INFO8010 : Neural style transfer

Maxime Meurisse,[1] **Adrien Schoffeniels**,[2] and **Valentin Vermeylen**[3]

[1] *m.meurisse@student.uliege.be (s161278)*
[2] *adrien.schoffeniels@student.uliege.be (s162843)*
[3] *valentin.vermeylen@student.uliege.be (s162864)*

I. INTRODUCTION

We decided to tackle the subject of *neural style transfer* for this deep learning project. The basic idea of this subject is to *merge the style of one image and the content of another image*, for example applying the style of an oil painting to a photo of an animal (Figure 1).

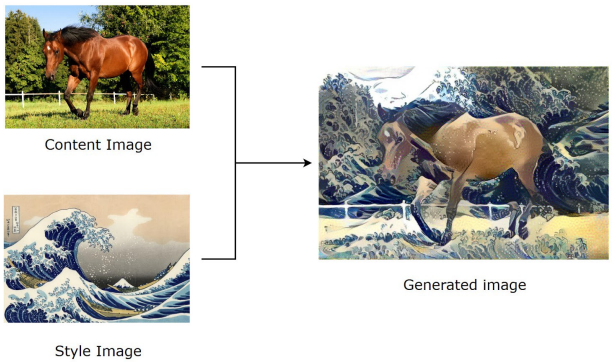


FIG. 1: An example of neural style transfer (Gatys et al. technique) [1].

However, the rich domain of neural style transfer does not stop there, many other possible applications and extensions exist : modifying the style of a painting to make it look like a photo (and vice versa), modifying the style of certain elements of a photo (for example, transforming horses into zebras by applying a striped texture to them), transforming a summer landscape into a winter landscape, coloring an image, etc.

The applications are many and varied. We have chosen to focus on the application of merging the style of an image and the content of another image.

To do this, we first started by exploring the technique explained in the paper by Gatys et al. [2]. We tried to re-implement this technique ourselves, to improve it, and to vary the different parameters to understand how it works. This method is described in section III A and its results are discussed in section IV A.

Secondly, we have looked at the CycleGAN (*Cycle-Consistent Generative Adversarial Networks*) technique introduced in the paper by Zhu et al. [3]. This technique, using generative adversarial networks, is very rich and allows many possible applications.

An example of style transfer using CycleGAN is shown in Figure 2.

FIG. 2: An example of neural style transfer (Zhu et al. technique) [4].

We have also tried to re-implement it by ourselves and to vary its different elements. This method is described in section III B and its results are discussed in section IV B.

Finally, we tried some extensions in order to dig deeper into the subject : use of different models, techniques to try to control colour, etc. These results are discussed in the corresponding sections of each method. The different difficulties encountered as well as our impressions of this work and our conclusion are explained in section V.

II. RELATED WORK

The field of neural style transfer and its applications are very rich. We have researched the literature to find out what the basic principles are, what the different techniques are and how they can be improved.

We first based ourselves, as explained in the introduction, on the paper by Gatys et al. and on the paper by Zhu et al. :

- [2] - Paper by Gatys et al. : a paper that introduces a technique for neural style transfer using a network, such as VGG19, already trained.
- [3] - Paper by Zhu et al. : a paper that introduces a technique, called CycleGAN, to perform unpaired image-to-image translation.

We then looked for some papers that could help us better understand the subject and improve our techniques. The field of style transfer in deep learning is very rich and its literature very varied. We consulted a wide range of sources but we focused on papers that were most relevant to our implemented techniques.

- [5] - Paper by Jing et al. : a paper that provides a comprehensive overview of the current progress towards neural style transfer.
- [6] - Paper by Gatys et al. : a paper that follows their original one ([2]) and introduces several improvements concerning the results that can be obtained. Examples are the possibility to include bitmaps, to control the colour transfer and the scale.
- [7] - Paper by Johnson et al. : a paper that takes advantage of *perceptual loss functions* to design a style transfer algorithm. The latter is compared with the original technique of Gatys et al.
- [8] - Paper by Ledig et al. : a paper that presents SRGAN, a generative adversarial network (GAN) for image super-resolution (SR).

III. METHODS

A. Neural style transfer (Gatys et al.)

The method presented in this section is that of the original paper that introduced neural style transfer. Although style transfer previously existed in a field called *non-photorealistic rendering* and were implemented through *artistic rendering* algorithms [5], they made no use of neural networks for the task and achieved limited performance and results.

The paper [2] makes use of convolutional neural networks to tackle the task. Those networks are the most powerful when dealing with images since they are able to quickly and efficiently detect local and global correlations in an image. Thanks to filters implemented at each layer, features of the image can be gradually extracted in a feedforward manner. The outputs of each layer consist therefore in *feature maps*, which are filtered versions of the input image.

