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).

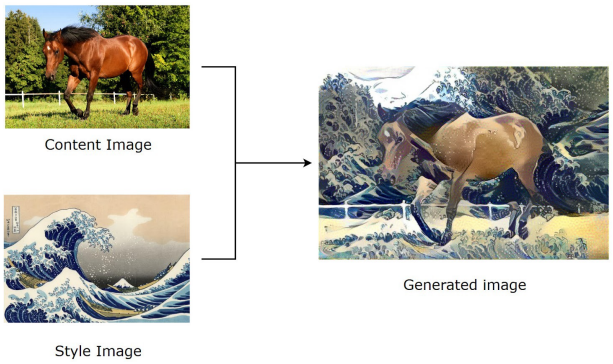


FIG. 1: An example of neural style transfer (Gatys et al. technique) [1].

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 (although it proved to not yield good results), 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.

Although this report is longer than what was expected, our task relied a lot on experiments and discussion of results, and we implemented two architectures and have a lot of pictures, so we felt that restraining ourselves to 8 pages would have meant leaving a lot of our work untalked about, since we also had to explain thoroughly our methods. We are however sorry for this lengthy report.

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). We used it for the discriminator architectures.

III. METHODS

Basic deep learning notions are explained in the appendix.

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 (the first layers of the network) contain low-level details. The authors therefore used a middle-level layer for content representation, since it was closer to the content image while not having all details. 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 :

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

where

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

and

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

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, the output being constructed iteratively based on the style and content images); we simply downloaded images of style and content from the Internet.

B. CycleGAN (Zhu et al.)

CycleGAN is a neural network that allows to perform *unpaired image-to-image translation.* As described on the paper's 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.

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

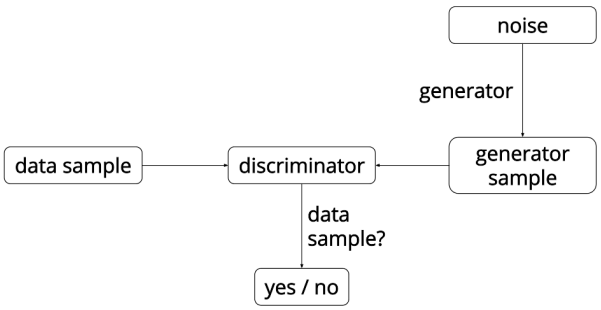


FIG. 3: Generative adversarial networks structure [11].

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. It can however be easily understood that the two networks have opposite goals, and their joint training can thus be tricky.

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).

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 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.

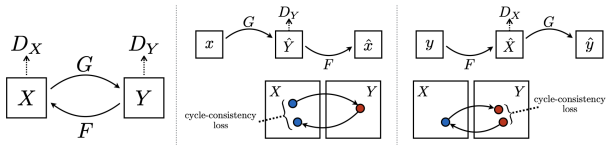


FIG. 4: Cycle consistent loss (CycleGAN) [3].

Finally, the total (generative) 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, the authors of CycleGAN used the 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 used a 70×70 PatchGAN.

The exact architectures are given in the appendix.

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. We first tried to implement them from the papers it referenced before realizing that the exact architecture was given in the appendix. We thus used that source to modify the networks we implemented from their references to have the same basis as them.

Finally, after having implemented the network, we went to the authors' GitHub repository[12] 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. They also used another convolutional block in their discriminator architecture, which is not mentioned in their paper.

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.

For the optimizer, we reused the same one as the authors (Adam), and created a learning rate scheduler acting as they wrote in their paper (linear decrease from 1 to 0 from epoch 100 to epoch 200).

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 and the adversarial loss, and then trains the discriminators by comparing the output of the discriminators for the true and false images to ground truths. The sources we used for this part in order to check that what we were doing was correct were a website explaining the CycleGAN implementation in Keras[13] and a github repository of someone implementing many GANs networks[14], as well as the authors' repository mentioned earlier. From these sources, we have taken the losses used by the authors. We have also taken inspiration for the procedure to train the network from the fourth tutorial of the course. The adversarial loss is an MSE loss, both for the generators and the discriminators, and the cycle loss and identity loss (introduced in the result section) are L1 losses.

