Image Style Transfer Using Convolutional Neural Networks

By : Clément Chadebec (clement.chadebec@mines-paristech.fr)
and : Thibault Lahire (thibault.lahire@student.isae-supaero.fr)

Abstract

Neural style transfer is an optimization technique taking three images, a content image, a style reference image (such as an artwork by a famous painter), and an input image which will receive the blend of the first two ones in an artistic and eye-catching manner. The objective is to transform the input image so as to look like the content image, but *painted* in the style of the style image.

Notations

As is the case in the publication [7] we study, we note $\overrightarrow{p}$ the original image (a picture of a landscape, for example), $\overrightarrow{a}$ the artwork, whose style is going to be extracted, and $\overrightarrow{x}$ the image generated, supposed to blend the style of $\overrightarrow{a}$ and the content of $\overrightarrow{p}$.

Outline

This report was done for the course "Introduction à l'imagerie numérique" taught by Yann Gousseau and Julie Delon at the master MVA (Mathématiques, Vision, Apprentissage/Learning). To tackle the huge subject of neural style transfer, we decided to focus on the following points. First, the previous attempts of doing style transfer are presented, in order to understand why introducing CNN to do style transfer is an amazing idea. We also present the industrial and academic interests of such a technique (chapter I). Then, we explain in details the structure of the CNN used in [7], as well as the theoretical framework of neural style transfer (chapter II). Finally, we analyze in details the technical choices that have been made in [7] and discuss them (chapter III).

1 Context

1.1 Before neural style transfer

Painting is a form of art that have aroused curiosity of computer scientists since the 90's [9]. The first advances made in style generation are due to the non-photorealistic rendering (NPR) techniques. Nowadays, these techniques are an established field in computer graphics. Fig. 1 shows the type of results yielded by NPR.

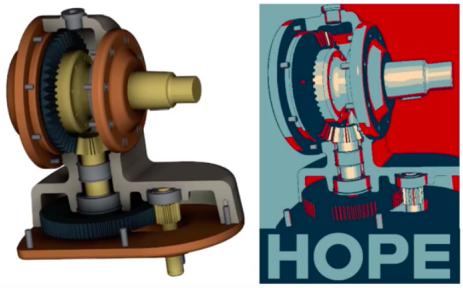


Figure 1: Illustration of what can be done with *Non-Photorealistic Rendering*

Figure 2: Illustration of what can be done with *Stroke-Based Rendering*

In the community of computer vision, the style transfer problem has also generated interest in the same period. Style transfer is seen as a texture synthesis problem, where the goal is to extract and transfer the texture from a source to a target. Before the rise of neural style transfer introduced first by Gatys et al. in [7] in 2015, style transfer relied on three major ideas.

In **Stroke-Based Rendering**, the idea is to artificially introduce strokes (such as brush strokes, tiles...) in a digital canvas to render a photograph with a particular style [13] (see Fig. 2). In **Region-Based Rendering**, the image is segmented to easily capture and transfer the texture [6] (see Fig. 3). In **Example-Based Rendering**, the algorithm learns in a supervised way the mapping between pairs of source images and target images [14].

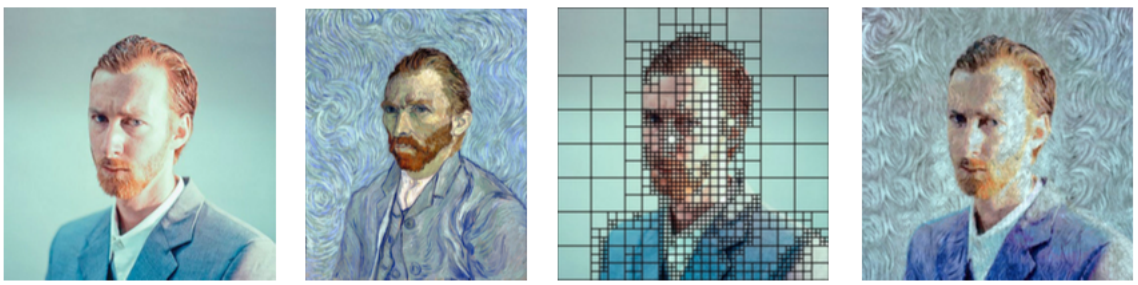


Figure 3: Illustration of what can be done with *Region-Based Rendering*

The problem of the last one is a bad scalibility, since bad performance is obtained on pairs never seen before. The issue of the first two techniques is a transfer too much focused on details, and not on global features (e.g. presence of buildings or trees in the picture).

The main success accomplished by Gatys et al. in 2015 is the possibility to capture semantic features of the images thanks to high layers in Convolutional Neural Networks (CNN).

1.2 Style transfer, a source of interest

Before we explain in details what a CNN is and how to proceed to extract high-level features from images (chapter II and III), it is important to mention why neural style transfer has become over the last few years a key industrial and research issue.

Industrial applications
Since 2015, neural style transfer has generated an interest from people far away from academic research. This technique has rapidly become a trend on social networks such as Facebook or Twitter, and applications doing neural style transfer have been developed (e.g. *Prisma* or *Ostagram*). Following this trend, Facebook AI Research (FAIR) developed mobile-embedded deep learning system (*Caffe2Go*) to run deep neural network on mobile phones.

Moreover, neural style transfer is a powerful tool to help computer artists creating new textures, or more generally game video designers.

Research interests
Neural style transfer has some limitations and issues. The first one is related to a more general problem in imaging: how do we evaluate the quality of the results? As shown in Fig. 4, it seems complicated at first sight to define a good set of criteria assessing the quality of a result. A lot of research remains to be done in identifying underlying aesthetic principles. Another issue is related to the problem of interpretability of deep neural networks. Indeed, one may need to control the underlying representations yielded by the CNN in order to diminish the *black-box* aspect of deep networks.

Furthermore, it has been proven that adversarial examples can be generated for neural style transfer. This means that, for some input images $\overrightarrow{x}$, the process fails [10], [8], as shown in Fig. 5. This is a big issue because it proves that every input images cannot be used (in particular, using a random image, which has been done by Gatys et al. and in this report, is in theory a bad idea. In practice, however, generating the adversarial example happens with very low probability...).

Eventually, researchers are improving neural style transfer to deal with the *three trade-off* : speed, flexibility, and quality. Given the ground-base technique introduced in 2015, the idea is now to look for the best hyperparameters ensuring the faster and the more flexible algorithms producing the higher quality.