The authors decided to use networks that were pre-trained on object recognition tasks. Such a network creates a representation of the input image that makes the object information increasingly explicit during the treatment, and the content of the image is thus given more weight than the fine details. Higher-level layers contain high-level content while lower-level ones contain low-level details. High-level layers are thus used for content representation. On the other hand, the style is captured through correlations between the filter responses of different layers.

The realization that made neural style transfer possible for the authors is that the style and content of an image are *separable in a convolutional neural network.*

As for the practical method proposed by the paper, they used the pre-trained VGG19 network, consisting of 16 convolutional layers and 5 pooling ones. The responses in a layer l being the features maps of that layer, they can be stored in a matrix $F^l \in \mathcal{R}^{N_l \times M_l}$, where F^l_{ij} is the activation of the i^{th} filter at position j in layer l, N_l is the number of distinct filters at layer l and M_l is the size of the feature map (height times width).

The loss that the authors have used in their work is the following one :

$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x}) \quad (1)$$

where

$$\mathcal{L}_{\text{content}}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} \left(F^l_{ij} - P^l_{ij}\right)^2 \quad (2)$$

and

$$\begin{aligned} \mathcal{L}_{\text{style}}(\vec{a}, \vec{x}) &= \sum_{l=0}^{L} w_l E_l \\ &= \sum_{l=0}^{L} w_l \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G^l_{ij} - A^l_{ij}\right)^2 \end{aligned} \quad (3)$$

The parameter p is the photograph (content image), x is the constructed one and a is the artwork (style image). Parameters α and β are simply weighting factors controlling the tradeoff between style and content. The content loss is the mean square error between the response matrix of the content image and that of the constructed image, while the style loss is defined as the weighted sum of the mean square errors between the Gram matrices for the generated image and the style one, taken over all layers considered for the style. The *Gram matrix* is a mathematical element that computes the correlation between the different filter responses for a given layer.

1. Network architecture

The authors used model VGG19 as a basis : VGG19 is a convolutional neural network which consists of 16 convolutional layers (to transform images to feature maps), 5 pooling layers (to reduce the dimensions of the data) and 3 fully connected ones (at the end where every neuron in one layer are connected to every neuron in another layer) where 19 layers are trainable (convolutional and fully connected layers). This model

is used to classify images.

The authors have modified this network by inserting layers to reconstruct the image. For the content, a layer was inserted after a convolutional layer. For the style, several layers have been added after convolutional layers (as explained previously, the style is rebuilt by calculating the correlation between the different features of different layers).

The insertion of additional layers can be done in several different places. We have chosen this architecture as a starting point : `normalization`, `conv_1`, `style_loss_1`, `relu_1`, `conv_2`, `style_loss_2`, `relu_2`, `pool_2`, `conv_3`, `style_loss_3`, `relu_3`, `conv_4`, `content_loss`, `style_loss_4`, `relu_4`, `pool_4`, `conv_5`, `style_loss_5`. One can notice that the architecture is not complete. Indeed, all layers after the last style layer inserted have been put aside : they are no longer useful once the content and style have been reconstructed. This first `normalization` layer is used to normalize input images in order to fit VGG19.

2. Implementation

We tried to implement this technique by ourselves with PyTorch. In order to correct our errors and improve our implementation, we used the [9] and [10] tutorials. Our basic implementation is as faithful as possible to the paper [2] : we have taken VGG19 and inserted the new layers at the same places, we have taken the same values of the α and β parameters, etc. We varied all these parameters afterwards. We didn't need to build up a large dataset of images (since we do not need to train VGG19); we simply downloaded images of style and content from the Internet.

B. CycleGAN (Zhu et al.)

CycleGAN is neural networks that allows to perform *unpaired image-to-image translation*. As described on its official web page, "*image-to-image translation is a class of vision and graphics problems where the goal is to learn the mapping between an input image and an output image using a training set of aligned image pairs.*" [4]. However, in many cases, paired images are not available. This technique has therefore been designed to support unpaired images.

Paired images are images with a common context and common elements (for example, an image of horse in a field and an image of a zebra in the same position in more or less an identical field) while unpaired images are images that are not really the same (for example, an image of a horse in a field and an image of a zebra, not in the same position, not in the same size and not in a field).

CycleGAN uses *generative adversarial networks*.

1. Generative models

Generative models are models capable of learning how to create data similar to the data they are provided with (training data). Intuitively, a model acting on data (e.g. writing good scientific articles) must have a good internal representation of this data. This internal representation could then be exploited to perform other tasks (e.g. classifying scientific articles).

More practically, a generative model is a model that attempts to learn the joint probability distribution $P(x, y)$ between data x and their label y (where a non-generative model learns the conditional probability $P(y|x)$). Thus, the model would be able to generate a new sample (x, y).