IV. RESULTS

A. Neural style transfer (Gatys et al.)

In order to better understand this technique and to obtain the best results possible, 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 with other generated ones, 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. We cannot use objective quantities such as recall or accuracy, so we can only talk about the losses, and it must also be said that the general allure of the curve is more important than the actual values taken, since those depend on the weight we will give the different loss terms, and try to objectivize a subjective goal (the network does not really extract content or style, this is what we perceive, and it is not perfect in doing so). The human judgment is therefore more important than the loss for this application (the loss is useful in a given run of the network, but not so much for comparing different parameters, a task for which the human eye is better suited).

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 input image. According to the literature, these two possibilities are equivalent. We

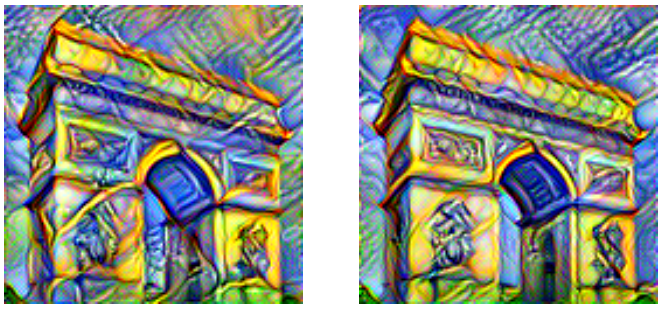
have therefore tried both versions.

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) Copy of content image (b) White noise

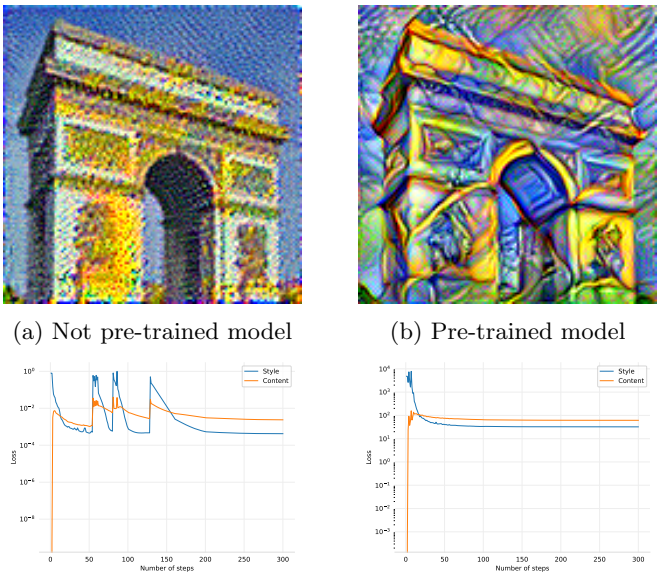
FIG. 6: Starting input image.

As can be seen, the results obtained are very similar. However, starting with a copy of the content image seems to give slightly 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.

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 non pre-trained model allows a less good extraction of the features of each image



(a) Not pre-trained model (b) Pre-trained model

(c) Not pre-trained model (loss functions) (d) Pre-trained model (loss functions)

FIG. 7: VGG19 model variations.
a

[a] All loss graphs are in pdf format, so they stay clear when zooming on them.

and therefore a much worse reconstruction and mix of content and style of each image. However, we can see that both networks are able to extract content (one of the first layers of the network, so the input is not yet very modified) but the not pre-trained model could not really pick up the style, which lead us to believe that networks trained on object recognition make use of the style for their task.

When looking at the losses, we see that the not pre-trained model has losses that are much smaller, but that also start much smaller. Since it has not been trained on image detection first, the backpropagation makes it so that the losses are reduced, although style is not extracted (the information conveyed by layers do not have the same meaning as for the trained model). Using a different scheduler also yields poor results. The value of the loss is thus less important than our visual perception of the result.

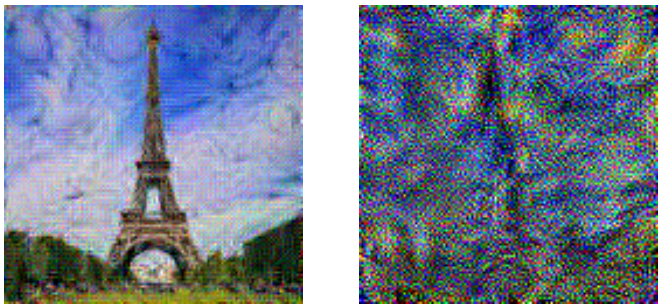
We tried to increase the number of epochs of the algorithm with the non pre-trained model, but the results obtained are systematically much lower than the results with the pre-trained model. To obtain better results, we should have trained the VGG19 model beforehand. Since image classification was not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

We also tried several other architectures to determine whether or not they may provide as good/better results

than the VGG architecture. We first started with the **Alexnet** network, but the results were far from satisfactory, as can be expected from such a simple network (containing only eight layers) [15].

