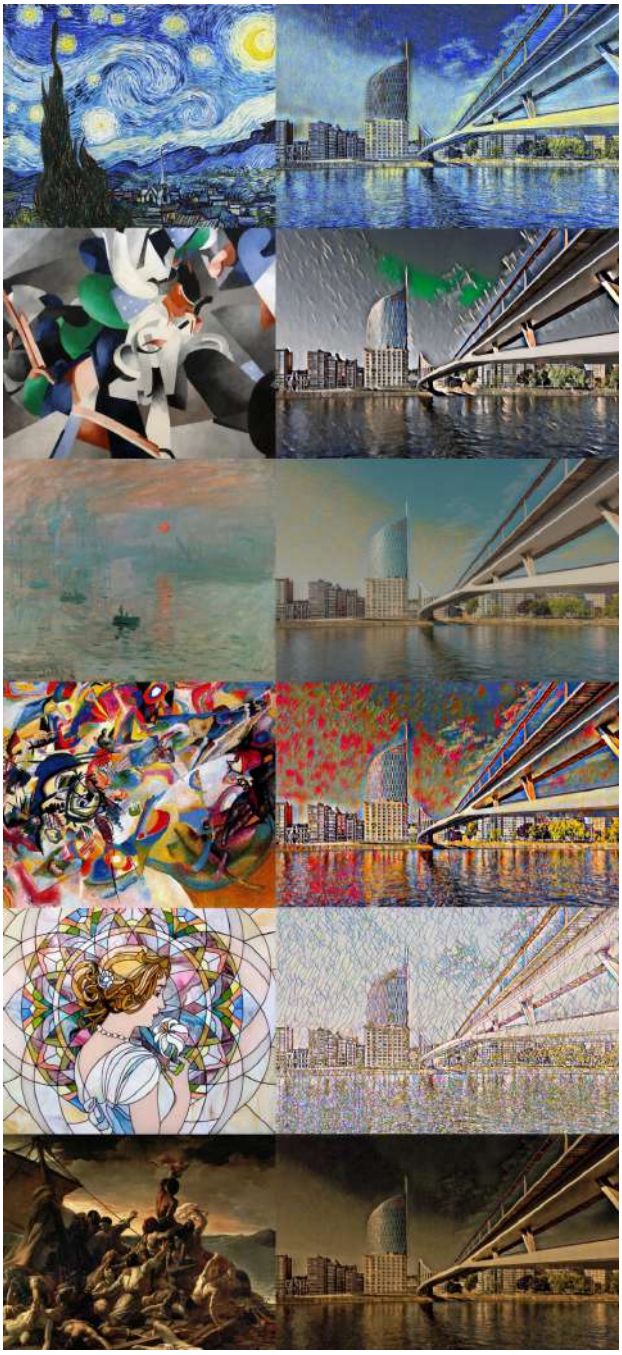

FIG. 17: Style images and the result images, on content image from Figure 3, as in Figure 4, but this time generated by the Fast Style Network.

Appendix B: Anime Style with Fast Style Network

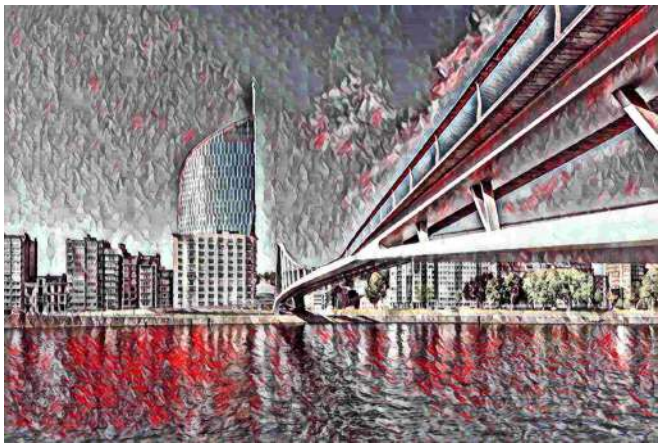

FIG. 18: FSN performs better than the first method with the japanese anime style on still images.

Appendix C: Fast Style Network with no noise-based temporal loss

Let us quickly see how the fast style network without the temporal (noise-based) loss performs compared to the network with this loss added. The following video shows this : `https://youtu.be/iRdxbWd8KUE`.

As can be seen, the video is more stable (less popping) with the temporal loss. We can however notice how stylizing directly through a deep neural network trained without the loss still performs pretty good, and is a lot more stable than doing optimization independently on each frame.

Appendix D: Learning rate with Fast Style Network

Figure 19 and 20 show content, style, temporal and total variation losses for different values of Adam learning rate. One can see that $\gamma = 0.001, 0.01$ lead to quite same results whereas $\gamma = 0.1$ is slightly worse.

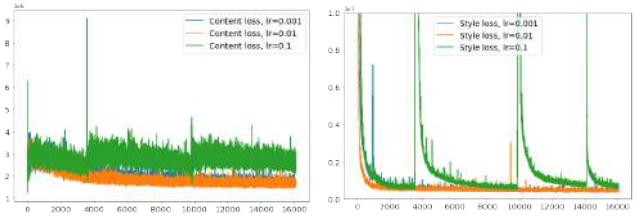

FIG. 19: Content and style variation losses for different values of Adam learning rate

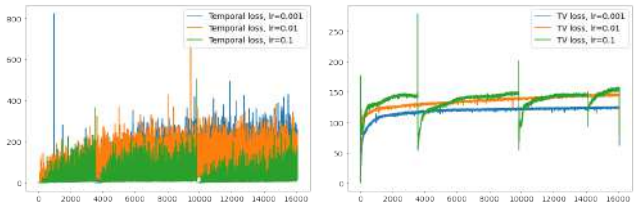

FIG. 20: Temporal and total variation losses for different values of Adam learning rate

Figure 21 shows 3 stylized frames of the Figure 3 with different Adam learning rates. We clearly see that $\gamma = 0.001, 0.01$ lead to nice results but $\gamma = 0.1$ gives a limited stylized image as it is no longer possible to clearly distinguish shapes, especially the trees located on the right of the image.

FIG. 21: Fast Style Network stylization for different values of Adam learning rate: $\gamma = 0.001$ for left image, $\gamma = 0.01$ for center image and $\gamma = 0.1$ for right image

Appendix E: Noised-based loss (Fast Style Net.): effect of the number of modified pixels

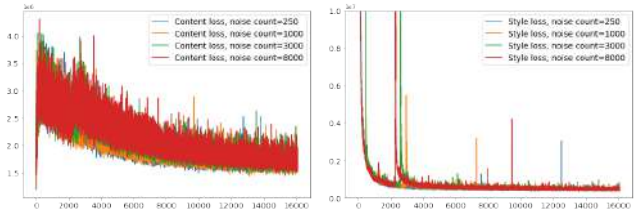

FIG. 22: Content and style variation losses for different values of the number of modified pixels

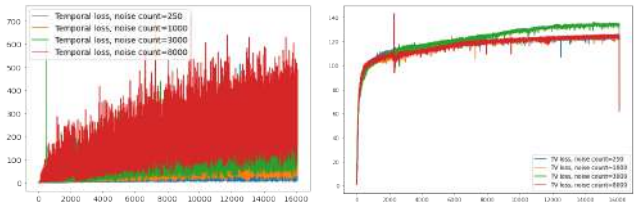

FIG. 23: Temporal and total variation losses for different values of the number of modified pixels