Style Transfer Literature Review

Marie Generali[1], Ángel David Reyero Lobo[1], and Nafissa Benali[1]

Université Paris-Saclay, IMO, 91405, Orsay, France
{nafissa.benali,marie.generali,angel.reyero-lobo}@universite-paris-saclay.fr,
Tuesday 14th March, 2023

Abstract. In this work we will discuss the topic of Film Layer in the form of a conditional instance normalization applied to the Style Transfer Problem. To understand this problem, we carefully studied the literature to understand how the Deep Convolutional Neural Networks were able to separate the style from the content to be able to transfer the style of an artwork to a photograph [1] . To make this neural networks fast enough to be applied in realistic time, [3] introduced the Film Layer that was able to reduce the number of parameters to be trained for each style to 0.2% of the original network. These parameters introduced the notion of style embedding space. Unfortunately, this embedding space was not flexible enough to be generalized to multiple styles. To solve this problem, [4] introduced a Style Prediction Network that computes these parameters fast and provides flexibility and diversification to the style embedding space. It solves the computational problem completely as it is instantaneous to transfer the style to an image, even from styles in which the network was not trained.

Keywords : FiLM Layer; Conditional Instance Normalization; Style Transfer; Texture Synthesis; Deep Convolutional Neural Networks;

1 Introduction

A pastiche (1) is an artistic work that imitates the style of another artist or period, often by combining different elements in a way that creates a new composition. In the context of computer vision and deep learning, a pastiche can refer to an image generated by a style transfer network, which takes as input a content image and a style image and produces an output image that preserves the content of the former while adopting the visual style of the latter. The study of the style of images started with the synthesis of textures. A great

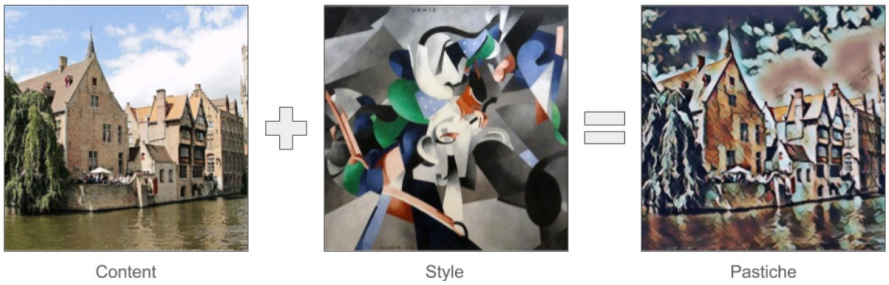


Fig. 1: The pastiche is created from a content image and a style image that are fed to the neural network.

advance was achieved by [1] where they mixed high level and low level image statistics to be able to define a loss to measure both content and style. With this network they are able to create an embedding of the style images that is flexible and generalizable. It allows a much faster training phasis. It is also much more efficient than a single purpose style transfer network. Building on the advances made by [1], researchers have continued to explore new ways of improving style transfer networks. One promising approach involves the use of the Feature-wise Linear Modulation (FiLM) layer, which allows for the selective application of style to different layers of a neural network. In this piece we will study the effect of FILM layers and conditional normalization put to use in style transfer neural networks.

2 FiLM Layer

In this section, we are doing a report of the Feature-wise transformation article on Distill [2].

The article discusses the use of feature-wise transformations as a family of conditioning mechanisms for integrating multiple sources of information in machine learning models. Feature-wise transformations involve modulating or conditioning the computation carried out by a model using information extracted from an

auxiliary input. These transformations act at the feature level and involve feature-wise interactions between the conditioning representation and the conditioned network.

The article starts by presenting an example of building a generative model of images of various classes. The traditional approach is to build a separate model for each class, which is not scalable and misses the opportunity to leverage commonalities between classes. Instead, feature-wise transformations can be used to learn a task representation that allows the model to capture and leverage the relationship between multiple sources of information.