The next one we tried was **GoogLeNet**. However, due to its structure containing parallel blocks and bypassing links, inserting style losses inside the Inception modules while maintaining its pretrained weights proved to be very difficult. We therefore only tried to insert such layers where only one path was possible, but it lead to very poor results.

The results of these two architectures are given at figure 8.

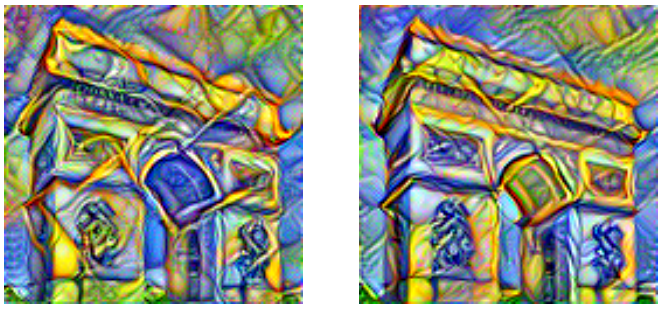


(a) GoogLeNet architecture (b) Alexnet architecture

FIG. 8: General architecture variations.

3. *Architecture of the model used*

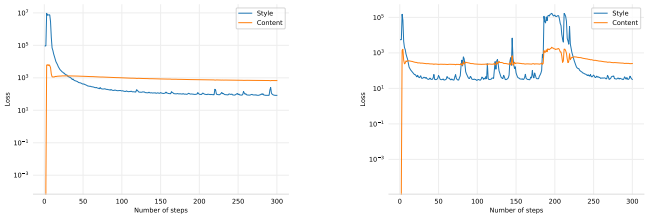
The original architecture of VGG19 has max pooling layers. The authors suggest in their paper to replace these max pooling layers by average pooling layers in order to obtain more appealing results. We have therefore compared the results with these two types of layers (Figure 9).



(a) Max pooling layers (b) Average pooling layers

FIG. 9: 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 10).

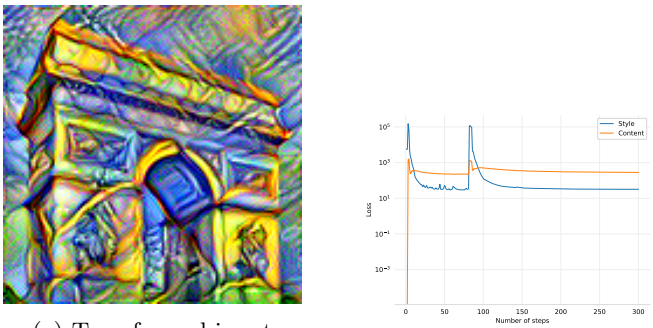


(a) Max pooling layers (b) Average pooling layers

FIG. 10: 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 learning rate *scheduler* (with a step size of 50 and a gamma multiplicator parameter of 0.3) when working with average pooling layers. This scheduler allows to reduce the learning rate as epochs pass, and thus to reduce the variations of the loss functions (the sudden high spikes were due to a too high learning rate, the model could not do anything but have its loss jump out of the potential well). We then restarted the algorithm with this scheduler (Figure 11).



(a) Transformed input image (b) Loss functions

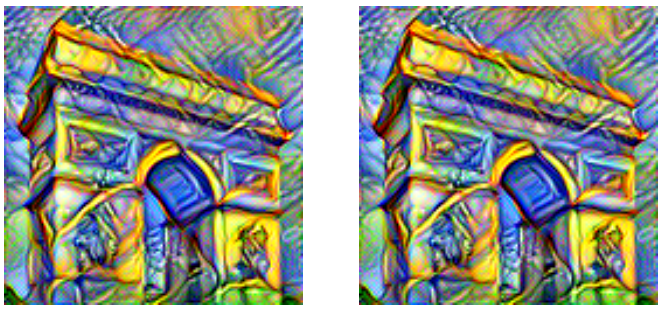
FIG. 11: Average pooling layers with scheduler (VGG19).

