Neural Style Transfer as a Data Augmentation Method

Matthieu Bricaire
ENSAE Paris
5 avenue Henry Le Chatelier, 91120 Palaiseau
`matthieu.bricaire@ensae.fr`

Yseult Masson
ENSAE Paris
5 avenue Henry Le Chatelier, 91120 Palaiseau
`yseult.masson@ensae.fr`

Rayan Talate
ENSAE Paris
5 avenue Henry Le Chatelier, 91120 Palaiseau
`rayan.talate@ensae.fr`

Click here to access the GitHub repository associated to this paper.

Introduction

For image classification tasks, deep convolutional neural networks like VGGNet (Simonyan and Zisserman, 2014 [1]) and RESNet (He et al., 2015 [2]) achieve remarkable success. However, the performance of these networks is heavily dependant on the quantity and diversity of the available training data. Limited datasets can hinder the ability of deep neural networks to generalize well to new, unseen instances, leading to overfitting and suboptimal performance. Collecting more labeled samples can be very difficult and/or expensive, as in the medical field for example, where examples of training data are scarse (Sarvamangala and Raghavendra, 2022 [3]). Data augmentation addresses this challenge, by applying a variety of transformations to the existing images, creating new training samples that maintain the inherent characteristics of the original data while introducing variability. Common transformations include rotation, flipping, scaling, cropping, and changes in brightness and contrast.

In this project, we aim to study the possibility of using neural style transfer for data augmentation, as in Zheng et al., 2019 [4]. Neural style transfer was first introduced by Gatys et al. [5], and consists of leveraging the artistic style of one image and applying it to the content of another. In the original method (descriptive approach), a pretrained convolutional network is used to extract features characterizing the style of a reference image (style image), as well as features characterizing the content of the image to transform (content image). These are used to define a style loss and a content loss, that are jointly minimized to obtain an output that incorporates the desired style while preserving the original content. However, this method requires a full gradient descent per pair of content/style images, which makes it not well suited for large-scale augmentation. Johnson et al. [6] extended this method by training a feed-forward generative network to learn a per-style mapping from content images to stylized images. Similarly to Gatys et al., this mapping is learnt by minimizing a loss function that simultaneously enforces content preservation and style transfer. After training such a model for one style, inference on a new content image is very fast, and can reasonably be scaled to a whole dataset.

As proposed in [4], we explore the potential of neural style transfer as a data augmentation technique, in the context of an image classification task performed on the Caltech101 dataset [7]. After comparing the descriptive and the generative approaches, we study the influence of style transfer augmentation on the performance of an image classifier.

1 Style transfer : methods

1.1 The descriptive approach

In 2016, Gatys et al. [5] introduced a new method to tackle the issue of style transfer by leveraging the recent advances of Deep Convolutional Neural Networks. Through the use of the VGG Neural Network (Simonyan and Zisserman, 2014 [1]), their method aims to extract the semantic content of an image (content image) as well as the style representation of another one (style image). Those two representations are then used to generate a new image, which matches both the content representation of the content image and the style representation of the style image. In their work, Gatys et al. [5] used a tailored version of the VGG19 network. For this project, we chose to work with a version of the VGG16 network, that was pretrained on ImageNet for an image classification task. This network is composed of 13 convolutional layers, 5 pooling layers and 3 fully connected layers, as shown on Figure 1 :

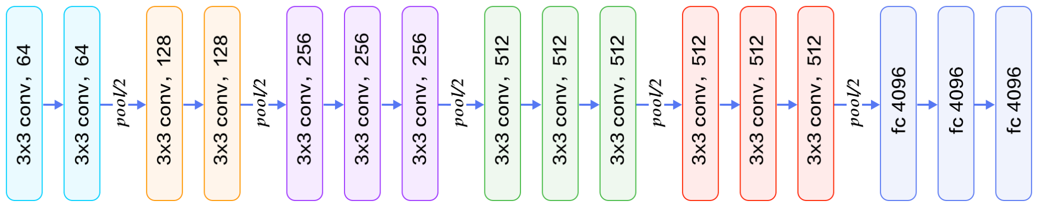


Figure 1: Architecture of the VGG16 neural network [8].

Inside the network, each convolution is followed by a ReLU activation, which reduces any argument to its positive part. As we are only interested in extracting the information encoded by the network's convolutional layers, the last 3 fully connected layers (used for classification) are discarded. To obtain a stylized image, Gatys et al. minimize a weighted sum of two loss functions, called the *feature reconstruction loss* and the *style reconstruction loss*. These two terms are used to measure the differences between the generated image and the content (resp. style) image.