The article explores several ways to integrate conditioning information into the network, including concatenation based conditioning, conditional biasing, conditional scaling, and feature-wise sigmoidal gating.

Concatenation-based conditioning involves concatenating the conditioning representation to the input of all layers in the network.

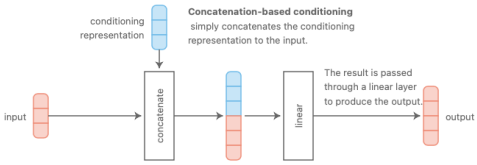


Fig. 2: Concatenation-based conditioning

While concatenation-based conditioning may be more flexible than the previous approach, it still has some drawbacks. First, it may not be the most effective way to use the conditioning information. For instance, some layers in the network may be more sensitive to the conditioning input than others, and it may be more beneficial to apply the conditioning in a way that is tailored to each layer's needs.

Conditional biasing adds a bias to the hidden layers based on the conditioning representation.

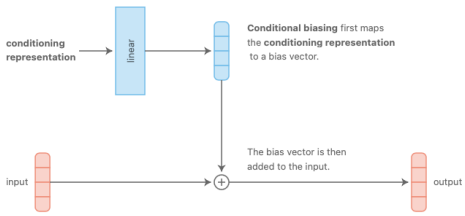


Fig. 3: Conditional biasing : adding a bias to the hidden layers based on the conditioning representation.

Conditional scaling scales hidden layers based on the conditioning representation.

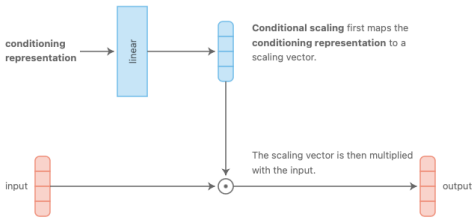


Fig. 4: Conditional scaling : scaling hidden layers based on the conditioning representation.

Feature-wise sigmoidal gating involves scaling each feature by a value between 0 and 1, as a function of the conditioning representation.

The article discusses the advantages and disadvantages of each approach and argues that combining additive and multiplicative interactions into a conditional affine transformation strikes a happy compromise between effectiveness and efficiency. It also highlights the flexibility of feature-wise transformations, which can be used in many neural network architectures to solve a surprisingly large and diverse set of problems.

The benefit of using feature-wise transformations is that they can be compounded across multiple layers, providing enough capacity to model complex phenomena in various settings. However, this technique only

enforces a limited inductive bias and remains domain-agnostic. While this can be a downside for some problems, it also enables these transformations to be effective across problem domains.

To better understand feature-wise transformations, the article adopts the nomenclature of Perez and al. and formalizes conditional affine transformations under the acronym FiLM, or Feature-wise Linear Modulation. The scaling and shifting are applied "feature-wise," which means that they are applied to each element of the input individually. In the case of convolutional networks, this means that the scaling and shifting are applied to each feature map of the input.

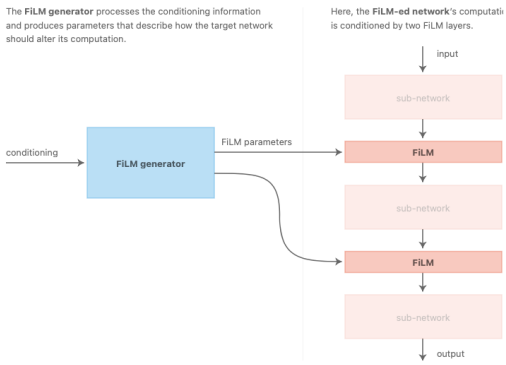


Fig. 5: Modulation induced by fully connected or convolutional FiLM layers

The equation

$$FiLM(x) = \gamma(z) \odot x + \beta(z) \tag{1}$$

shows how the transformation is applied. The input to the FiLM layer is x, and there is also a conditioning input called z. The scaling and shifting vectors, γ and β, are dependent on z. The $\odot$ symbol represents element-wise multiplication. The article suggests that you can interact with fully-connected and convolutional FiLM layers as shown in 5 to get a better understanding of how they work and the kind of modulation they allow.