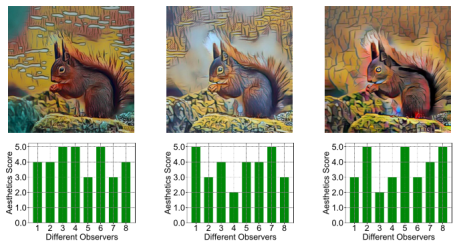


Figure 4: 8 people (same professional area) have been asked to note the quality of these 3 images, different results have been obtained

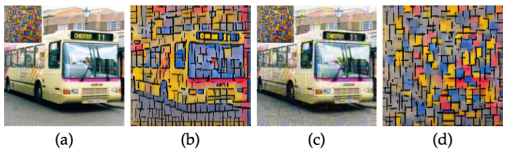


Figure 5: Work done by a CNN on style transfer (a, b). Work done by an adversarial network to fool the CNN (c, d). On the right picture, the CNN has clearly been fooled and the style transfer has failed

2 CNN and deep representations

This section presents the Convolutional Neural Network (CNN) used in this study and the way it works. Then, we present the theoretical bottleneck of neural style transfer.

2.1 Description of VGG19

Neural style transfer is enabled by a CNN introduced for the ImageNet competition called VGG19 [12]. We first look at its structure thanks to the following lines of code :

```
from keras.applications.vgg19 import VGG19
model = VGG19()
model.summary()
```

The program outputs the following construction presented in Fig. 6, or, in a more artistic manner, in Fig. 7.

```
Model: "vgg19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv4 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv4 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv4 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 143,667,240
Trainable params: 143,667,240
Non-trainable params: 0
_________________________________________________________________
```

Figure 6: Architecture of VGG19. Source: keras

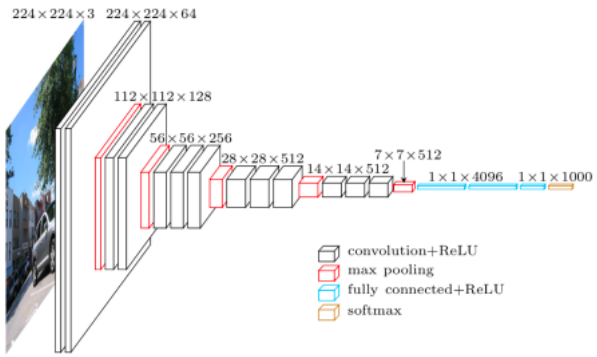


Figure 7: Artistic view of the architecture of VGG19. Source: [4]

The input of this CNN is a tensor of size $224 \times 224 \times 3$. In other words, it is a picture of height 224 and width 224 encoded on the classical RGB channels. Then, VGG19 is made of a succession of convolutional layers and maxpooling layers. After 5 blocks using these operations, the output tensor is flattened into a vector which feeds a densely connected neural network, also known as MLP (Multi-Layer Perceptron). The final output of VGG19 is a probability distribution over 1000 possible states. We now describe step by step how such a CNN works.

Convolutional layer [3]
VGG19 begins with 5 blocks alternating convolutional layers and maxpooling layers. As written above, the input of this CNN is a tensor of size $224 \times 224 \times 3$. The first operation on this tensor is done by a convolutional layer, whose output is another tensor of size $224 \times 224 \times 64$. We say that the input feature map has a depth of 3 corresponding to colors, whereas the output feature map has a depth of 64 and a patch size of 224×224. After this first convolutional layer, there are 64 features. These features have been obtained thanks to a convolutional operation with 64 different kernels.

Definition A kernel is a *small* 3D tensor whose aim is to apply a convolution to the input tensor. The depth of this tensor is equal to the depth of the input tensor, and its height $\times$ width is generally 3×3 or 5×5. In our case, we deal with a kernel of size $3 \times 3 \times 3$. If our objective were to train such a layer (which is not the case, we will come later to that point), the coefficients of the kernel tensor would be the parameters to be learnt. But there would be other parameters to learn !

Biases The other parameters to learn in a convolutional layer are the biaises. More precisely, there is one bias per kernel (as is the case in other deep architectures were there also is one bias per weight matrix. The bias in a convolutional layer plays the same role as the role it plays in other deep architectures).

Number of parameters As a consequence, for the convolutional layer we are considering, there are $(3 \times 3) \times 3 \times 64 + 64 = 1792$ parameters to tune. Let's sum up this in a more general way by defining C the number of channels in the input, N the number of kernels, or equivalently, the number of channels in the output, K the size (width or height) of the kernels used in the convolutional layer, and P the number of parameters in the layer [2]. We have : $P = K^2 \times C \times N + N$. In other words, each kernel has $K^2 \times C$ coefficients and there are N kernels.

How does a kernel work and what is a convolution operation ? The kernel

works by sliding $3 \times 3 \times 3$ windows of the input tensor, stopping at every possible location. Fig. 8 illustrates the process on a toy problem. However, with this process, we should have obtained in output a tensor of size $222 \times 222 \times 64$, which is not the case. Since we have the outputs height and width equal to the inputs height and width, the convolutional layer we deal with is using padding. Padding consists of adding an appropriate number of rows and columns on each side of the input feature map so as to make it possible to fit center convolution windows around every input tile. For a 3×3 window, you add one column on the right, one column on the left, one row at the top, and one row at the bottom.

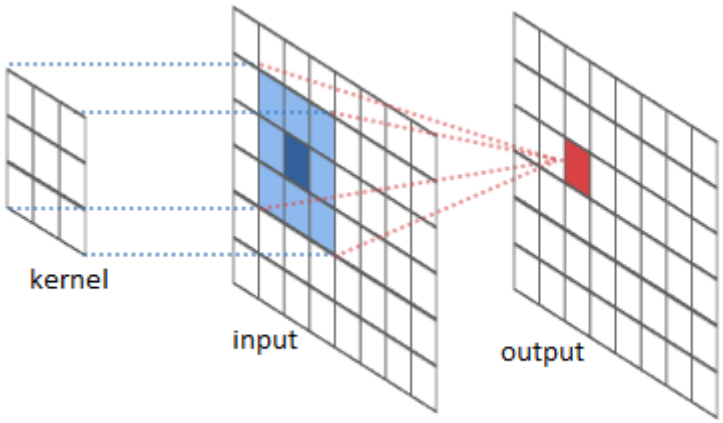


Figure 8: Image convolution with a $3 \times 3 \times 1$ kernel. We have to imagine that the kernel moves all around the input image. Source: [1]

First block of VGG19

After the first convolutional layer, we have a tensor of size $224 \times 224 \times 64$. The architecture is such that there is another convolutional layer after this one, whose output is of size $224 \times 224 \times 64$ as well. Now, we deal with a kernel of size $3 \times 3 \times 64$ and we reuse padding to have output height and width equal to the input height and width. We can check that the number of parameters is indeed $(3 \times 3) \times 64 \times 64 + 64 = 36928$.