1.1.1 Content representation

On the one hand, Gatys et al. [5] aim to extract the semantic content of a content image y_c, so that it is preserved on the generated image $\hat{y}$. This is achieved by encouraging $\hat{y}$ to have similar feature representations to y_c, inside the VGG16 network (denoted by ϕ). Considering a layer l, its output after processing an input image x is a feature map of shape $C_l \times H_l \times W_l$, where C_l is the number of channels, and H_l (resp. W_l) refers to the height (resp. width) of the feature map. This output can be stored in a matrix $F_x^l \in \mathbb{R}^{C_l \times M_l}$, with $M_l = H_l \times W_l$. With these notations, the feature reconstruction loss between $\hat{y}$ and y_c writes as the squared Frobenius norm between there representations at a chosen layer l_0 :

$$\ell_{feat}^{\phi,l_0}(\hat{y}, y_c) = \frac{1}{2}\sum_{i,j}(F^l_{\hat{y},i,j} - F^l_{y_c,i,j})^2 \tag{1}$$

In practice, we extract the feature representations after the second layer of the fourth block of the VGG16 architecture (referred to as 'conv4_2'). Only one layer is used to compute the feature reconstruction loss, as we do not want to retain too much information regarding the global arrangement of the scene on the content image. Including more exhaustive content information may encourage the generation process to focus more on preserving the original content than transferring the style.

1.1.2 Style representation

On the other hand, Gatys et al. aim to extract the texture and color information of the style image y_s, to transfer it to the generated image $\hat{y}$. This is achieved by encouraging $\hat{y}$ to have similar correlations between filter responses, compared to y_s. The correlations between these filter responses are measured by the Gram matrix. With the same notations as in the previous subsection, the normalized Gram matrix of an input x after a given layer l is defined by : $G_x^l = \frac{1}{C_l M_l} F_x^l (F_x^l)^T \in \mathbb{R}^{C_l \times C_l}$. This matrix encodes the correlations between the filter responses of the layer l to x. It thus captures information about which features of x tend to activate together. Using the previous definitions, the contribution of the layer l to the style reconstruction loss writes :

$$\ell_{style}^{\phi,l}(\hat{y}, y_s) = \frac{1}{4}\sum_{i,j}(G^l_{\hat{y},i,j} - G^l_{y_s,i,j})^2 \tag{2}$$

However, the total style reconstruction loss takes multiple layers into account, at different depths of the network. This increases the amount of style-related information considered during the style transfer process, and thus leads to more appealing and satisfying results. If a set of S_L layers $l_1, ..., l_L$ are included in the style extraction process, the total style reconstruction loss writes :

$$\ell_{style}^{\phi,S_L}(\hat{y}, y_s) = \frac{1}{L}\sum_{p=1}^{L}\ell_{style}^{\phi,l_p}(\hat{y}, y_s) \tag{3}$$

During this project, to compute the style reconstruction loss as Gatys et al. did, we included the first layer of each of the five convolutional blocks in the VGG16 architecture. From now on, these layers will be referred to as ‘conv1_1’, ‘conv2_1’, ‘conv3_1’, ‘conv4_1’ and ‘conv5_1’.

1.1.3 Total loss

Finally, given a content image y_c, a style image y_s, and layers l_0 and S_L to perform feature and style reconstruction, the optimization problem writes :

$$\hat{y} = \arg\min_{y} \alpha\ell_{feat}^{\phi,l_0}(y, y_c) + \beta\ell_{style}^{\phi,S_L}(y, y_s) \tag{4}$$

with α and β two hyperparameters, weighting the feature and style reconstruction losses.

1.2 The generative approach

Johnson et al. [6] propose a refinement of the method designed by Gatys et al.. Indeed, since their descriptive approach requires a full gradient descent to stylize an individual image, it is not adapted to the data augmentation task we would like to conduct. For this task, a much faster method is needed, that can be used to transform a whole dataset in a reasonable time. To make up for this lack of speed, Johnson et al. propose a so called generative architecture, in which a convolutional neural network learns to apply one specific style to any provided image. Once this network is trained, its inference time is orders of magnitude faster than Gatys’ method, and is quick enough to make it an appropriate solution to augment a dataset containing several thousands of images.

The method proposed by Johnson et al. consists in a deep learning architecture, composed of an image transformation network, and a loss network. This architecture can be seen on Figure 2 :

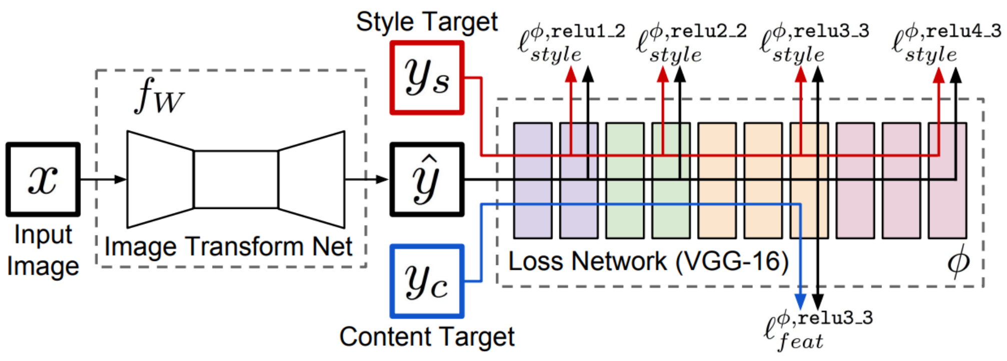

Style Target
$\ell_{style}^{\phi,\text{relu1\_2}}$
$\ell_{style}^{\phi,\text{relu2\_2}}$
$\ell_{style}^{\phi,\text{relu3\_3}}$
$\ell_{style}^{\phi,\text{relu4\_3}}$
y_s
f_W
x
$\hat{y}$
y_c
Input Image
Image Transform Net
Loss Network (VGG-16)
ϕ
Content Target
$\ell_{feat}^{\phi,\text{relu3\_3}}$