We observe that the scheduler seems to be efficient since the variations of the loss functions have almost completely disappeared (a sharper decrease would remove the spike, but is dependent on the other parameters). 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 11 for example), we realize that they stabilize quite quickly (after about 300 epochs), whatever the scheduler we chose (since the learning rate decreases over time, the losses end up converging).

In order to confirm this result, we ran the algorithm by increasing the number of epochs to 1000. The result obtained is shown in Figure 12.



(a) 300 epochs (b) 1000 epochs

FIG. 12: Number of steps (epochs) variations.

As expected, the two results are exactly the same. Indeed, considering the evolution of the loss functions (Figure 13) per 1000 epochs, we can confirm that they stabilize quite quickly. We have therefore set the number of epochs to 300 by default.

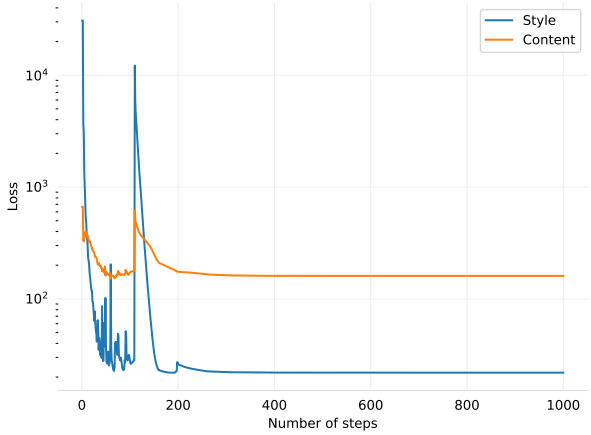
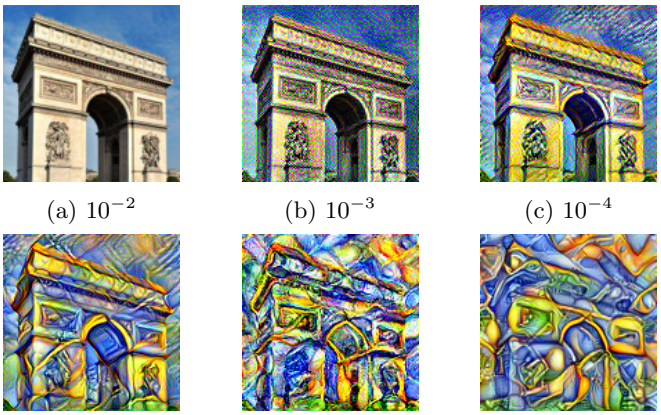


FIG. 13: Loss functions for 1000 epochs.

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 14).



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

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

FIG. 14: 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. Furthermore, we need a value for one of the two weights to determine the other, and the best results were obtained with a content weight of 10.

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 15).

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

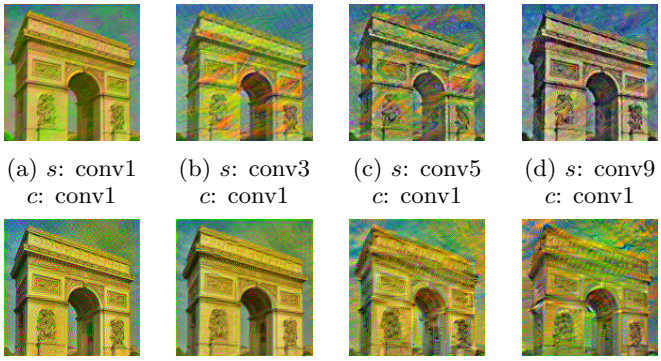
(a) $w = [1, 1, 1, 1, 1]$ (b) $w = [1, .8, .6, .4, .2]$ (c) $w = [.75, .5, .2, .2, .2]$

FIG. 15: Style layers' weights variations.

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 filters 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 16. Only one style layer is used in order to best see its impact, although several are used in practice.



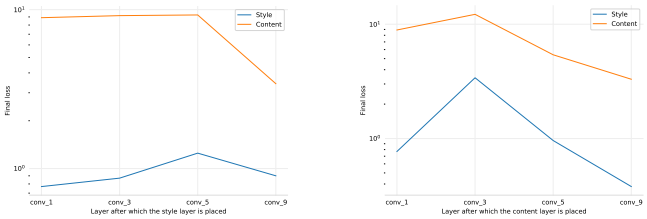
(a) s: conv1 c: conv1 (b) s: conv3 c: conv1 (c) s: conv5 c: conv1 (d) s: conv9 c: conv1

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

FIG. 16: 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).