In summary, the article emphasizes the importance of feature-wise transformations in deep learning and provides a formalized understanding of FiLM layers as a means of incorporating conditioning information into neural network architectures.

Overall, the article provides a useful overview of feature-wise transformations as a family of conditioning mechanisms and their applications in machine learning. It emphasizes the importance of integrating multiple sources of information in solving real-world problems and the role of feature-wise transformations in enabling effective conditioning.

3 Cost function

A huge limitation on style transfer was that it was only based on low-level image features, so it was not possible to separate the content and the texture from the style image only to extract the texture. [1] solved the loss problem by introducing Deep Convolutional Neural Networks to extract high-level semantic information from the images. On the one hand, two images are similar in content if their high-level features as extracted by an image recognition system are close in Euclidean distance. This idea is completely opposed to the previous per-pixel content losses, which are not really adapted to the content because if we shift all the image by 1 pixel, the loss would be huge even though it is the same content. On the other hand, two images are similar in style if their low-level features extracted by an image recognition system share the same spatial statistics.

3.1 Content Representation

For notation, a layer with N_l distinct filters has N_l feature maps, each one of size M_l, where M_l is the height multiplied by the width of the feature map. The outputs of a layer l can be stored in a matrix $F^l \in \mathbb{R}^{N_l \times M_l}$ where F^l_{ij} is the activation of the i-th filter at position j in layer l. Let p be the original image (photograph) and x the image that is generated (the output), and P_l and F_l their respective feature representation in layer l. Then, we can compute the loss between p and x by

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

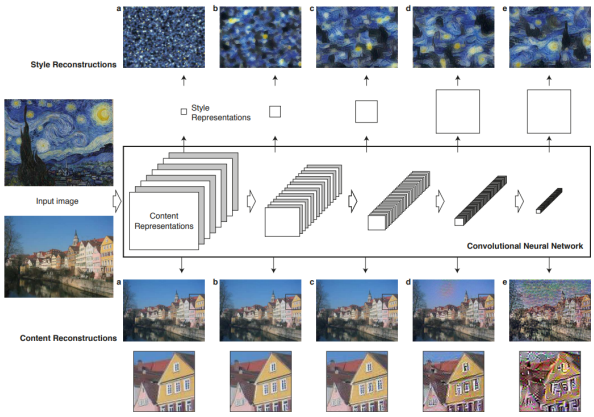


Fig. 6: Image reconstruction through the VGG layers.

We observe how the number of filters is augmented through the layers and how each filter is reduced through the network. On the bottom we observe how we can reconstruct content images from the layers (particularly, 'conv1_2' (a), 'conv2_2' (b), 'conv3_2' (c), 'conv4_2' (d) and 'conv5_2' (e) of the original VGG-Network). We observe that in the lower layers we conserve perfectly the image, and that at the higher layers pixel information is lost, but we can conserve higher-level image content. On the top, we reconstruct the style image. To reconstruct this style we compute correlation between different features and different layers. (Particularly, only 'conv1_1' to construct the (a), 'conv1_1' and 'conv2_1' for the (b), 'conv1_1', 'conv2_1' and 'conv3_1' for the (c), 'conv1_1', 'conv2_1', 'conv3_1' and 'conv4_1' for the (d), 'conv1_1', 'conv2_1', 'conv3_1', 'conv4_1' and 'conv5_1' for the (e).

When we use Deep Convolutional Neural Networks already trained for object classification,we may notice that in the lower layers we are close to the per-pixel representation, as we have shown in the 6 with the lower content reconstructions. As we go along the network, high-level features are conserved and the pixel location is lost, ensuring the content recognition. Hence, we will only use higher layers for the content representation and we will compute this content loss in only one high level, so we may omit the l of the loss function in the rest of the document.