Figure 2: Model architecture by Johnson et al. [6]

1.2.1 Image transformation network

First, the image transformation network is a CNN, trained to transform an input image x into a stylized image $\hat{y}$. More precisely, the input image x first passes through 3 convolutional layers (downsampling). It then goes through 5 residual blocks, that enable to overcome optimization difficulties related to the network's depth. Finally, 3 convolutional layers are used, to obtain an output image $\hat{y}$ of the same size as x (upsampling). Except for the last one, all convolutional layers are followed by instance normalization [9] and ReLU activation. The different characteristics of the aforementioned layers can be seen in Table 1 :

Table 1: Architecture details of the image transformation network

Layer	**Input Channels**	**Output Channels**	**Kernel size - Stride - Upsampling**
Conv1	3	32	9x9 kernel - stride 1
Conv2	32	64	3x3 kernel - stride 2
Conv3	64	128	3x3 kernel - stride 2
Res1	128	128	Residual Block
Res2	128	128	Residual Block
Res3	128	128	Residual Block
Res4	128	128	Residual Block
Res5	128	128	Residual Block
Deconv3	128	64	3x3 kernel - stride 1 - upsample 2
Deconv2	64	32	3x3 kernel - stride 1 - upsample 2
Deconv1	32	3	9x9 kernel - stride 1

A point of note is that since the image transformation network is fully convolutional, it can process images of any shape. In practice, we mainly worked with RGB images of size 256x256, to preserve a decent image resolution while maintaining reasonable computation times.

1.2.2 Loss network

The loss network is also a CNN, but this one has been pretrained on another task than style transfer. Similarly to Gatys' method, we worked with VGG-16, and froze its weights during the training. Indeed, during the training, only the weights of the image transformation network are updated. The loss network is used to measure how well the stylized image $\hat{y}$ combines the style of y_s (style image) and the content of $y_c = x$ (content image). The loss network is thus used to compute the objective function of the style transfer problem, that will be minimized during the training.

Similarly to the work of Gatys et al., the loss function proposed by Johnson et al. is perceptual. It does not quantify the per-pixel differences between two images, but rather measures the differences between some of their high-level features, extracted from the pretrained VGG-16 network. Indeed, it has been shown that deep convolutional networks are able to encode texture (style) and content information in their feature maps, that can be extracted and reused [10] [11]. In practice, the loss function minimized during the training is a weighted sum of a *feature reconstruction loss*, a *style reconstruction loss* and a *total variation regularization* term.

With the same notations as the ones used in the presentation of Gatys' descriptive approach, Johnson et al. use the following feature reconstruction loss :

$$\ell_{feat}^{\phi,l_0}(\hat{y}, y_c) = \frac{1}{N_{l_0} M_{l_0}} \sum_{i,j} (F_{\hat{y},i,j}^{l_0} - F_{y_c,i,j}^{l_0})^2 \tag{5}$$

where l_0 refers to the layer at which the content feature maps $F_{\hat{y}}^{l_0}$ (resp. $F_{y_c}^{l_0}$) of the stylized image $\hat{y}$ (resp. the content image y_c) are extracted. Up to a normalizing factor, this feature reconstruction loss is almost identical to the one from Gatys et al.. In practice, we used the output of the 'conv2_2' layer, which corresponds to the setting proposed by Johnson et al..

Regarding the style reconstruction loss, the contribution of a layer l to the style reconstruction loss and the associated total style reconstruction loss write :

$$\ell^{\phi,l}_{style}(\hat{y}, y_s) = \sum_{i,j}(G^l_{\hat{y},i,j} - F^l_{y_c,i,j})^2 \qquad \ell^{\phi,S_L}_{style}(\hat{y}, y_s) = \frac{1}{L}\sum_{p=1}^{L}\ell^{\phi,l_p}_{style}(\hat{y}, y_s) \tag{6}$$

Again, up to a constant, Johnson et al. use the same style reconstruction loss as Gatys et al.. However, to compute this loss, they use the last layer of each of the first 4 VGG16 blocks, namely the 'conv1_2', 'conv2_2', 'conv3_3' and the 'conv4_3' layers.