2. Generative adversarial networks

Generative adversarial networks are models whose main idea is to put in competition two neural network models. One is called the *generator* and is in charge of generating noisy data. The other is called the *discriminator* and receives the training data and the noisy data from the generator and has to be able to distinguish between the two sources.

An illustration of the structure of a generative adversarial networks is presented in Figure 3.

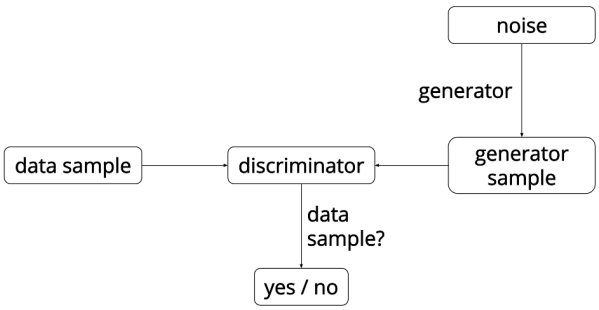


FIG. 3: Generative adversarial networks structure [11].

As the training progresses, the generator produces less and less noisy data and the discriminator becomes better at distinguishing the source from the data it receives. The ultimate goal is to make the generator generate data that will be indistinguishable from the training data by the discriminator.

In such a model, we can backpropagate gradient information from the discriminator back to the generator

network, so the generator knows how to adapt its parameters in order to produce output data that can fool the discriminator.

3. CycleGAN

Coming back to the application of neural style transfer, the idea behind the use of generative adversarial networks is *to learn a mapping between the input image and the output image*. However, this technique works for paired images. CycleGAN is a technique that works on unpaired images (or more generally, unpaired input data). For example, it will be able to transform a horse in a photo into a zebra (by applying a striped texture to it) without even needing a zebra image in the same position, with the same background, etc. (i.e. paired images of horse and zebra).

The operating principle of CycleGAN is as follows : let one set of images (for example, images of horses) in an X domain and another different set of images (for example, images of zebras) in a Y domain. The goal is to obtain, via training, a mapping $G : X \to Y$ and a mapping $F : Y \to X$. Mathematically, G and F should be inverse to each other and both mapping should be bijections. This hypothesis being made, G and F are trained simultaneously via a *cycle consistency loss* that encourages $F(G(x)) = x$ and $G(F(y)) = y$.

The creators of CycleGAN have in fact exploited the fact that a translation must be "*cycle consistent*" using this technique. Concretely, this means, for example, that if a sentence is translated from English into French, the re-translation of the freshly obtained French sentence into English should give the original sentence.

Exploiting this property avoids problems such as *mode collapse* : the model could generate an image which is in the domain of Y, and which would therefore be validated by the discriminator, without having any link with the domain X (for example, generating an image of a zebra that is always the same and does not look anything like the image of the original horse).

In total, the model contains two loss functions : an *adversarial loss function* (since it is a generative adversarial model) and a *cycle consistent loss function* (to take into account the cycle consistent property).

The adversarial loss function is defined as

$$\begin{aligned}\mathcal{L}_{\text{GAN}}\left(G, D_Y, X, Y\right) = &\mathbb{E}_{y \sim p_{\text{data}}\,(y)}\left[\log D_Y(y)\right] \\ &+ \mathbb{E}_{x \sim p_{\text{data}}\,(x)}\left[\log\left(1 - D_Y(G(x))\right)\right]\end{aligned} \tag{4}$$

where D_Y is a discriminant which aims to distinguish the images y from the translated images $G(x)$. They also consider the same loss for D_X.

The cycle consistent loss function is defined as

$$\begin{aligned}\mathcal{L}_{\text{cyc}}\left(G, F\right) = &\mathbb{E}_{x \sim p_{\text{data}}\,(x)}\left[\|F(G(x)) - x\|_1\right] \\ &+ \mathbb{E}_{y \sim p_{\text{data}}\,(y)}\left[\|G(F(y)) - y\|_1\right]\end{aligned} \tag{5}$$

taking into account that the result must satisfy the *forward cycle consistency* ($x \to G(x) \to F(G(x)) \approx x$) and the *backward cycle consistency* ($y \to F(y) \to G(F(y)) \approx y$).

An illustration of the cycle consistent loss is shown in Figure 4.

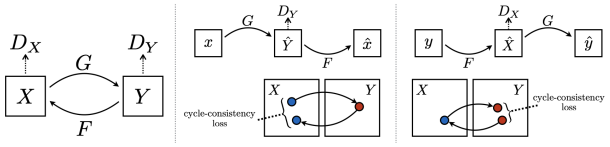


FIG. 4: Cycle consistent loss (CycleGAN) [3].