Maxpooling layer

The pooling layer takes in input a tensor of size $224 \times 224 \times 64$ and outputs a tensor of size $112 \times 112 \times 64$. This operation is very simple : it aggressively downsamples the input tensor by dividing its height and width by 2. To do this, max pooling extracts windows from the input feature maps and outputs the max value of each channel. It is conceptually similar to convolution, except that instead of transforming local patches via a learnt linear transformation (the convolution kernel), they are transformed via a max operation. Max pooling is not the only way to achieve such an agressive downsampling. In particular, and this is what is used in neural style transfer, we can use an average pooling, where each local input patch is transformed by taking the average value of each channel over the patch, rather than the max. Fig. 9 illustrates the difference between max and average pooling on a toy problem. Note that there is no parameter to learn for this step.

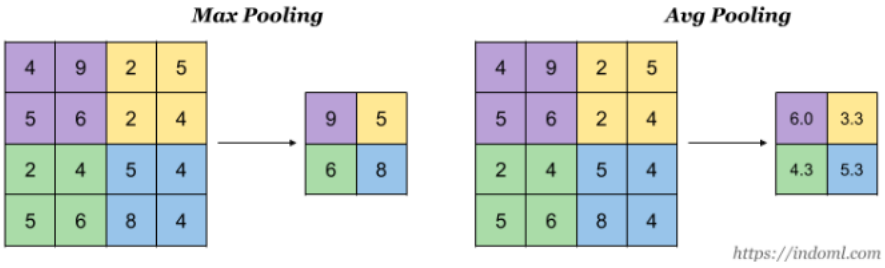


Figure 9: Max and average pooling on a simple example. Source: [1]

Flattening

After 5 blocks of convolutional and max (or average) pooling, we end up with a $7 \times 7 \times 512$ tensor that is flattened in a vector of dimension 25088 because $7 \times 7 \times 512 = 25088$. This vector is the input of a densely connected network of two layers.

Layer in a densely connected network

After the flattening, the vector of size 25088 is transformed into another vector of size 4096. This operation requires a lot of parameters. Indeed, such an operation is typically done by learning a matrix of size 4096×25088, as we can imagine from Fig. 10. Moreover, as it is always the case in neural networks, there is one bias for each component of the output. In other words, for this first fully connected layer, there are 102764544 parameters.

The second dense layer follows the same rules, that's why we have $4096 \times 4096 + 4096 = 16781312$ parameters. Idem for the last layer for the predictions, there are $4096 \times 1000 + 1000 = 4097000$ parameters.

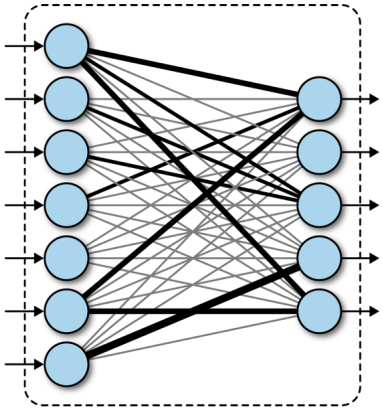


Figure 10: Dense layer in a fully connected network

Output layer and goal of VGG19

The output layer is a vector of size 1000, because VGG19 was created to do image classification on 1000 classes. VGG19 was tuned to be efficient for the ImageNet competition. For this competition, each team tunes its network during days on a huge image dataset, hoping to reach the best results on the test dataset. VGG19 achieved a good score, and its creators made its tuning public. In the neural style transfer, we reuse the parameters tuned for the competition.

What did the network learn ?
We call low-level features the lines, curves and details that can be found in a picture. We call high-level features or semantic features more abstract characteristics of the picture, such as the presence of a human face or not. The low-level features are encoded in the first layers (or early layers) of the network, whereas the high-level features are encoded in the last layers (or late layers). In Fig. 11, an example of what a CNN can learn is shown. This picture is not extracted from VGG19, because the latest feature map of VGG19 are of depth 512, which means to much pictures to show. However, for the network presented here, the feature maps shown are of depth 64. As we see, the first layers learn basic characteristics (lines, curves, etc...) and details, whereas the last layers learn more abstract features such as textures. Fig. 12 presents what VGG19 produces at certain layers.

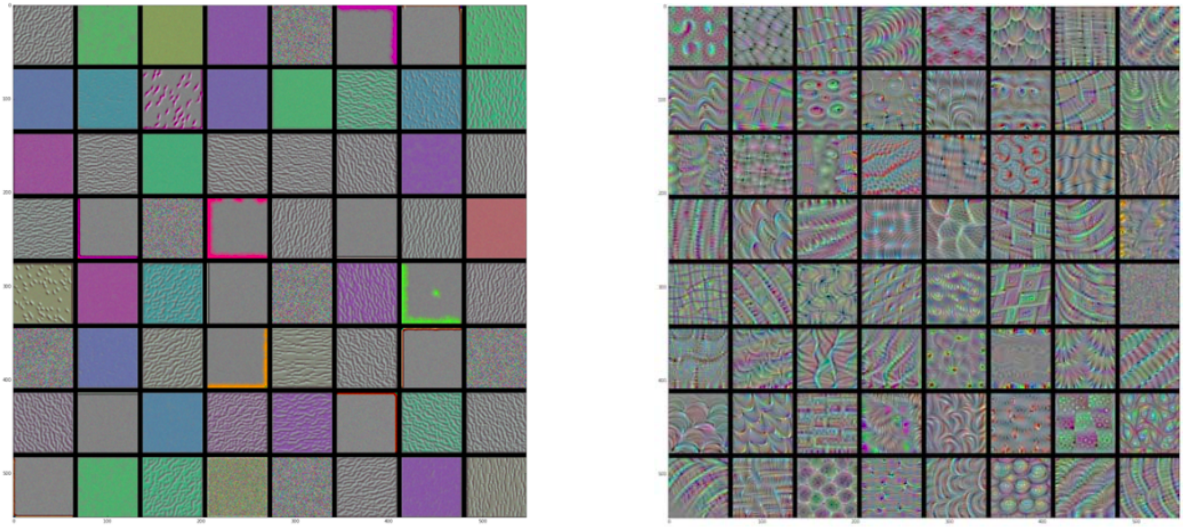


Figure 11: Visualization of the feature map for a low layer (left) and a high layer (right), source: [5]

Why using a pre-trained network ?
VGG19 was trained on image classification, so it has learnt to recognize both high-level features and low-level features in a large variety of images. In other words, the features in the layers of our network are already responsive to real-world content. Doing neural style transfer boils down to modifying the input image (and not the parameters of the networks !) so that the image activates a certain number of features. More precisely, the input image is modified so that it activates the texture of the artwork and the objects of the landscape of the real-world picture.