Finally, to encourage spatial smoothness in the generated image $\hat{y}$, Johnson et al. propose to regularize it using the total variation regularization. $\hat{y}$ being a 3 dimensional tensor, we denote (C, H, W) its number of channels, height and width. With these notations, the total variation regularization of $\hat{y}$ along its height dimension writes :

$$\ell^{H}_{TV}(\hat{y}) = \sum_{c=1}^{C}\left(\sum_{h=1}^{H-1}\sum_{w=1}^{W}|\hat{y}_{c,h+1,w} - \hat{y}_{c,h,w}|\right) \tag{7}$$

The total variation regularization of $\hat{y}$ along its width dimension is defined similarly. With these two terms, the total variation regularization of $\hat{y}$ is defined as :

$$\ell_{TV}(\hat{y}) = \ell^{H}_{TV}(\hat{y}) + \ell^{W}_{TV}(\hat{y}) \tag{8}$$

Finally, given a content image y_c, a style image y_s, and layers l_0 and S_L at which perform feature and style reconstruction, the optimization problem writes :

$$\hat{y} = \arg\min_{y} \alpha\ell^{\phi,l_0}_{feat}(y, y_c) + \beta\ell^{\phi,S_L}_{style}(y, y_s) + \gamma\ell_{TV}(y) \tag{9}$$

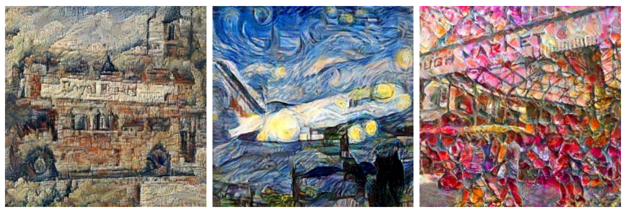
with α, β and γ the weights of the different terms.

2 Style transfer : experiments and results

Here, we present the experiments conducted on both studied style transfer approaches, as well as the results we obtained. To conduct our experiments, we chose three style images presenting very different texture and color characteristics, that can be seen on Figure 7.

2.1 Descriptive approach

Style transfer can be performed with the descriptive approach, by initializing the generated image with a white noise and minimizing the total loss defined in (4). In practice, we applied the aforementioned styles to different content images to stylize (Appendix 8). The obtained results is displayed on Figure 3. On this figure, each image is the result obtained after 1000 gradient descent iterations, as we believed it was a fair number that yielded convincing results (see Appendix (9) to visualize the impact of the number of iterations on the generation). For images (a) et (b), we let $\alpha = 1$ and $\beta = 1000$, resulting in a ratio $\alpha/\beta = 10^{-3}$. However, for the third image, a heavy shift in weight towards the content yielded better results: we let $\alpha = 100000$ and $\beta = 1000$, resulting in a ratio $\alpha/\beta = 10^2$.



(a) Bus
Tapestry style

(b) Plane
Starry night style

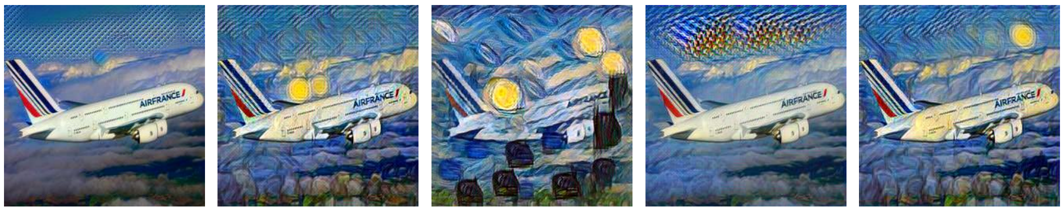
(c) Market
Mosaic style

Figure 3: Style transfer of three different style images to three different content images

The results are visually satisfying, as the generated images nicely balance the desired styles and contents. It appears that depending on the style, a very different content-to-style ratio is required for the output image to be acceptable. Indeed, while a ratio of 10^{-3} is enough to preserve the original content for the tapestry (a) and starry night (b) styles, a much bigger ratio is needed to produce a picture in which the mosaic style does not completely overwrite the content of the content image.

2.2 Generative approach

To study the influence of the model's hyperparameters on the quality of the style transfer, we trained several models on the COCO 2014 training dataset (82783 images). In particular, we explored the variations induced by some modifications of the model's setting on the obtained results (feature reconstruction layer, ratio $\frac{\alpha}{\beta}$, total variation regularization). Each model was trained for 2 epochs with a batch size of 4, and a learning rate of 10^{-3}. The obtained results are shown on figure 4 :



(a) 'conv2_2' $\frac{\alpha}{\beta} = 10^{-4}$ (b) 'conv2_2' $\frac{\alpha}{\beta} = 10^{-5}$ (c) 'conv2_2' $\frac{\alpha}{\beta} = 10^{-6}$ (d) 'conv3_3' $\frac{\alpha}{\beta} = 10^{-5}$ (e) 'conv2_2' $\frac{\alpha}{\beta} = 10^{-5}$ - no reg

Figure 4: Studied model configurations - Starry night style

First, we fixed the layer used to compute the feature reconstruction loss ('conv2_2') as well as the total variation regularization weight ($\gamma = 10^{-7}$), and studied the impact of the content-to-style ratio α/β on the quality of the final style transfer. This ratio proved tricky to balance, as well as very influential on the final results. Indeed, as can be seen on Figure 4, a ratio of 10^{-4} (a) yields a stylized image that looks almost identical to the original one (the content is well preserved, but the style has not been taken into account), which shows that the relative importance given to the content compared to the style was too high. Conversely, a ratio of 10^{-6} (image (c)) gives an image where the style is strongly present, which makes it hard to recognize the original plane. A visually pleasing balance between these two extremes is achieved with a ratio of 10^{-5} (image (b)), for which both the original content and the destination style have been homogeneously combined.