3.2 Style Representation

We will use a feature space to capture the texture of the image. We may assume that if a visual texture is spatially homogeneous, then the lower-order spatial statistics may be represented by a Gram matrix expressing the spatially-averaged correlations between the different filter responses. These Gram matrix may be represented by a square matrix $G^l \in \mathbb{R}^{N_l x N_l}$ (With N_l the number of filters in the layer), where G^l_{ij} is the inner product between the vectorised feature maps i and j in layer l:

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk} \tag{3}$$

By including these correlations we can obtain a multi-scale representation of the input image which only captures the texture as shown in the style reconstructions in 6.

Now, as we have computed the Content Loss, we would like to measure a Style Loss between the original style image a (artwork) with the generated image x. Then, let A_l and G_l the Gram Matrix computed for a and x at layer l. Then, we can define

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

and the total style loss as:

$$\mathcal{L}_{style}(a, x) = \sum_{l=0}^{L} w_l E_l \tag{5}$$

where w_l are weighting factors of the contribution of each layer to the total style loss.

3.3 Total Loss

As our objective is style transfer from a style a to a content image p , we may combine these two losses into a total loss so that our problem can be expressed as the following optimization problem:

$$\mathcal{L}_{total} = \min_x \mathcal{L}_{content}(x, p) + \lambda_s \mathcal{L}_{style}(x, a) \tag{6}$$

with λ_s the weight that we give to the style loss. With this parameter we are able to get the style-content trade-off, as shown in the figure 15. With this total loss we can compute the standard error back-propagation. We can observe this process in the figure 7, where we apply a back-propagation for the Gatys algorithm.

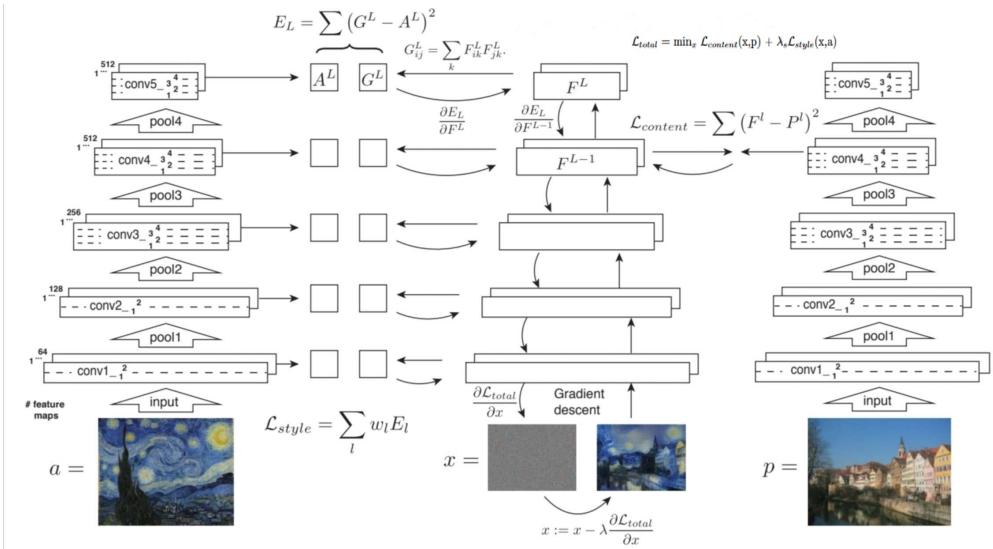


Fig. 7: We begin by extracting the content and style features. To do it, we pass the artwork a (left) and the photograph p (right) through the VGG network and we store the multiple A_ls (left) and one content representation P_l (right). Then, a random white noise x is passed through the network to compute its content features F_l and style features G_l. On each layer included in the style representation, we compute the style loss (left). On the right, we calculate the content loss. The total loss is a linear combination of the content and style loss. Its derivative with respect to the pixel values can be computed using backpropagation (middle). This gradient is used to iteratively update the image x until it matches both the style features of the style image a and the content features of the content image p simultaneously.