As shown by the figures 16a, 16b, 16c and 16d, 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 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 (see Figure 17a). Note that from the 9th layer, the loss begins to lessens, but subjectively we can see that the style isn't as well reproduced as for previous layers.

In the same way, when we move the content layer along the network (as shown by the figures 16a, 16e, 16f and 16g), 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, except for the first layer but, subjectively, the content then looks too similar to the one of the original image, as we have just mentioned (see Figure 17b).



(a) Style layer (b) Content layer

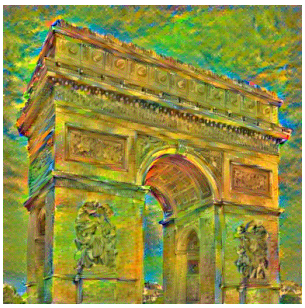
FIG. 17: Evolution of the losses depending on the places of the style and content layers.

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 18b.

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

(a) Starting architecture



(b) Modified architecture

FIG. 18: Comparison of the result that can be achieved by modifying the network architecture with the starting point architecture.

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 (set to 10) and the weight of the style 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 and the desired results.

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 19.

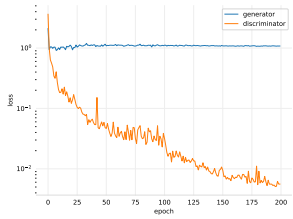
B. CycleGAN (Zhu et al.)

When implementing the paper, we decided to use the dataset "Monet and landscapes", found on the authors' GitHub page[12]. However, when launching our script, we quickly realized that, with the low resources we could obtain at one time on the GPU, we would not be able to train the network on the 7000 images on the training sets. Indeed, training the model for 200 epochs (as recommended by the authors) on 20 batches of 3 images of each domain per epoch took us between 40 and 50 hours. We therefore had to test strategies to have results by the deadline for the project. These strategies and their results are discussed here, as well as our attempts for explaining the different results. The input landscape image to be translated into Monet's style is given at Figure 20. It can be noted that the difference of results between images used in the training or not was not visible to us.

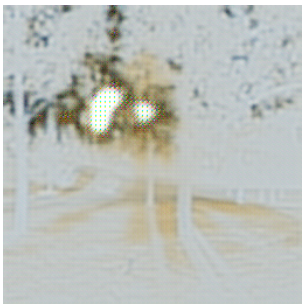
FIG. 20: Input image for CycleGAN

1. *Limiting the number of batches per epochs*

This is the strategy that lead to the 40-50 hours stated above. Here, we decided to use the training set and to restrict the number of batches per epoch at 20 (containing 3 images of each style). One result as well as the losses can be seen in Figure 21b. As can be seen, the result is disappointing but can be explained by the fact that the generator has not been able to improve and reduce its loss, while the discriminator loss decreased greatly over the epochs. This may be a case of vanishing gradient. We believe it may come from the fact that the training dataset was too small. Indeed, over 200 epochs, we only passed twice through all the dataset (containing more than 6000 images).



(a) Losses for the limitation of batches.



(b) Result obtained.

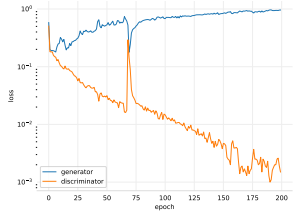
FIG. 21: Results obtained for the limitation of number of batches per epoch.

2. *Training only one model*

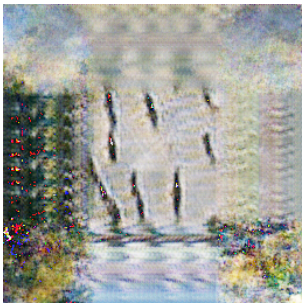
Since we are not interested to go from painting to photograph for this task, we have tried to only train one generator and one discriminator (thus removing the term cycle in CycleGAN) on the testing set (120 images of Monet and 750 images of landscapes). The results were quite promising, but we realized that all input images output close images to one another. We believe this is due to

FIG. 19: Some results obtained with Gatys et al. technique.