Finally, the total loss function that the model tries to minimize is defined as

$$\begin{aligned}\mathcal{L}\left(G, F, D_X, D_Y\right) = &\mathcal{L}_{\text{GAN}}\left(G, D_Y, X, Y\right) \\ &+ \mathcal{L}_{\text{GAN}}\left(F, D_X, Y, X\right) \\ &+ \lambda \mathcal{L}_{\text{cyc}}(G, F)\end{aligned} \tag{6}$$

where λ controls the relative importance of the two objectives.

4. Network architectures

Concerning generator architectures, authors of CycleGAN use architecture from Johnson et al. [7] : 6 residual blocks for 128×128 training images and 9 residual blocks for 256×256 or higher-resolution training images.

Concerning the discriminator architectures, they use a 70×70 PatchGAN.

5. Implementation

In order to implement the CycleGAN, we decided to not base ourselves upon a tutorial but instead to try to recreate the models and its training from the paper [3] itself. The network architecture is referenced in section III B 4 of this paper and makes reference to the network used in another paper [7]. The exact architecture of the networks of this paper being given in the supplementary and using the discriminator architecture of [8]. We therefore tried to implement our network based on these sources and finished doing so before realizing that

the exact architectures of CycleGan were given in the appendix of their paper, located after the sources. We thus used that source to modify the networks created to have the same basis as them. For the generator, we did not modify anything since the architecture we used and modified based on their comment in their own implementation was the same as the one provided in their appendix. For the discriminators, however, we had to modify our architecture since they did not mention that the kernel sizes were of 4, and so we went with the kernel size of 3 referenced by the original paper they based upon, and they used a stride of 2 for all layers, which was not the case in [8]. The number of blocks they used also differed.

Finally, after having implemented the network, we went to their GitHub repository for the paper and compared our architecture and theirs, to make sure everything was correct. The only modification we had to make was change some reflection padding to simple padding with 0. They mentioned in their paper they used reflection padding to reduce artifacts but it was not done for all convolutional layers, so we modified that.

After the network architectures, we had to acquire a dataset of images and paintings. To do so, we used a bash script found on their GitHub repository that downloads the same datasets as they used in their research. We decided to use landscapes and Monet paintings for the two domains of interest. We also had to create a custom data loader that would return an image of both domain when called, and to do so we looked into PyTorch documentation and StackOverflow questions.

For the optimizer, we reused the same as the authors, and created a learning rate scheduler acting as they wrote in their paper.

Finally, we could start the training. The default number of epochs was set to 200, as recommended by the authors. However, we had to change the training dataset, as will be explained in the discussion section. The training first creates fake images by feeding true images to the generators, and then trains the generators by using a loss combining the cycle loss (going to then from a domain to the other should yield an image close to the input one) and the adversarial loss (can we trick the discriminators into thinking a generated image is real), and then trains the discriminators by comparing the output of the discriminators for the true and false images to ground truths.

IV. RESULTS

A. Neural style transfer (Gatys et al.)

In order to better understand this technique to obtain the best possible results, we tried to vary the different parameters (and hyperparameters) controlling the model. These are listed below :

- input image;
- model used;
- architecture of the model used;
- number of steps (epochs);
- weights;
- added layers.

We discuss results obtained in a qualitative way by comparing the image obtained with the initial images and in a quantitative way by discussing the loss functions. Since this is a very visual, and therefore very subjective, application, there are few quantitative criteria to discuss. The quality of the result obtained depends very much on the user and his visual feeling.

1. Input image

The input image is the image that will be generated by the model. It can be initialized in two different ways : either by taking a copy of the content image, or by taking a white noise version of the content image. According to the literature, these two possibilities are equivalent. We have therefore tried both versions.

We chose the *Arc de Triomphe* (Paris) as the reference content image (5a) and some paintings as reference style images (5b and 5c). These images will be used in further discussions. Obviously, the parameters that give good results for these images will not necessarily give good results for all images. We will discuss this point later.

(a) Content image (b) Style image 1 (c) Style image 2

FIG. 5: Reference images used for discussions.

For this first test, we kept the original implementation, faithful to paper [2], with 300 epochs. The results obtained are presented in Figure 6.

(a) Start with copy of content image

(b) Start with noisy content image

FIG. 6: Input image variations.

As can be seen, the results obtained are very similar. However, starting with a copy of the content image seems to give smoother results for the majority of cases. For this reason, we will choose to start with a copy of the content image afterwards.

2. *Model used*

The authors used a pre-trained version of the VGG19 model. This version already has a set of defined weights for a better classification and therefore better results for style transfer. We tried to run the algorithm with the non pre-trained VGG19 model to compare the two versions. The results obtained are shown in Figure 7.

(a) Not pre-trained model

(b) Pre-trained model

FIG. 7: VGG19 model variations.

We can easily observe that the pre-trained model gives much better results than the non pre-trained model. This result seems logical since the pre-trained model allows a less good extraction of the features of each image and therefore a much worse reconstruction and mix of content and style of each image.