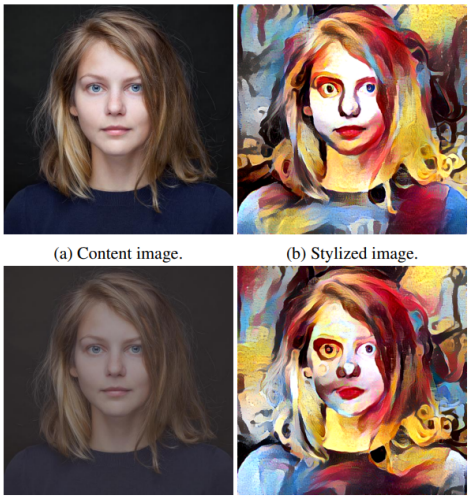
We can also select different layers to represent the content and style, which leads to different visual results. To achieve a smoother and more continuous style, we need to choose high-level layers in the network. That's why we've chosen 'conv1_1', 'conv2_1', 'conv3_1', 'conv4_1', and 'conv5_1' for the style representation. On the other hand, when it comes to the content representation, as shown in Figure 6, matching with higher-level layers of the network makes the content less dependent on detailed pixel information. This can be observed in Figure 8 where we change the content representation from conv4_2 to conv2_2.

Fig. 8: We changed the content loss to be computed with the conv2_2 instead of the conv4_2, and we can observe that we are more pixel constrained.

4 Model

4.1 Conditional Instance Normalization

All convolutional weights of a style transfer network can be shared across many styles, and it's sufficient to adjust the parameters for an affine transformation after normalization for each style. The work by Gatys et al. (2016) [1] used conditional batch normalization, which yielded good results but was computationally inefficient. To solve this problem, [8] proposed conditional instance normalization, which is faster and produces better results. This normalization allows for the removal of instance-specific contrast information from the content image. Without this normalization, there were inconsistencies in the resulting image, such as the contrast of the content image affecting the pastiche, as shown in Figure 9. The contrast of the pastiche should only depend on the style and not the content.



(a) Content image. (b) Stylized image.

(c) Low contrast content image. (d) Stylized low contrast image.

Fig. 9: Trained following the architecture from Gatys. We are able to see how the contrast of the content image alters the style of the pastiche when it should not.

To understand the difference between these two normalizations, given a x_{tijk} denote its $tijk$-th element, where k and j span spatial dimensions, i is the feature channel (color channel if the input is an RGB image), and t is the index of the image in the batch. Then, with batch normalization we will normalize all the batch images using the same mean and standard deviation.

$$y_{tijk} = \frac{x_{tijk} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}, \quad \mu_i = \frac{1}{HWT}\sum_{t=1}^{T}\sum_{l=1}^{W}\sum_{m=1}^{H} x_{tilm}, \quad \sigma_i^2 = \frac{1}{HWT}\sum_{t=1}^{T}\sum_{l=1}^{W}\sum_{m=1}^{H}(x_{tilm} - mu_i)^2. \tag{7}$$

On the other hand, with this instance normalization, we only normalize each image individually.

$$y_{tijk} = \frac{x_{tijk} - \mu_{ti}}{\sqrt{\sigma_{ti}^2 + \epsilon}}, \quad \mu_{ti} = \frac{1}{HW}\sum_{l=1}^{W}\sum_{m=1}^{H} x_{tilm}, \quad \sigma_{ti}^2 = \frac{1}{HW}\sum_{l=1}^{W}\sum_{m=1}^{H}(x_{tilm} - mu_{ti})^2. \tag{8}$$