the fact that we only had for the generator loss the adversarial one, and so our model may well have overfitted or gone into mode collapse (since all its predictions went to close outputs). To try to solve that, we have launched the same experiment, but implemented the identity loss described by the authors (the loss compares the result of the passage of an image from the Monet painting in the painting-to-landscape generator to the same image, since the network should then output an image that is ideally identical to the input one). We believed that should solve the fact that all input images lead to close outputs. The loss and result obtained for this idea without the identity loss is given at Figure 22b. As can be seen, the two losses evolve in opposite fashion and, at one point, the generator loss decreased vastly while the discriminator one increased. Decreasing the learning rate at that point might have been a good idea, however we had no idea about the behaviour to expect from this test and used the learning rate of the original architecture and did not save intermediate models.



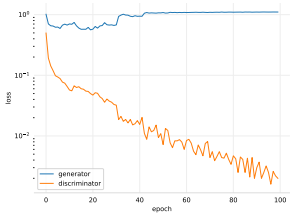
(a) Losses for only one model.



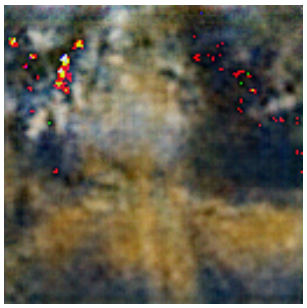
(b) Result obtained.

FIG. 22: Results obtained for training only one generator and discriminator.

The results for the models to which we added the identity loss are given at Figure 23b. The model also includes a LR scheduler that reduces the learning rate when it detects a plateau for both generator and discriminator (although it may have been better to only use it on the generator). The initial scheduler provided worse results, all other things being equal, than the plateau one. As can be seen, the loss for the generator first decreases before increasing. This does not mean that the output images get worse but rather that the discriminator becomes too good, since the results are not that bad. They indeed resemble the input images and have a painting feel, although we had to reduce the training dataset size and the number of epochs to get results before the deadline, so letting it train longer may well yield better results (we will let it run longer to get results for the presentation). Although the generator loss end up increasing, the results are better than for the previous tests. This model is the only one presented here which uses the 128x128 architecture, the others use the 256x256 one.



(a) Losses.

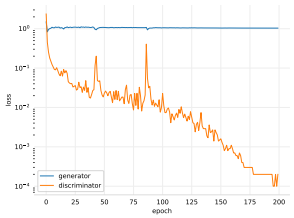


(b) Result obtained.

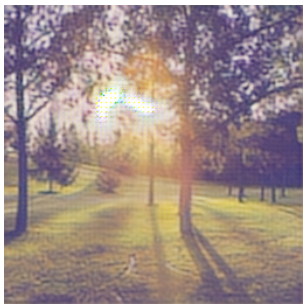
FIG. 23: Results obtained for training only one generator and discriminator, using a LR scheduler and the identity loss.

3. *Training on less data*

Instead of limiting the number of batches but keeping the whole training dataset, as we did for the first strategy described here, we decided to restrict ourselves to a smaller dataset, but each epoch would see each sample through. The datasets chosen were simply the pruned training ones (50 Monet painting and 120 landscapes), and the results are given at Figure 24b. The loss behaves as for the limitation of the number of epochs, and the result is simply a paler version of the input. Maybe, this time, the dataset was not exhaustive enough for the model to pick up Monet's style.



(a) Losses for the training on less data.



(b) Result obtained.

FIG. 24: Results obtained for the limitation on the training dataset size.

Other tests have been made without success and are not reported here, and more results are given in the appendix B and not here because the models did not have the time to run to completion due to a high utilization of the GPU cluster for the last week of the semester, and because our models take tens of hours to train (and thus we are only able to build on top of our discoveries after the training is complete, which was delayed since we were not prioritized due to the high training time required). Better results should however be obtained when these models have completed their training.

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, although our results are very good in our opinion.

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 (although we believe that not using a model pretrained on object recognition would have actually lead to worse 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 since this is a very subjective application, but nevertheless, from a qualitative point of view, the results are very appealing.