2.2 Content representation

One of the objective of neural style transfer is to produce an image $\overrightarrow{x}$ with a content close to the content of the artwork $\overrightarrow{p}$. In order to do this, a layer of the CNN is chosen (and this choice will be discussed later) to be the reference of the features of $\overrightarrow{p}$ to be extracted. Since high-level layers encode high-level features, i.e. what is called *content* in this study, it is a good idea to take a quite deep convolutional layer for this task. More practically, we are considering the output of a given layer l, i.e. a tensor of size

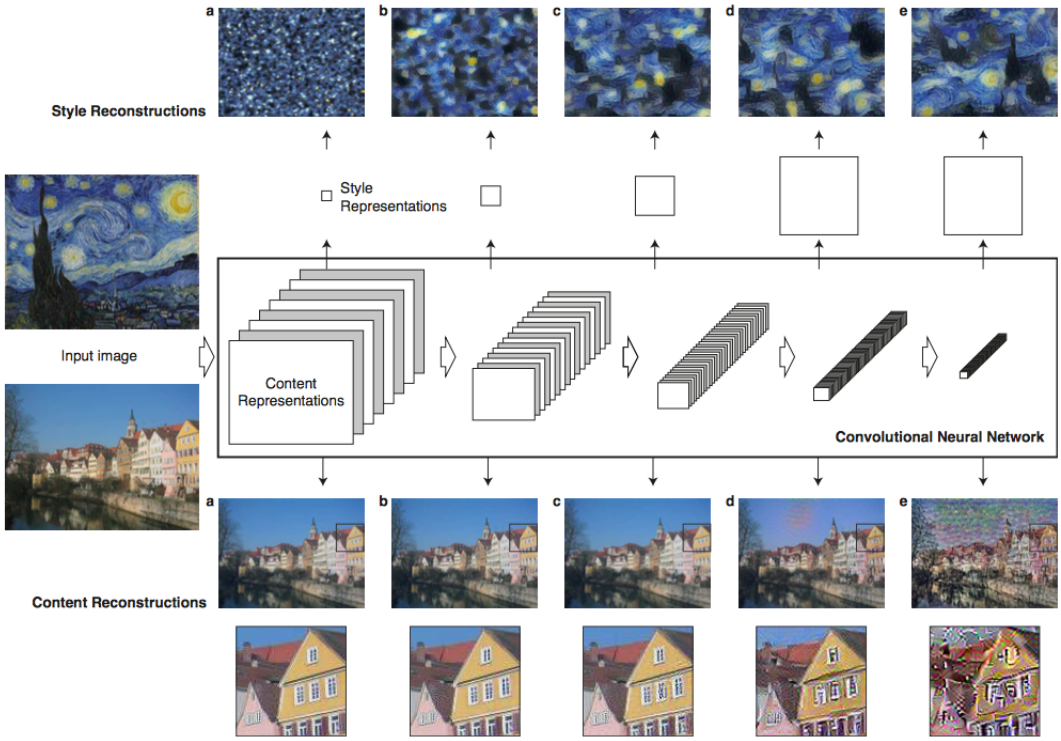


Figure 12: Modification of the input images "content" and "style" through the preparametrized network VGG19

$\sqrt{M_l} \times \sqrt{M_l} \times N_l$. This means there are N_l features or filters; and we use the strange notation $\sqrt{M_l}$ to indicate that the square of size $\sqrt{M_l} \times \sqrt{M_l}$ can just be viewed as a vector of size M_l. (We introduced the weird $\sqrt{M_l}$ just to stick to the notation of the paper...) We note this output feature map F^l and we see it as a matrix : $F^l \in \mathbb{R}^{N_l \times M_l}$. Therefore, $F^l_{i,j}$ denotes the coefficient of the i-th filter (or feature) of the output of layer l at position j.

It is now possible to define the content loss. Let P^l be the feature map of the content image $\overrightarrow{p}$ produced by VGG19 at the output of layer l and F^l be the feature map of the image $\overrightarrow{x}$ produced by VGG19 at the output of layer l. We define a squared-error loss :

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

The derivative of this loss with respect to the coefficient $F^l_{i,j}$ is easy to compute :

$$\frac{\partial \mathcal{L}_{content}}{\partial F^l_{i,j}} = \begin{cases} (F^l - P^l)_{i,j} & \text{if} \quad F^l_{i,j} > 0 \\ 0 & \text{if} \quad F^l_{i,j} < 0 \end{cases} \tag{2}$$

From this expression, the gradient with respect to the image $\overrightarrow{x}$ can be computed using standard error back-propagation. Thus we can change the initially random image $\overrightarrow{x}$ until it generates the same response in a certain layer of the Convolutional Neural Network as the original image $\overrightarrow{p}$.

2.3 Style representation

In the previous part, we were looking at extracting the content of the image $\overrightarrow{p}$, that's the reason why we decided to select a quite deep layer (a choice that will be discussed later) and considered its output feature map. Now, we want to extract the style of the artwork $\overrightarrow{a}$. Intuitively, it is understandable that extracting a style involves both low- and high-level features of $\overrightarrow{a}$. As a consequence, we consider several layers of the network. We introduce the style loss as :

$$\mathcal{L}_{style}(\overrightarrow{a}, \overrightarrow{x}) = \sum_{l=0}^{L} w_l E_l \tag{3}$$

where E_l is the contribution of layer l in the loss, and w_l a weight associated to the loss E_l in the total style loss. To understand the way E_l is computed, we need to introduce the Gram matrix $G^l \in \mathbb{R}^{N_l \times N_l}$, which gives us feature correlations. Indeed, $G^l_{i,j}$ is computed as the inner dot product of the filters i and j in layer l :

$$G^l_{i,j} = \sum_k F^l_{i,k} F^l_{j,k} \tag{4}$$

If we note A^l and G^l the style representations of $\overrightarrow{a}$ and $\overrightarrow{x}$ (respectively) in layer l, the contribution of layer l to the style loss is :

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G^l_{i,j} - A^l_{i,j})^2 \tag{5}$$

A^l and G^l are both Gram matrices, and E_l encodes the squared differences between these two matrices, in other words the squared difference between correlations. As written in eq. [3], we combine the losses for each layers to obtain the total loss. By including the feature correlations of multiple layers, we obtain a stationary, multi-scale representation of the input image, which captures its texture information but not the global arrangement.