This instance normalization (or "contrast normalization") replaces the batch normalization everywhere in the generator network g. The results improvement can be observed in 10.
Finally, [3] introduced the parameters γ and β of dimensions $S \times C$ with S the number of styles and C the number of output maps, as we have one row per style. Then, at the output map j we apply $\gamma_{s,j}$ and $\beta_{s,j}$ if we want to get the pastiche with the s-style. Then, at the end of each feature map:

$$z = \gamma_s \left(\frac{x - \mu}{\sigma}\right) + \beta_s \tag{9}$$

4.2 Architecture

This model was mainly developed in three steps. The first step was introduced by [1] (the algorithm is called $ANeuralAlgorithmofArtisticStyle$). We reduce the problem to an optimization problem with a single neural network, as shown in the figure 7. We use a VGG network that was trained in [6] for object recognition

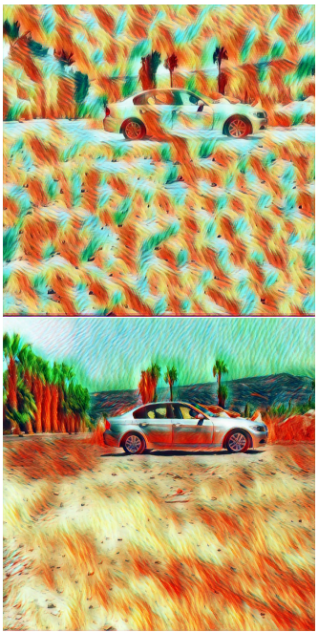


Fig. 10: Trained following the architecture from [7]. For the first image we use a conditional batch normalization, and for the second one a conditional instance normalization.

and localization. We use the feature space provided by a normalized version of the 16 convolutional and 5 pooling layers of the 19-layer VGG network. Therefore, this architecture mainly provides the use of VGG for the loss.

However, this model is not computationally efficient because we do not learn from the common structures of the style images and we train one network for each style and image, making it impossible to create interactive applications for style transfer. To solve this problem, [8] introduced a network that can transfer one style to any image rapidly. However, it was a single-style trained network.

Therefore, [5] introduced a more advanced model to group multiple styles in a single network and to train only 0.2% of the total parameters by training γ and β for each style (3k parameters out of 1.6M for the entire network). This is achieved using a Style Transfer Network that reduces the style into a point in the embedding space. The architecture of this model is shown in Figure 11. The hyperparameters used to train this network

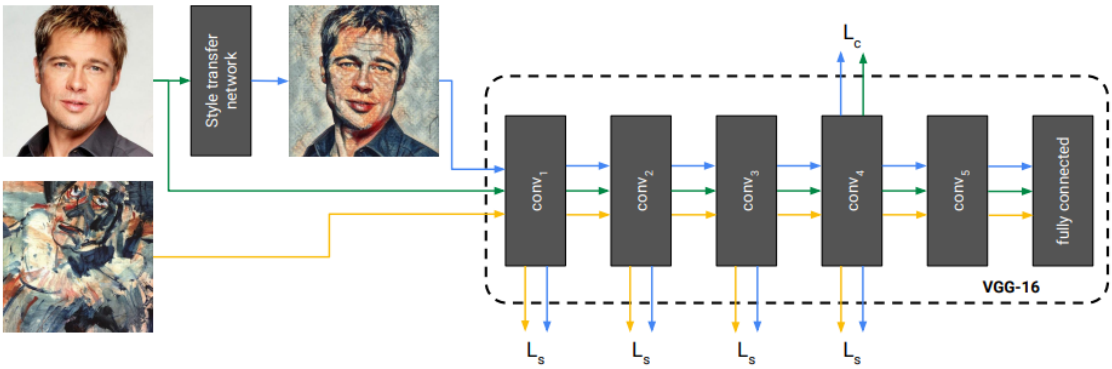


Fig. 11: Style Transfer Network introduced by [3] which is able to do a simple style embedding. It computes the content loss and style loss in the same was as Gatys, explained at 7

are shown in the figure 22.