We tried to increase the number of epochs of the algorithm with the non pre-trained model, but the results obtained are systematically much lower than the results with the pre-trained model. To obtain better results, we should have trained the VGG19 model beforehand. Since image classification was not the main subject of our project, we worked directly with the pre-trained model afterwards.

3. *Architecture of the model used*

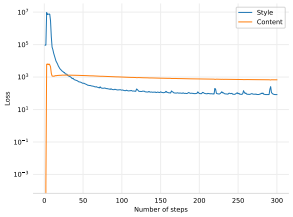
The original architecture of VGG19 has max pooling layers. The authors suggest in their paper to replace these max pooling layers by average pooling layers in order to obtain more appealing results. We have therefore compared the results with these two types of layers (Figure 8).

(a) Max pooling layers

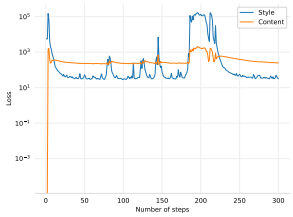
(b) Average pooling layers

FIG. 8: VGG19 model architecture variations.

At first glance, both results seem to be very good. However, in general, some images give very bad results with average pooling layers (while results with max pooling layers are always very good). We have therefore observed the evolution of the loss functions (Figure 9).



(a) Max pooling layers



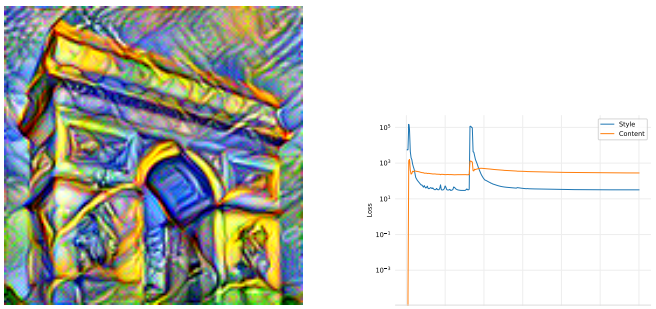
(b) Average pooling layers

FIG. 9: VGG19 model architecture loss functions.

We observe that in the case of an architecture with average pooling layers, the loss functions (and especially the one concerning the content) vary strongly during the epochs while in the case of max pooling layers, the loss functions remain stable. This could explain the very bad results obtained on some images with average pooling layers.

To overcome this, we have added a *scheduler* (with a step size of 50 and a gamma parameter of 0.3) when working with average pooling layers. This scheduler allows to reduce the learning rate during epochs, and thus

to reduce the variations of the loss functions. We then restarted the algorithm with this scheduler (Figure 10).



(a) Transformed input image

(b) Loss functions

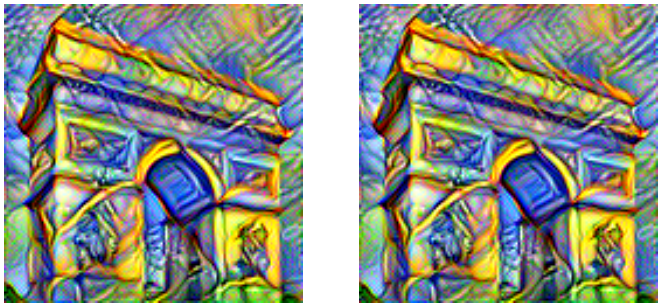
FIG. 10: Average pooling layers with scheduler (VGG19).

We observe that the scheduler seems to be efficient since the variations of the loss functions have almost completely disappeared. In general, the results that were previously bad have turned out to be very good with the scheduler. The images obtained with average pooling layer seem to be smoother. So we kept this type of layer (with a scheduler).

4. *Number of steps (epochs)*

By default, we have set the number of steps (epochs) to 300. The results are obtained quickly and seem to be good. When we observe the loss functions (Figure 10 for example), we realize that they stabilize quite quickly (after about 300 epochs).

In order to confirm this result, we run the algorithm by increasing the number of epochs to 1000. The result obtained is shown in Figure 11.



(a) 300 epochs

(b) 1000 epochs

FIG. 11: Number of steps (epochs) variations.

As expected, the two results are exactly the same. Indeed, considering the evolution of the loss functions (Figure 12) per 1000 epochs, we can confirm that they stabilize quite quickly. We have therefore set the number of epochs to 300 by default.