Then, we decided to modify the convolutional layer used to compute the feature reconstruction loss. Namely, we tried to use the 'conv3_3' layer, which is located deeper than the 'conv2_2' layer in the VGG16 architecture. On image (d), it appears that despite using the same setting as (b) except for the feature reconstruction layer, the stylized image is completely different from (b). In particular, it nicely preserves the original content, but doesn't recover well the desired style. Again, this shows that the proper balance between the feature and style reconstruction losses is highly dependent on the model's configuration, and should be tuned appropriately.

Finally, in order to assess the importance of the total variation regularization, we trained a model that had the same hyperparameters as (b), but we removed the regularization term from the total loss function. The obtained result can be seen on image (e). Both (b) and (e) images are visually similar, and nicely incorporate both the original content as well as the desired style. So it is not that clear whether the regularized model performs better than the unregularized one. In general, the model setting from (b) consistently provided good results, so we kept it for our data augmentation tasks.

2.3 Comparison between the two approaches

Since both approaches can yield visually convincing results with the right setting, their main difference lies in their generation speed. To compare the methods, we benchmarked them on a NVIDIA RTX 2070 SUPER GPU, using 256x256 images. We computed the generation time of the descriptive method for different numbers of iterations, and measured the average inference time of the generative method over a test set of 10 images. We then computed the associated speedup factors, and grouped the results in Table 2:

	Gatys et al.			**Johnson et al.**	**Speedup**		
Style \ Iterations	300	500	1000	—	300	500	1000
Starry night	9.37s	15.46s	32.31s	**0.031s**	302x	499x	**1042x**
Tapestry	9.63s	15.09s	33.67s	**0.032s**	301x	472x	**1086x**
Mosaic	9.97s	15.54s	31.70s	**0.031s**	322x	501x	**1023x**

Table 2: Generation speed comparison (in seconds) between both methods

In the above table, it appears that the generative approach is much faster than the descriptive one to stylize images. On the one hand, independently of the used style, it takes on average 0.03s to the generative approach to stylize a 256x256 image. On the other hand, it takes around 30s to the descriptive approach to produce a convincing result (1000 iterations). In [5], Gatys et al. mention that synthetizing 512x512 images took them around 1h on a NVIDIA K40 GPU. While our hardware is probably better than theirs (see here for a comparison) and we work with smaller images, it is hard to tell how much this GPU difference impacted our results, as they do not mention how many iterations they used to synthetize an image. In the end, we find that the generative approach is more than 1000 times faster than the descriptive one, which corresponds to the results obtained by Johnson et al. [6]. However, one has to acknowledge that training a generative model from scratch roughly takes 2 hours on our GPU. Thus, in the context of data augmentation, it becomes more advantageous to work with the generative approach as long as the dataset to augment contains more than 250 images.

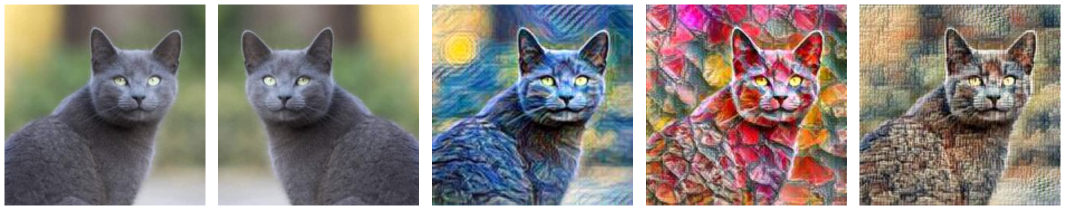
3 Classification

In order to study the effect of style transfer augmentation for an image classification task, we used a VGG16 network on the Caltech101 dataset [7].

3.1 Data augmentation

The Caltech101 dataset consists of 8,677 images belonging to 101 categories. While the number of images in a category can go up to 800, most of them contain about 50 images. When we separate this dataset into train, validation and test sets, there are only about 30 images left in most categories of the training data. As deep classification models require a large number of instances to train, data augmentation becomes useful in this setting. Therefore, we used different kinds of augmentation techniques to create more training instances.

The first type of augmentation we used is the flipping method, which we used as a baseline method for data augmentation. In our case, we chose to flip the images horizontally (Figure 5.b). Regarding the style transfer augmentation (achieved with the generative method), we used three style images with very different textures and styles (Figure 7 in appendix), to maximize the 'data diversity' brought by the style transfer. Figure 5 shows the 4 augmentation techniques used on an original image.



(a) Original image (b) Flipping (c) Starry night (d) Mosaic (e) Tapestry

Figure 5: Types of data augmentation

To assess the usefulness of our different data augmentation methods, we combined them in different ways to augment the train set before training our model: no augmentation (baseline), flip, tapestry, mosaic, starry night, starry night + flip, starry night + tapestry.

3.2 VGG16 fine-tuning

We use a custom VGG16 model to classify the Caltech101 images (Figure 1). As in section 1.1, we discard its last three dense layers, and replace them by the following ones:

- Global average pooling, to transition from feature maps to dense layers by summarizing the presence of a feature in an image [12].
- Dense layer, with 1024 neurons and a ReLU activation function.
- Dropout layer with rate 0.2, to reduce overfitting.
- Dense layer, with 101 neurons (the number of classes) and a softmax activation function, which gives the probability of each class.