Finally, in [4], we replaced the simple embedding lookup with a style prediction network. This method was trained on a corpus of 80,000 images and is able to better generalize. The model architecture diagram is shown in Figure 12. The style transfer network largely follows the same structure as [3], and the style prediction network largely follows the Inception-v3 architecture. To obtain the embedding vector, we employ the pre-

trained Inception-v3 architecture and compute the mean across each activation channel of the Mixed-6e layer, which returns a feature vector with the dimension of 768. Then, we apply two fully connected layers on top of it to predict the final embedding. Therefore, this method learns the mapping from the style image to style parameters directly. We train the style prediction network P(·) and style transfer network T(·) on a large corpus of photographs and paintings.

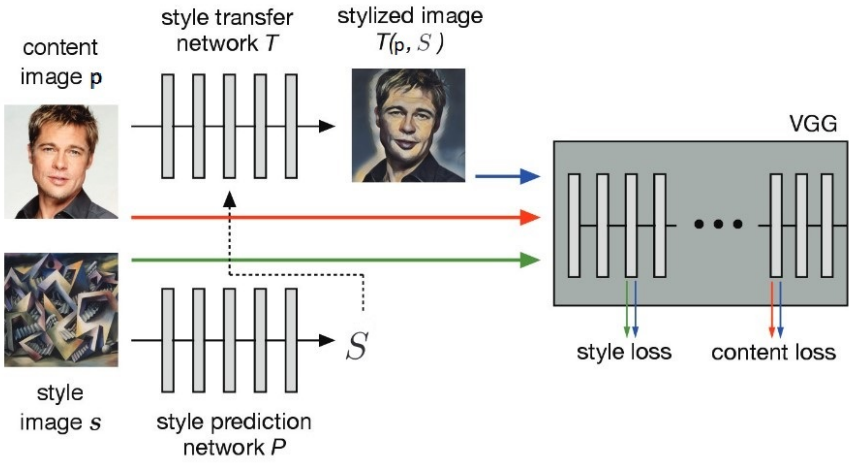


Fig. 12: This model introduces a style prediction network to predict an embedding vector S=$\{\gamma_s, \beta_s\}$ from the input style image s. Then, the style transfer network transforms the photograph into a stylized representation. The content and style losses are derived from the distance in representational space of the VGG image classification network as explained in [1].

This network is able to generalize to new styles. If we observe figure 13 this indicates that the method performs stylizations on observed paintings with nearly equal fidelity as measured by the model objectives for unobserved styles.

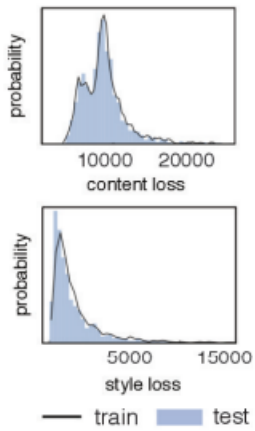


Fig. 13: Distributions of content and style losses in test and train datasets of the Arbitrary Style Transfer trained on the PBN dataset for the style images and the Imagenet dataset for the content images.

Importantly, as demonstrated in the figure 14 that increasing the corpus of trained painting style confers the system the ability to generalize to unobserved painting styles. As detailed in [4], even if the content is not really changed, it takes 16,000 paintings for the style loss to converge.