Note that the derivatives of E_l are easy to compute. We have :

$$\frac{\partial \mathcal{L}_{style}}{\partial E_l} = \begin{cases} \frac{1}{N_l^2 M_l^2} \left((F^l)^\top (G^l - A^l)\right)_{j,i} & \text{if} \quad F^l_{i,j} > 0 \\ 0 & \text{if} \quad F^l_{i,j} < 0 \end{cases} \tag{6}$$

This means, as in the previous part, that back-propagation can be used to compute these derivatives and update the input image $\overrightarrow{x}$.

2.4 Style transfer

Creating a new image $\overrightarrow{x}$ that matches the content of $\overrightarrow{p}$ and the style of $\overrightarrow{a}$ can therefore be done by minimizing both the content loss and the style loss. The total loss we have to minimize to do neural style transfer is then :

$$\mathcal{L}_{total}(\overrightarrow{p}, \overrightarrow{a}, \overrightarrow{x}) = \alpha \mathcal{L}_{content}(\overrightarrow{p}, \overrightarrow{x}) + \beta \mathcal{L}_{style}(\overrightarrow{a}, \overrightarrow{x}) \tag{7}$$

We will discuss the choice of α and β in the next chapter. Performing the optimization on this loss is easy because we know analytically the expression of the gradient of $\mathcal{L}_{total}$ with respect to $\overrightarrow{x}$. In the implementation we use in keras, a first-order method

is performed, whereas the authors of the original paper used L-BFGS. For an *efficient* first-order optimizer, we do not see any difference with L-BFGS.

To conclude this chapter, Fig. 13 is a summary of the way neural style transfer works (extracted from the original paper).

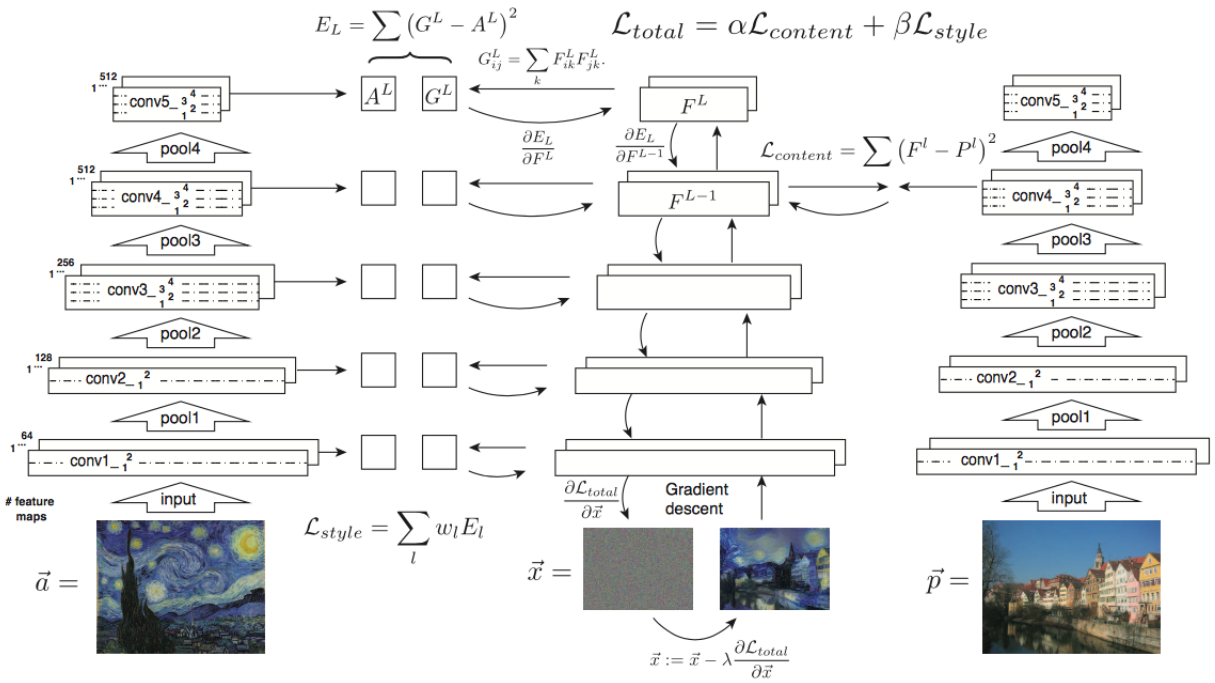


Figure 13: Summary of the neural style transfer method

3 Results and discussions

3.1 Trade-off between content and style matching

As described above, the style transfer is performed by minimizing a total loss $\mathcal{L}_{total}$ which is composed by two distinct weighted losses that quantify respectively content image $\overrightarrow{p}$ matching $\mathcal{L}_{content}$ and style image $\overrightarrow{a}$ matching $\mathcal{L}_{style}$. In order to correctly represent the style image $\overrightarrow{a}$ while not totally overwriting the content image $\overrightarrow{p}$, a trade-off has to be found. This means that we have to look for α and β (weights in total loss) parameters that will give access to a satisfying visual effect. As this objective is rather subjective, α and β are chosen by trying different values and compare the results. This is what has been performed by the authors of the article. One can remark that considering that the total loss $\mathcal{L}_{total}$ is simply a linear combination of the content and style loss, it is quite easy to modify the weights manually to rather emphasise one or the other. In that particular respect, we conducted four experiments reproducing what has been performed by the authors but with other images. We tested the style matching while considering different *weight factors* α/β ranging from 10^{-1} to 10^{-4} and compared the obtained results with the ones presented in the article. Our experiments considered a white gaussian noise as starting point, the loss function as described in 2.4 and a fixed number of iterations

$n = 3000$. The results as described by the authors can be seen in Fig. 14 while ours are presented in Fig. 15.