B. CycleGAN (Zhu et al.)

The results obtained from CycleGAN were very disappointing at first, and we had to try numerous changes to finally obtain something that produced results that were not completely horrid. The best results were obtained from training a single GAN model (one generator and one discriminator), while CycleGAN provided bad results, probably because we did not provide a big enough dataset due to time restrictions (training on the original dataset would have taken weeks). For our application, since we did not need to go from painting to landscape, a simple GAN network might have been better (at least easier to train) but a CycleGAN has proved to provide good results nonetheless in the authors' paper and experiments. The use of a plateau learning rate scheduler improved somewhat the results, compared with the technique of the authors. The main limitation is that, in this state, this CycleGAN implementation does not provide acceptable results. The improved solutions that are being trained are discussed in the appendix B and will be showed at the presentation if they prove successful. One other big limitation of this method is the time needed to train the network and the impossibility to generalize its results, as will be discussed in the section below.

After comparing our implementation with that of the author and other found on the internet, and asking an assistant, we believe that our implementation is not wrong, and so the only thing we did differently from the authors was not train on the whole training dataset (which was not possible in our case). If our other tests do not yield good results, this will be our hypothesis for our disappointing results. During this project, we learned that training a GAN network was not an easy task, since two networks were working to achieve two opposite goals. We expected to get better results more easily.

C. Comparison between the two solutions

As can be understood from the papers, CycleGAN is a general tool used to go from domain to domain, whatever these domains. It has therefore not been tailored with neural style transfer in mind, but works quite well on that task, although the training time and resources required to obtain satisfactory results are much more expensive than for the other paper. Indeed, CycleGAN requires a dataset of images from both domains to be able to train, and the greater the size of those datasets, the better the results, whereas the solution proposed by [2] needs only the content image and the style image. It is therefore much more flexible than CycleGAN because it can take almost any content and style images and provide good results immediately (with some tweaking of parameters maybe required), while CycleGAN is trained on two specific domains. In our implementation we chose Monet paintings and landscapes, and the model trained on those, which means that it will not be able to translate input images in another style than Monet without training it on that particular style, and that is something that is not necessarily possible for generic styles (a painter has a style that it uses in hundreds of paintings, whereas a random image may be one of a kind, leading to an impossible translation by CycleGAN).

With that being said, CycleGAN provides better, more natural results than the other solution for two given domains, because it learns the style of the painter (or the artist) instead of the style of a given painting or art piece,

but at the cost of a training process involving thousands of images and hundreds of hours, which make it less versatile than Gatys' solution, which works on the fly.

Appendix A: Deep Learning terms used

Since we were asked to describe what we did as if the reader had never taken a Deep Learning course, we gather here the different concepts used in this report, and give them personal definitions (helped at times with the slides of Introduction to Machine Learning given by Mr Geurts and Wehenkel during the first semester) :

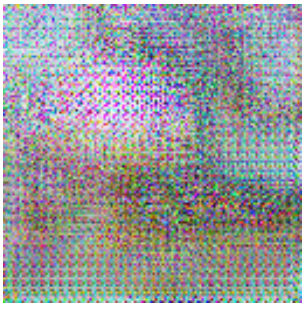
- **Convolutional layer** : A convolutional layer simply applies a filter to its input. That filter is a matrix, which computes, for each element of the input (representing usually a pixel), a weighted sum of its neighbours. This allows to detect local features. The output of such a filter is thus called a **feature map**.
- **Pooling layers** : A pooling layer (or subsampling layer) is used to aggregate values in a local region, thus reducing the size of the output compared to the input. The two types of pooling layers used in our implementations were the average pooling (the aggregated value is the average of the region) and the max pooling (the aggregated value is the local maximum of the region).
- **Loss** : A loss is the function that is used in order to evaluate the candidate solutions (set of weights of the network, network architecture,...) to the tasks. The optimal solution maximizes (or minimizes) such a loss and is approximated through (or found by, depending on the algorithm) optimization algorithms. Such a function helps describing the goodness of a complex structure with a single number[16].
- **Fully-connected layer** : Such layers are usually used to combine the features previously extracted by convolutional layers in a non-linear way. Indeed, these layers act as universal function approximators (when there are at least two of them).
- **Normalization layer** : Such a layer is used to normalize the input via a mean and a standard deviation. Those two values used in our network have been empirically derived by the authors of VGG.
- **ReLU** : A rectifier linear unit is an activation function that is zero for all negative inputs and the value of the input otherwise. An activation layer is used to transform the output of a layer and add it non-linearities.
- **GAN** : This term has already been extensively defined when talking about CycleGAN.
- **Backpropagation** : This algorithm is used to adjust the different parameters of a network in order to decrease the loss (or increase it if we want a high loss).
- **Learning rate** : The learning rate is the step size at each iteration of the optimization algorithm (gradient descent). A high rate will allow high steps in the loss plane while a smaller rate will force the loss to move in its near neighbourhood. For such a reason, we usually set a high rate at the beginning and decrease it over time, so that the algorithm can converge to a local minimum and not jump out of the potential well (if the learning rate is too large, the loss may inevitably exit the well).