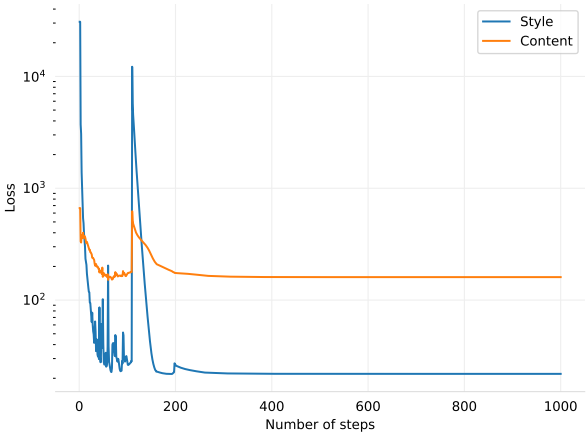
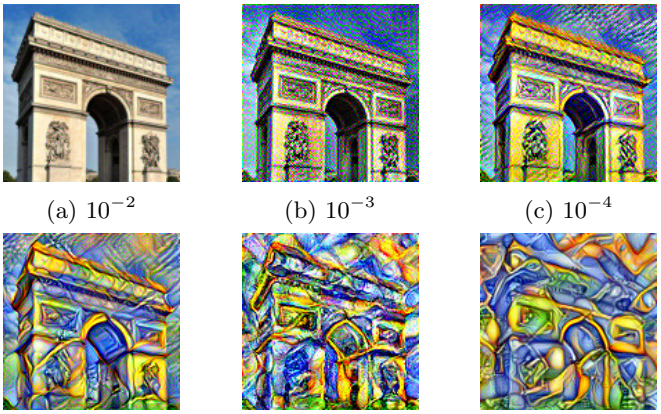


FIG. 12: Loss functions for 1000 epochs.

5. *Weights*

It is possible to vary the α and β weights in the loss function (1). These weights are used to distribute the style and content of the resulting image. In their paper, the authors use an α/β ratio equals to 10^{-4}. We have tried several ratios to observe the variations of the results obtained (Figure 13).



(a) 10^{-2} (b) 10^{-3} (c) 10^{-4}

(d) 10^{-5} (e) 10^{-6} (f) 10^{-7}

FIG. 13: Results for different values of ratio α/β.

We observe that if the ratio is too high (10^{-2}), the result is not influenced at all by the style image and, on the contrary, if the ratio is too low (10^{-7}), the style totally takes over the result. A suitable ratio seems to be around 10^{-5}. This result is of course highly dependent on the images and the expected result : a low contrast style image with dull colors will require a very low ratio to appear while a very bright and colorful style image will appear with a higher ratio. In general, the 10^{-5} ratio seems to give good results for all image combinations. We have therefore kept this value.

We note that the α value is much smaller than the β value. Indeed, the base image being a copy of the content image, it is not necessary to greatly modify it from a content point of view.

We also tried to assign a weight to each layer added to rebuild the style image. Indeed, the authors mention in their paper that "*[...] when matching the style representations up to higher layers in the network, local images structures are matched on an increasingly large scale, leading to a smoother and more continuous visual experience. Thus, the visually most appealing images are usually created by matching the style representation up to the highest layers in the network. [...]*" [2]. We therefore varied the influence of the different layers used to rebuild the style image by assigning them different weights (Figure 14).

(a) $w = [1, 1, 1, 1, 1]$ (b) $w = [1, .8, .6, .4, .2]$ (c) $w = [.75, .5, .2, .2, .2]$

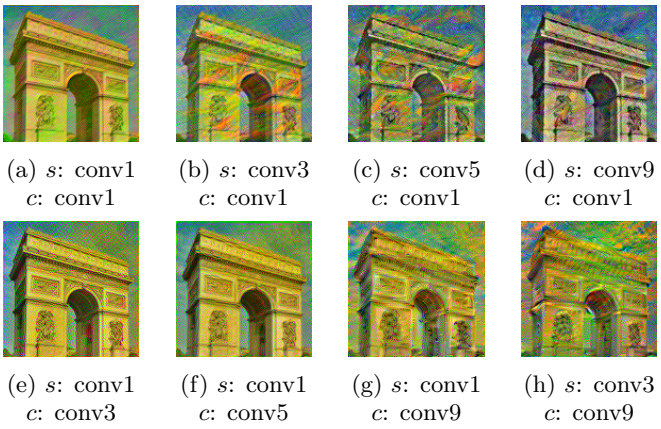
FIG. 14: Style layers' weights variations.

We get different results, but each one is very good. We find that, logically, by assigning less weight to the different style layers (especially the last ones), we obtain results where the style is less marked. Again, the ratings are very subjective and depend on the user's desired results.

6. *Added layers*

The authors mention in their paper that the layers considered to reconstruct the image of content and the image of style have a significant influence on the result. The content image is rebuilt after a convolutional layer and the style image is rebuilt based on a correlation of several layers. We tried to put them at different places inside the VGG19 model in order to observe the influence on the result obtained and try to find the best configuration. The results are shown in Figure 15.