Our VGG16 model was pretrained on the ImageNet dataset [13], and we used transfer learning [14] to fine-tune it. More specifically, we started by freezing all the pretrained VGG16 blocks, and training only the top layers mentioned above. During this step, the model trained for a maximum of 20 epochs with a learning rate of 10^{-4}, and stopped if the validation accuracy did not improve for 3 epochs (early stopping with a patience of 3), to reduce overfitting. Then, once the top layers were well adjusted to the VGG16 model, we unfroze its last block (last 3 convolutional layers and max pooling layer) and resumed training for at most 20 epochs, with a patience of 5. In this part, we set the learning rate at 10^{-5}. In this setting, the weight updates were voluntarily made smaller, in order not to lose the information learnt during the pre-training step.

This strategy is different from the one in Zheng et al. [4]. Indeed, they did not remove the fully connected top layers, but rather replaced the last one by a dense layer with the right number of classes. This last fully connected layer is also the only one they train in their experiments. These choices allow them to do many experiments with many augmentation methods, because the training time is small enough. However, the top fully connected layers in the pretrained VGG16 model are very influenced by the labels of the ImageNet dataset on which it was trained, and may not generalize well to another dataset. This is why we decided to remove them completely, and add our own classifier at the top of the last convolutional block. Following the same logic, in the second training, we unfreeze only the top convolutional block (which is the most specific to the ImageNet dataset [14]) and keep the pretrained weights for the rest of the VGG16 model.

3.3 Results of the augmentation experiments

As mentionned above, we stop training the model when the validation accuracy stops increasing. Table 3 shows the number of epochs needed in the first part (training of the custom top layers only) and in the second part (joint training of the top layers and the last VGG16 block) to achieve the best accuracy on the validation set. As we can see, none of the models reached the maximum number of epochs, which was of 20 in each step. Therefore, trying to train those models with more epochs would only have ended up in overfitting.

Table 3: Number of epochs for first and second training

	No augmentation	Flip	Mosaic	Tapestry	SN [a]	SN + flip	SN + tapestry
First	19	17	9	12	14	14	8
Second	9	11	12	11	11	4	2

[a]SN : Starry night

Figure 6 presents the accuracies of the different models on the train, validation and test sets. For lisibility, the y axis goes from 0.9 to 1.

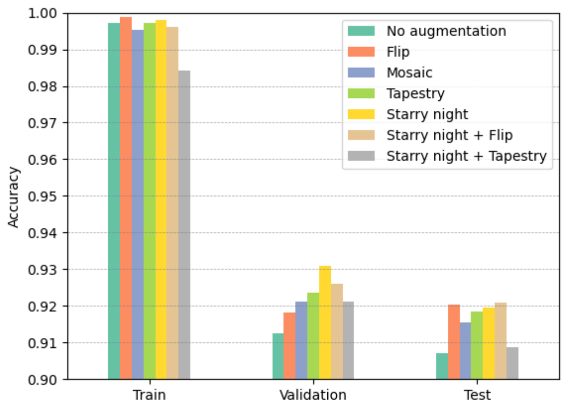


Figure 6: Accuracies for each augmentation strategy

First, we obtain better results than Zheng et al. [4]. Indeed, for VGG16 and Caltech101, they obtain an accuracy of 0.83 with no augmentation and of 0.85 with the best augmentation method. The accuracies of our models on the test set, with or without augmentation, are always higher than 0.9. This may be due to the differences in the classification strategy mentionned above.

In our experiments, models trained on an augmented train set perform better on the test set than the model trained without augmentation. Except "Starry night + Tapestry", all augmentation methods achieve an accuracy at least 1 percentage point higher than without augmentation. This is very encouraging, as style transfer always seems useful as a data augmentation method. However, the models perfoming best on the train set are "Flip" and "Starry night + Flip". In this experiment at least, style transfer augmentation does not perform better than baseline methods like flipping. These results do not corroborate those from Zheng et al., who find an increase in accuracy of 2 percentage points (from 0.83 to 0.85) with style transfer compared to traditional augmentation techniques. This is not surprising, because since the paper from Zheng et al., recent works like [15]) showed that simple augmentation methods (flipping, changing the brightness, etc.), performed at random and with varying intensity degrees, out-perform most sophisticated augmentation methods.

Furthermore, we can notice a hierarchy in the styles regarding the effectiveness of style transfer augmentation. Indeed, for all 3 sets (training, validation and test), "Starry night" yields better results than "Tapestry", which itself has better accuracies than "Mosaic". This may be due to the fact that "Mosaic", and to a lesser extend "Tapestry", add a lot of texture and shapes to the images, which can hinder the ability of the model to classify well. Finally, we notice that "Starry night + Tapestry" always performs worse than both "Starry night" and "Tapestry", and is not much better than "No augmentation" on the test set. This underperformance can also be seen in Table 3 : this model only trained for 10 epochs in total, 8 in the first training and 2 in the second one, which means that the validation accuracy did not improve during the training.