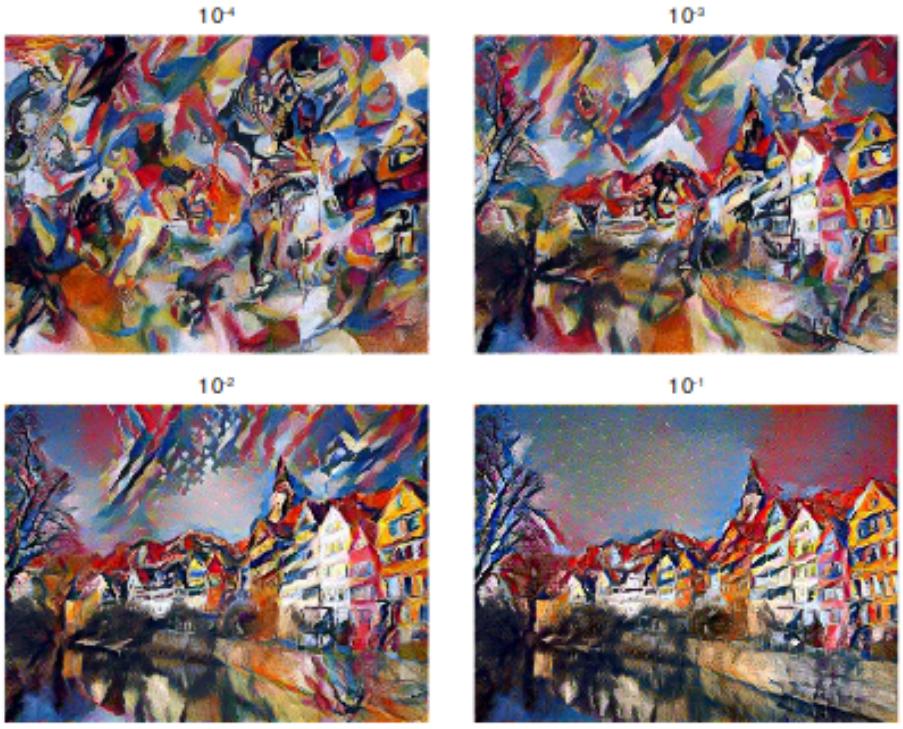


Figure 14: Influence of weights in image representation (authors' experiments)

We can see that the results we obtained with the Van Gogh painting as style image reference are in line with the conclusion of the article. Indeed, with weights $\alpha/\beta = 10^{-1}$ and $\alpha/\beta = 10^{-2}$, the outcome is rather a juxtaposition of the two reference images without really mixing them. This is even more flagrant on the bottom left image of Fig. 14. On the other hand, if we consider a weight $\alpha/\beta = 10^{-4}$, the style image completely overrides the content image which is not satisfactory.

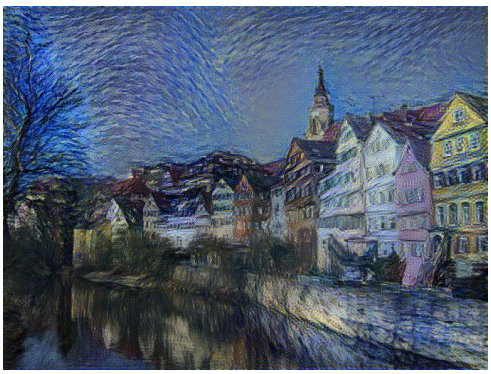
3.2 Effect of different layers of the Convolutional Neural Network

In this particular section, we will discuss a bit further the effect of the layers we choose in the loss definition of the CNN on the resulting image. In particular, it should be noted that in the content loss $\mathcal{L}_{content}$ as defined in 2.4 the authors only consider a specific layer conv4_2 while in the style loss $\mathcal{L}_{style}$ they take into account all the layers (conv1_1, conv2_1, conv3_1, conv4_1, conv5_1) and sum them up. In the article, this choice has been motivated by the fact that the authors found the best visual effect they could get when they consider these layers. On the other hand, they challenged the choice of conv4_2 to match content features by trying to match these features using the layer conv2_2 and highlighted the difference this choice results in. We did the same thing for different layers considering both content image and style image and got the results presented in Fig. 16 and Fig. 17. It must be noted that the following results have been obtained with a gaussian white noise as starting image, considering a weight factor of $\alpha/\beta = 10^{-3}$ and a fixed number of iterations $n = 3000$.

(a) $\alpha/\beta = 10^{-4}$

(b) $\alpha/\beta = 10^{-3}$

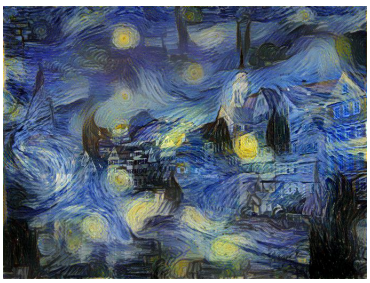
(c) $\alpha/\beta = 10^{-2}$



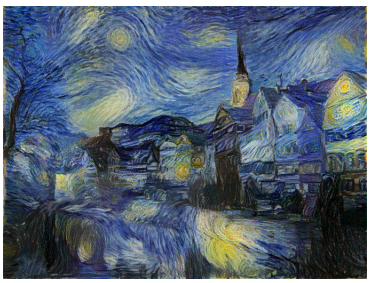
(d) $\alpha/\beta = 10^{-1}$

Figure 15: Influence of weights in image representation (our experiments)

(a) original content image



(b) conv2_2



(c) conv4_2

Figure 16: Content image: Layers Influence

Hereinabove are represented the experiments we conducted to assess the layer influence on the content image. We choose to focus on a *lower* layer conv2_2 and an *upper* layer conv4_2. This choice has been made by the authors as well and seemed justified to us to test the sensitivity of the model. The choice in layers for the style image remained unchanged during all experiments (all layers considered).

The outcome of this is that the choice in content layer has quite a strong influence on the final result. Indeed, fitting the model with conv2_2, a lower layer, preserves better fine details as it can be seen on the roofs and houses on the right of the picture whose shape perfectly fits the original content image. The image as presented on Fig. 16b rather corresponds to a juxtaposition of both the content of Fig. 16a and the original style image whereas Fig. 16c demonstrates a better balance between style and content and hence gives a better visual effect. This is directly linked to the features map as presented in Fig. 11

which strongly differs from *low* layers to *high* layers.