As shown by the figures 15a, 15b, 15c and 15d, the style seems to be better reproduced when the style layer is placed in the beginning of the network. When it's placed after the first convolutional layer, the general color scheme is well respected but the colour shifts are smooth. We do not perceive brush strokes as they are in the original image. As we move the layer along the network, the colour scheme is less and less well-reproduced, but the structure of the style image (the brush strokes) is more and more visible. This



(a) s: conv1 c: conv1 (b) s: conv3 c: conv1 (c) s: conv5 c: conv1 (d) s: conv9 c: conv1

(e) s: conv1 c: conv3 (f) s: conv1 c: conv5 (g) s: conv1 c: conv9 (h) s: conv3 c: conv9

FIG. 15: Influence of the position of the construction layers (s is the layer after which the style layer is positioned, c is the layer after which the content layer is placed).

is the result expected since the higher convolutional layers capture higher level content. In accordance with these subjective observations, we found that the style loss increases when the layer is moved along the network.

In the same way, when we move the content layer along the network (as shown by the figures 15a, 15e, 15f and 15g), the representation of the monument becomes less and less precise, since low-level convolutional layers care about details while high-level ones care more about objects and their arrangement, without too much details. This way, placing the content layer further in the network allows to better apply the style to the image (it does not seem to be a picture with a filter on top anymore). Once again the style loss confirms these observations, as it lessens when the content layer advances in the network.

Based on these findings, we manage to get nice results by placing the style layers at different locations in the network (to get the right colours, but also a little bit of the form of the style picture), but giving more weight to the ones that come first, and by placing the content layer around the middle of the network (the exact location can be adapted according to the input images). An example of what can be achieved is shown in Figure 16b.

7. *Final results*

We have tried to explore the technique of Gatys et al. and to vary its different parameters in order to better understand it. In summary, we observed that taking as input image a copy of the content image gives smoother results; a pre-trained model logically gives better results; average pooling layers (with a scheduler) give, on average, better results; 300 epochs seem to be sufficient and that, finally, a ratio of 10^{-5} between the weight of the content and the weight of the style

(a) Starting architecture (b) Modified architecture

FIG. 16: Comparison of the result that can be achieved by modifying the network architecture with the starting point architecture.

seems to be good on average. Other parameters such as where new layers are added and the weights assigned to these layers depend very strongly on the input images.

We applied the algorithm to a whole series of images, adjusting the parameters as best we could according to the different discussions made previously. Our results are presented in Figure 17.

B. CycleGAN (Zhu et al.)

to do

V. DISCUSSION

A. Neural style transfer (Gatys et al.)

Our implementation of the technique of Gatys et al. is quite faithful to the original paper. Although inspired by tutorials, we have added our own personal touch in order to re-implement certain aspects and to better understand it.

The results we get seem good at first glance. However, the field of style transfer is very rich and has evolved a lot since the publication of the paper by Gatys et al. [2] and many methods that drastically improve the results have emerged. We have focused here on the basic principles but a whole range of improvements could have been possible. Our model has its limitations in that, given the rapidly stabilizing loss functions, we cannot get much better results just by playing with the basic parameters.

We used a pre-trained model for this project. We didn't bother trying to train the model by ourselves as it would have been very time consuming and not really related to our topic. However, training a model (e.g. VGG19) with a wide range of artistic images (paintings, etc.) might have led, maybe, to better results.

Finally, we realized how much deep learning techniques, even modern ones, require a lot of work on parameters and training in order to obtain suitable results. Moreover, a series of parameters that give suitable results for one image will not give suitable results for another; it is difficult to generalize the model.

We are satisfied with the results we are achieving. It is difficult to judge their quality from a quantitative point of view in view of this very subjective application, but nevertheless, from a qualitative point of view, the results are very correct.

B. CycleGAN (Zhu et al.)

The results obtained from CycleGAN were very disappointing. Despite many fixes, the losses for the discriminators were systematically reduced during the training while the ones for the generators stayed near constant.