Conclusion

Through this project, we showed that Neural Style Transfer can be considered an adequate method for data augmentation. While the original work by Gatys et al. [5] offers an effective but slow implementation, Johnson et al. [6] proposed a much more efficient and scalable method, which made style transfer suited for large dataset augmentation. Through our application of their method on the Caltech 101 dataset, we found that for every style used to augment the dataset, our VGG16 model achieved better test accuracy than without any augmentation. Although it is unclear whether style transfer performs better than other types of data augmentation, the fact that it never harms the model's performance makes it a decent augmentation method, that could perhaps be implemented in a broader data augmentation framework such as TrivialAugment [15].

References

[1] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *arXiv 1409.1556* (Sept. 2014). DOI: `10.4850/arXiv.409.1556`.

[2] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: 7 (Dec. 2015).

[3] Sarvamangala d r and Raghavendra Kulkarni. “Convolutional neural networks in medical image understanding: a survey”. In: *Evolutionary Intelligence* 15 (Mar. 2022), pp. 1–22. DOI: `https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Gatys_Image_Style_Transfer_CVPR_2016_paper.pdf`.

[4] Xu Zheng et al. “STaDA: Style Transfer as Data Augmentation”. In: Jan. 2019, pp. 107–114. DOI: `10.5220/0007353401070114`.

[5] Leon Gatys, Alexander Ecker, and Matthias Bethge. *Image Style Transfer Using Convolutional Neural Networks*. June 2016. arXiv: `1409.1556 [cs.CV]`.

[6] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. “Perceptual Losses for Real-Time Style Transfer and Super-Resolution”. In: vol. 9906. Oct. 2016, pp. 694–711. ISBN: 978-3-319-46474-9. DOI: `10.1007/978-3-319-46475-6_43`.

[7] Li Fei-Fei, Rob Fergus, and Pietro Perona. “Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories”. In: *Computer Vision and Pattern Recognition Workshop* (2004).

[8] *’VGGNet-16 Architecture: A Complete Guide’*. URL: `https://www.kaggle.com/code/blurredmachine/vggnet-16-architecture-a-complete-guide`.

[9] Andrea Vedaldi Dmitry Ulyanov and Victor Lempitsky. *Instance Normalization: The Missing Ingredient for Fast Stylization*. 2017. arXiv: `1607.08022 [cs.CV]`.

[10] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. *Texture Synthesis Using Convolutional Neural Networks*. 2015. arXiv: `1505.07376 [cs.CV]`.

[11] Aravindh Mahendran and Andrea Vedaldi. *Understanding Deep Image Representations by Inverting Them*. 2014. arXiv: `1412.0035 [cs.CV]`.

[12] Jason Brownlee. *A gentle introduction to pooling layers for Convolutional Neural Networks*. July 2019. URL: `https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/`.

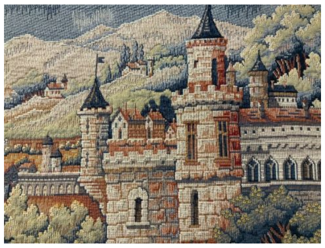
[13] Jia Deng et al. “ImageNet: a Large-Scale Hierarchical Image Database”. In: June 2009, pp. 248–255. DOI: `10.1109/CVPR.2009.5206848`.

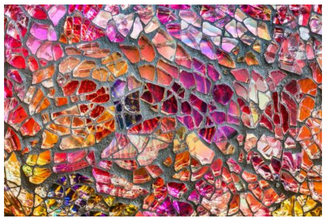
[14] *’Transfer learning and fine-tuning’*. URL: `https://www.tensorflow.org/tutorials/images/transfer_learning`.

[15] Samuel Muller and Frank Hutter. “TrivialAugment: Tuning-free Yet State-of-the-Art Data Augmentation”. In: Oct. 2021, pp. 754–762. DOI: `10.1109/ICCV48922.2021.00081`.

A Appendix

A.1 Styles images


(a) Tapestry


(b) Mosaic

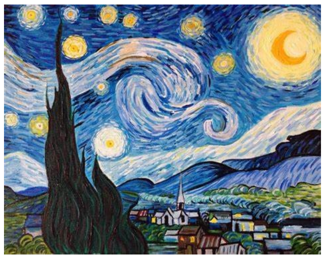
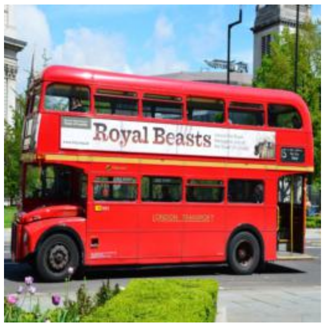
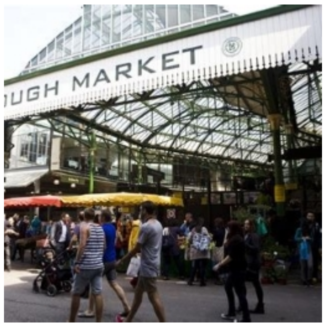

(c) Starry night

Figure 7: Style images

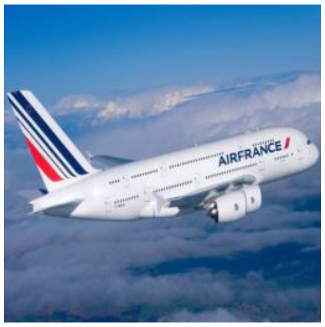
The images can be found here for tapestry, here for mosaic, and here for starry night.