(a) original style image (b) conv2_1 (c) conv4_1

Figure 17: Style image: Layers Influence

Again, the same experiments have been conducted on the style image side. Here, instead of taking into account each and every layers in the loss definition, we decided to focus on a *low* layer conv2_1 and an *upper* layer conv4_1 as we did for the content image. The layer considered for the content image loss remained fixed during all experiments (conv4_2). As expected, we observe that the choice in layer has again an important impact on the resulting image. Indeed, while Fig. 17b still leave stage to the content image, Fig. 17c completely overrides it. Nonetheless, the result presented in Fig. 17b is not perfect as some area remained *blurred* (top right corner). This is due to the starting image (white noise) which is not completely modified while considering only conv2_1 to perform the fitting. Finally, it can be concluded that such a model remains quite sensible to layers choice and one needs to conduct several experiments to assess which choice allows the best visual effect.

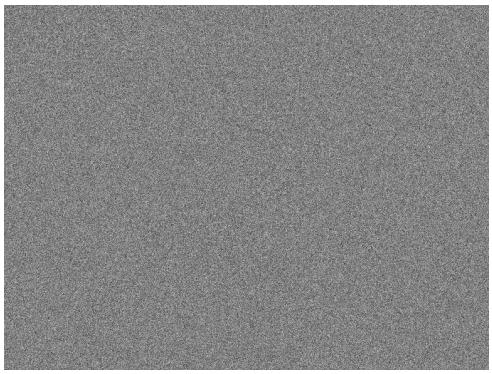
3.3 Initialisation of gradient descent

As discussed before, the initialization seems to be a key element as well, and has been quickly evoked in the article. The presented method takes as starting point a white noise image and then performs model fitting by using gradient descent on it. On their side, the authors challenged this approach by proposing to start either from the content image or the style image and got *similar* results. Hence, they concluded that while the outcomes were not completely similar, they remained in line with the expected visual effect and so the model demonstrates some robustness with regard to the initialization. We decided to conduct the following experiments on our side:

1. Start from gaussian random noise image: Fig. 18a (base case)
2. Start from the content image: Fig. 18c (authors' case)
3. Start from an image with black stripes regularly spaced on a white background 18e (pathological case)

All the experiments have been conducted with weight factor $\alpha/\beta = 10^{-3}$, the loss as described in 2.4 and a fixed number of iterations $n = 3000$. Outcomes are presented in Fig. 18.

In line with the authors' conclusion, we see that starting form a white noise or the content image does not drastically affect the resulting image. Both results are quite

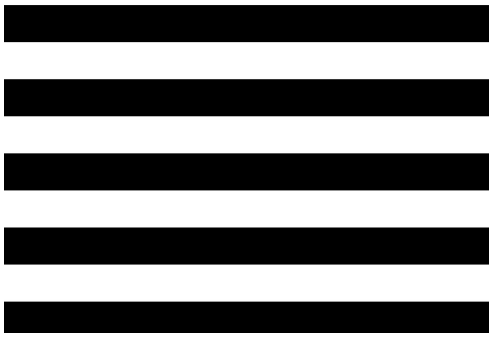


(a) White Gaussian Noise

(b) Outcome

(c) Content Image

(d) Outcome



(e) Black stripes on white background

(f) Outcome

Figure 18: Influence of initialization

visually similar and both mix style and content image. However, intuitively, it can be noted that starting from a white noise will leave room for randomness in the outcome image while always starting from the same image will rather give the same output all the time with some noise coming from the stochastic optimization aspect (if we use SGD approach for instance). Finally, we decided to test the algorithm with what according to us would constitute a pathological case. To do so we considered a hand-made image composed by regularly spaced black stripes on a white background. As we know that CNN does not handle well long distance pixels dependency we found it would be of interest to test the algorithm on this particular image. As expected, the outcome is not satisfactory as we still see the stripes on the resulting image Fig. 18f. We tried to fit the model with even more iterations $n = 10000$ and got the results pictured in Fig. 19b.

(a) Black stripes on white background (b) Outcome

Figure 19: Model fitting $n = 10000$ iterations

3.4 Stochastic outcome

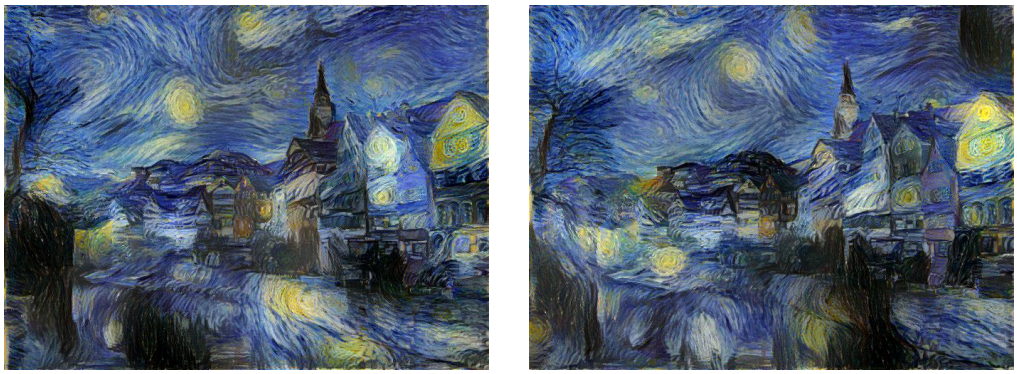
As discussed above, starting from a white noise seems to be an interesting choice in terms of both visual aspect outcome and the randomness of the output this approach allows. To assess the influence of such a random aspect we tried 4 different model fittings starting from 4 randomly generated images and got the results as presented in Fig. 20. While all results output are in line with expectations they remain quite different. This is due to the fact that when we conduct the optimization step we get stuck in local minimums and so do not converge to the same minimizer. Nonetheless, this is a good point if we look at the problem on a playful way as it enables the user to generate a large amount of images that more or less respect the desired visual aspect without being similar.

It is important to note that, even if we have obtained quite different results, they were all satisfactory. As evoked at the end of the introduction, it could have been possible (but it happens with very low probability) to generate a random noise which is an adversarial example, and for which the style transfer would fail [10], [8].