Appendix B: CycleGAN additional results

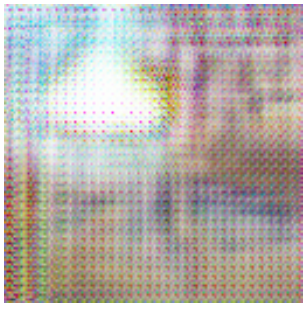
This appendix contains further tests of different parameters of CycleGAN.

1. Dataset size

Although it could not be tested thoroughly, we believe that, for a given model, the bigger the training set the better the results. The results given in Figure 25b. They were tested for 60 epochs, since we had a time limit, and the results are shown on one of the training images. As can be seen, the larger dataset yields better results, although, for a size of 120+750 images, we get worse performances, probably because the number of epochs should be made bigger for larger datasets, and because the loss suddenly increased in the training, since we used the learning rate of the authors. No conclusion can thus be made, but our belief is that larger training datasets yield better results and better generalisation. Models should have been trained on 200 epochs, as the authors do, but the utilization of the GPUs during the last week was too high for these models to finish. The results of this ongoing experiment will be displayed at the presentation.



(a) Dataset of 10+30 images.



(b) Dataset of 50+150 images.

FIG. 25: Training dataset sizes results.

2. Other ongoing experiments

Some of the models that are still training are :

- The adding of the identity loss to the general model.
- The same as above, but with a smaller dataset.
- The training of the general model with some tensors detached at different places to try to get the generators to perform better, with and without the identity loss.
- The same as above, but with a plateau LR scheduler.

Although they were launched on a Monday, these models had to wait until Thursday to be launched on the cluster, and thus have not completed the training as of Friday (and training without a GPU was not possible). However, should the results be satisfactory, which we believe they will, since these changes build upon previous successful discoveries, they will be showcased on the day of the presentation.

Appendix C: CycleGAN full architecture

These architectures are taken verbatim from the authors' paper.

For the **generator** :

Let c7s1-k denote a 7x7 Convolution-InstanceNormReLU layer with k filters and stride 1. dk denotes a 3 x 3 Convolution-InstanceNorm-ReLU layer with k filters and stride 2. Reflection padding was used to reduce artifacts. Rk denotes a residual block that contains two 3 x 3 convolutional layers with the same number of filters on both layer. uk denotes a 3 x 3 fractional-strided-ConvolutionInstanceNorm-ReLU layer with k filters and stride $\frac{1}{2}$.

The network with 6 residual blocks consists of :

c7s1-64,d128,d256,R256,R256,R256,R256,R256,
R256,u128,u64,c7s1-3

The network with 9 residual blocks consists of :

c7s1-64,d128,d256,R256,R256,R256,
R256,R256,R256,R256,R256,R256,u128,u64,c7s1-3

For the **discriminator**:

Let Ck denote a 4 x 4 Convolution-InstanceNorm-LeakyReLU layer with k filters and stride 2. After the last layer, we apply a convolution to produce a 1-dimensional output. We do not use InstanceNorm for the first C64 layer. We use leaky ReLUs with a slope of 0.2.

The discriminator architecture is :

C64-C128-C256-C512

However, in their implementation, the authors added after C512 another block containing a Conv layer outputing 512 channels, followed by an InstanceNorm and a LeakyReLU, before the final convolutional layer.

[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] Jun-Yan Zhu. Cyclegan original github repository. https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix/blob/master/models/cycle_gan_model.py.

[13] Jason Brownlee. How to implement cyclegan models from scratch with keras. https://cutt.ly/1yIqEZK.

[14] Erik Lindernoren. Gans implementations. https://github.com/eriklindernoren/PyTorch-GAN/tree/a163b82beff3d01688d8315a3fd39080400e7c01/implementations/cyclegan.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[16] Jason Brownlee. Loss and loss functions for training deep learning neural networks. https://cutt.ly/ayIqREw.

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