(a) Picture of a bus



(b) Picture of a market

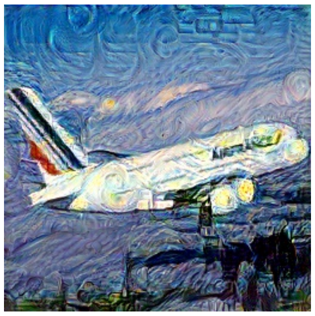


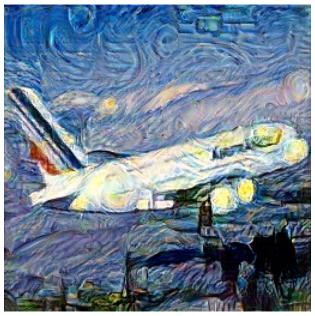
(c) Picture of a plane

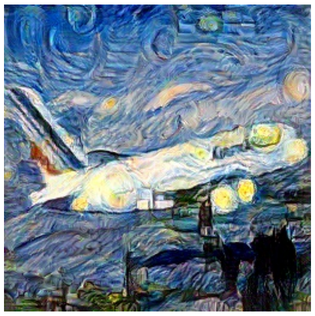
Figure 8: Some base images

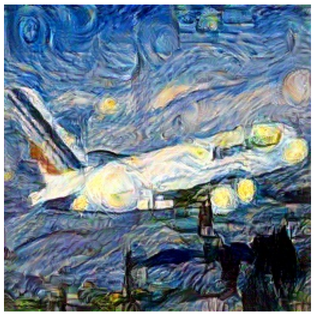
A.2 Impact of the number of iterations on a descriptive generation

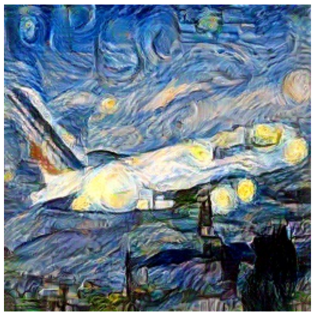
9 shows the number of iterations can have quite an impact on the final generation. Intuitively, since our loss is heavily weighed towards the style loss with a ratio $\frac{\alpha}{\beta} = \frac{1}{1000}$, our model will try to stick more and more to the representation of the style image, since it enables it to further reduce its loss. However, we wanted to have a fixed number of iterations, so there could be a relevant comparison between the generative and descriptive approaches. Since this ratio is used as a default ratio for our work, we motivated our final decision by the results obtained with it. Here, we can see that the images generated by a few number of iterations (images 9a, 9b, 9c) are quite loyal to the original content of the base image. The images generated through many iterations (images 9g, 9h, 9i) replicate the style very well, but the tradeoff is that the content becomes increasingly harder to recognize with respect to the base image. We believe the images of the middle row are a good compromise, and decided to pick 1000 iterations since our main goal was to show the style transfer of an image to another, more than a perfect tradeoff between content and style.

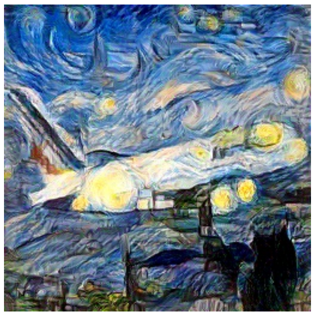

(a) 100 iterations

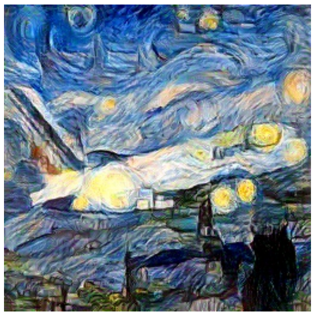

(b) 200 iterations

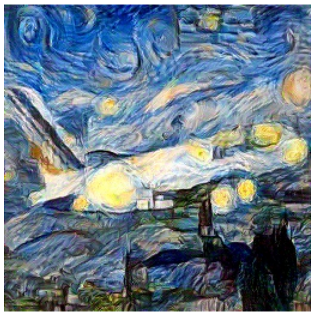

(c) 400 iterations


(d) 600 iterations


(e) 800 iterations


(f) 1000 iterations, as in 3b


(g) 1200 iterations


(h) 1400 iterations

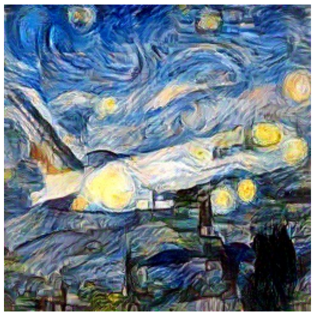

(i) 1600 iterations

Figure 9: transfer of a plane image into Van Gogh's starry night painting style, for different numbers of iterations. For every image, $\alpha = 1$ and $\beta = 1000$