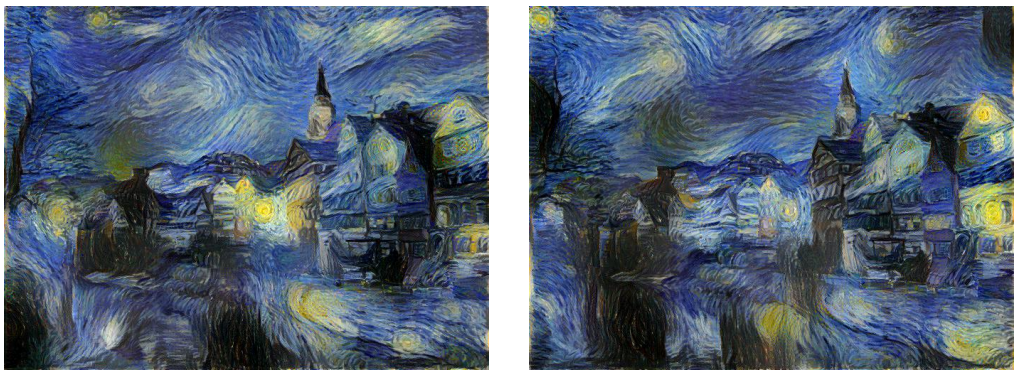
Therefore, since we can consider ourselves as *lucky* on these four examples, and since a video is a sequence of images that are highly correlated, the following questions can be asked: Are we increasing the probability of generating an adversarial example if we do neural style transfer on video, and is this style transfer stable between two consecutive frames? Stable means, roughly speaking, that the style on a certain patch of a frame will be highly similar to the same patch of the next frame.

We have tested these ideas on the free of rights video that can be found at (`https://www.pexels.com/video/video-of-a-city-1906730/`). We used as target image the content image itself, and the same number of iterations and style-content-ratio between frames. The Van Gogh' starry night was our style reference, and we fixed the seed of the optimizer to reduce as much as possible the stochasticity.

Since we are not starting from a random noise, but from the content image, there is no possibility to generate "by chance" an adversarial input. To answer the second part of the question on the stability, we observe a quite stable result, but it can be demonstrated that adding a time dependency gives better results [11].



(a) (b)



(c) (d)

Figure 20: Different outcome using white noise as starting point

3.5 Photorealistic style transfer

Further to the results observed above while using a painting as style image, one may wonder if we obtain the same results with two rather close photographs. This experiment has been conducted in the paper and instinctively we would assume that, as the algorithm seems to handle pretty well paintings, there is no reason that it would not be able to do so for photographs. Indeed, the outcome can be seen in Fig. 21 and presents a satisfactory result even though some blurred areas remain on the final picture coming again from the white noise initialization.

Besides the playful aspect of such an algorithm, we can think of other applications such as image restoration. Indeed, as the proposed method seems to be able to combine efficiently two images, we can try to restore a dark image which will here play the role of the content image using another *similar* image in terms of content but with brighter colors as style image. We tried this on two selected images and a number of iterations $n = 5000$. The result can be seen in Fig. 22 and is quite disappointing at first sight. Nonetheless, it can be seen that the content image has been respected and the outcome is quite close to what we would expect in terms of shape and colors. One must note that it will remain very difficult to completely restore an image with this technique. Indeed, the algorithm will always try to combine both images, meaning that the colors of the style image will most of the time be over-represented in the resulting image.

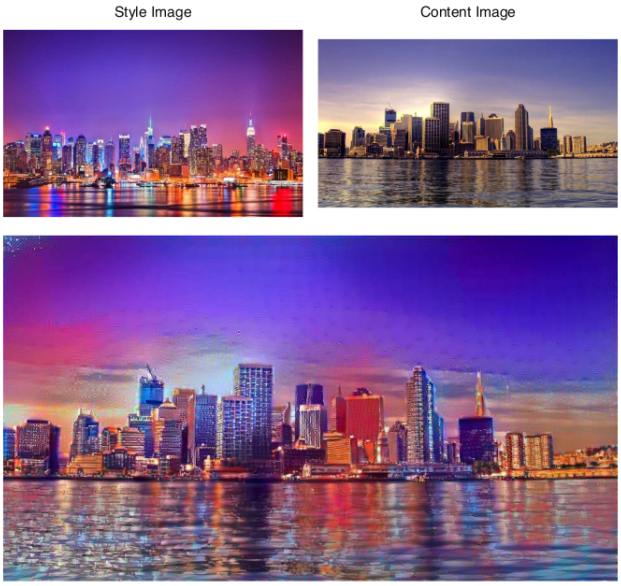


Figure 21: Photo Realistic style transfer

(a) Dark image (b) Reference Image (c) Outcome

Figure 22: Dark image restoration

4 Conclusion

Neural style transfer demonstrates quite impressive visual results when compared to former methods. The method is rather flexible as it remains quite robust to style image and content image change, as well as starting image, as long as it is not a pathological choice. Thus, neural style transfer has also been successfully applied to videos.

However, as demonstrated in the different experiments we conducted, this technique remains quite sensible to the choice of weights as well as the layers considered. In the absence of an *objective* target, the sizing of these parameters has to be performed iteratively and evaluating the visual result.

The method can demonstrate strong results in terms of photograph merging as described in 3.5. Unfortunately, as seen in the very last paragraph of the report, this technique is not as efficient as we would expect to perform other application, as dark image restoration.

References

[1] Beginners guide to convolutional neural networks. https://towardsdatascience.com/beginners-guide-to-understanding-convolutional-neural-networks-ae9ed58bb17d. Accessed: 2019-12-01.

[2] Number of parameters and tensor sizes in a convolutional neural network (cnn). https://www.learnopencv.com/number-of-parameters-and-tensor-sizes-in-convolutional-neural-network/. Accessed: 2019-12-01.

[3] Student notes: Convolutional neural networks (cnn) introduction. https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/. Accessed: 2019-12-01.

[4] What is the vgg neural network? https://www.quora.com/What-is-the-VGG-neural-network. Accessed: 2019-12-01.

[5] Francois Chollet. *Deep Learning with Python*. Manning Publications Co., Greenwich, CT, USA, 1st edition, 2017.

[6] O. Frigo, N. Sabater, J. Delon, and P. Hellier. Split and match: Example-based adaptive patch sampling for unsupervised style transfer. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 553–561, June 2016.

[7] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[8] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. 2014. cite arxiv:1412.6572.

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

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

[11] Manuel Ruder, Alexey Dosovitskiy, and Thomas Brox. Artistic style transfer for videos. *CoRR*, abs/1604.08610, 2016.

[12] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.

[13] David Vanderhaeghe and John Collomosse. Stroke Based Painterly Rendering. In Paul Rosin and John Collomosse, editors, *Image and Video-Based Artistic Stylisation*, volume 42 of *Computational Imaging and Vision*. Springer Berlin / Heidelberg, 2012.

[14] Mingtian Zhao and Song-Chun Zhu. Portrait painting using active templates. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering*, NPAR '11, pages 117–124, New York, NY, USA, 2011. ACM.