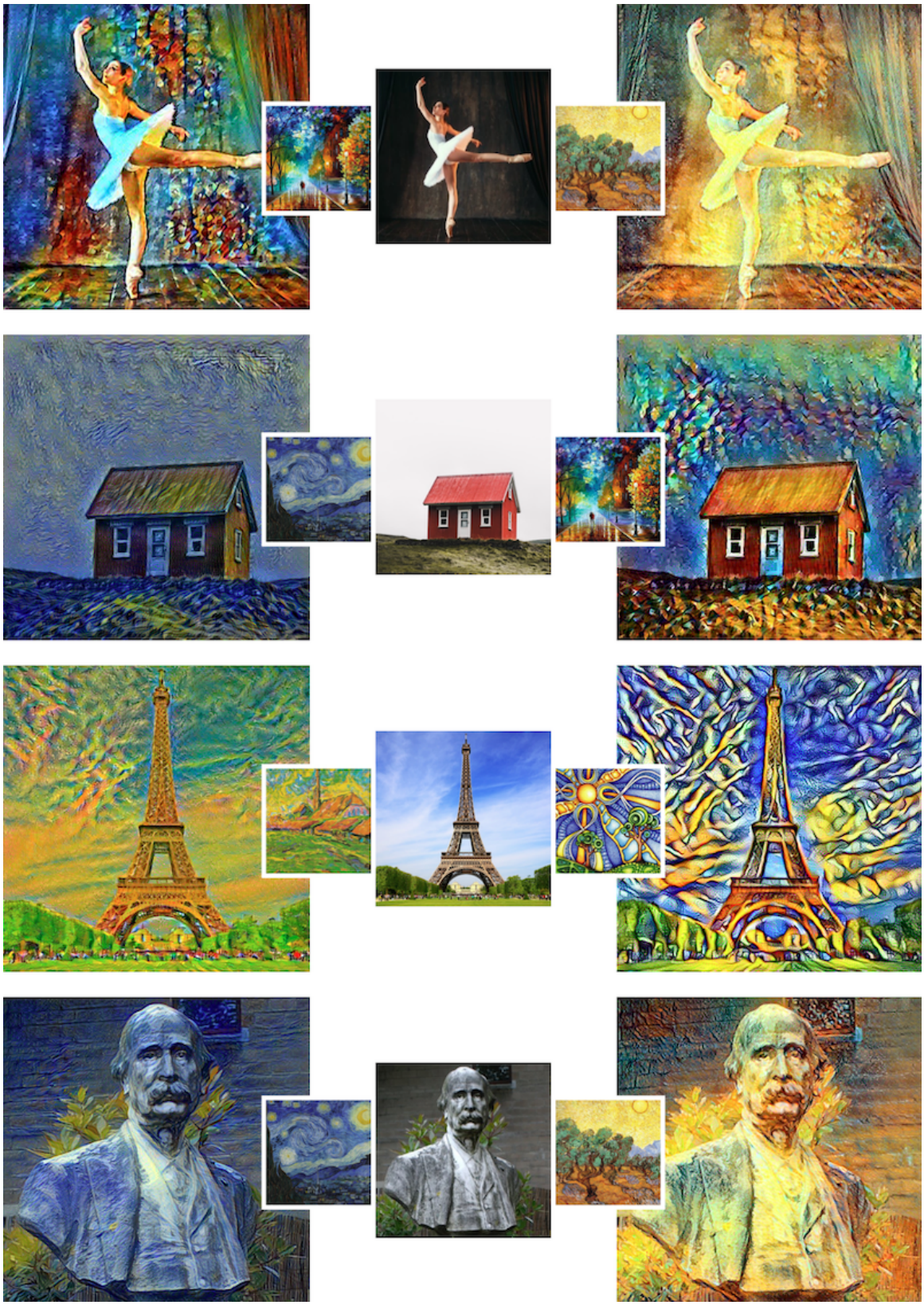


FIG. 17: Some results obtained with Gatys et al. technique.

[1] Vamshik Shetty. Neural style transfer tutorial -part 1. https://towardsdatascience.com/neural-style-transfer-tutorial-part-1-f5cd3315fa7f, 2018.

[2] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015.

[3] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, abs/1703.10593, 2017.

[4] Jun-Yan Zhu. Unpaired image-to-image translation using cycle-consistent adversarial networks. https://junyanz.github.io/CycleGAN/, 2017.

[5] Yongcheng Jing, Yezhou Yang, Zunlei Feng, Jingwen Ye, and Mingli Song. Neural style transfer: A review. *CoRR*, abs/1705.04058, 2017.

[6] Leon A. Gatys, Alexander S. Ecker, Matthias Bethge, Aaron Hertzmann, and Eli Shechtman. Controlling perceptual factors in neural style transfer. *CoRR*, abs/1611.07865, 2016.

[7] Justin Johnson, Alexandre Alahi, and Fei-Fei Li. Perceptual losses for real-time style transfer and super-resolution. *CoRR*, abs/1603.08155, 2016.

[8] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew P. Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. *CoRR*, abs/1609.04802, 2016.

[9] Alexis Jacq. Neural transfer using pytorch. https://pytorch.org/tutorials/advanced/neural_style_tutorial.html, 2017.

[10] Gregor Koehler. Neural style transfer in pytorch. https://nextjournal.com/gkoehler/pytorch-neural-style-transfer, 2020.

[11] John Glover. An introduction to generative adversarial networks (with code in tensorflow). https://cutt.ly/VyTQ8jt, 2016.

[12] Guodong Zhang. Programming assignment 4: Cyclegan. http://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/assignments/a4-handout.pdf, 2018.