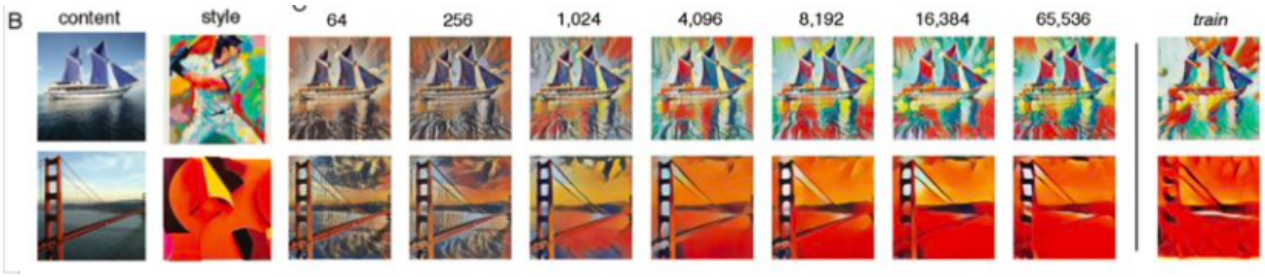


Fig. 14: We can observe the importance if training with multiple images. The number over the images the number of instances used to train the models. When we use more images we are able to preserve more the style.

5 Experiments

The optimization process for style transfer can be represented mathematically using the following objective function: $\min_x \mathcal{L}_{content}$(x,p) + $\lambda_s \mathcal{L}_{style}$(x,a). This equation uses two loss functions, $\mathcal{L}_{content}$(x, p) for the content loss and $\mathcal{L}_{style}$(x, a) for the style loss, to determine the optimal transformation x. The Lagrange multiplier λ_s is used to balance the relative importance of these two losses in the overall objective function.

The first step we took was to test different values of the interpolation weight λ_s. Interpolation is a parameter that allows us to control the degree of style transfer in the image. A weight of 0 corresponds to the identity transformation, while a weight close to 1 results in a completely stylized image. Note that the identity transformation is performed by feeding in the content image as the style image. In this study, we decided to test the λ_s parameter such that $\lambda_s \in \{0, 0.2, 0.4, 0.6, 0.8, 1\}$. This is what we observe in figure 15

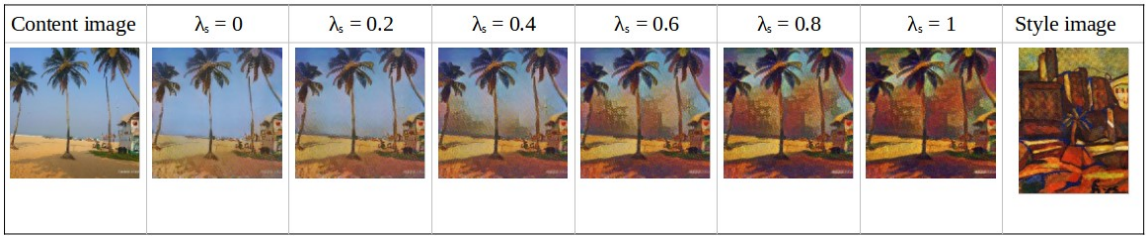


Fig. 15: Representation of the transformation undergone by the content image for different interpolation weights with a single style image.

When working with one style image at a time we also investigated the effect of the gamma and beta parameters on style transfer by neutralizing them. When the beta parameter is set to zero, the shift parameter in the normalization step is disabled, and when the gamma parameter is set to one, the scale parameter is disabled. This can lead to an increase in the contrast of the output image, resulting in a more stylized appearance, but it may also introduce unrealistic colors and other artifacts.
The technique in which γ and β are used, known as conditional instance normalization, normalizes each unit's activation using a linear transformation that specifies the learned mean and standard deviation of the unit for a particular painting style. The concatenation of the learned means and standard deviations constitutes a roughly 3000-dimensional embedding vector representing the artistic style of a painting. Although setting beta or gamma to zero can be useful for achieving specific visual effects in style transfer, it should be used carefully in combination with other techniques to ensure that the output image is of high quality and fidelity.

To perform the style transfer, we used two reference images, each representing a different style. (16) The experiment was conducted on a set of five content images, with an interpolation weight of 0.5. By using two distinct style reference images and varying the content images, we aimed to test the versatility of the style transfer method across a range of scenarios. This is what is illustrated in figure 17 and figure 18

The next topic that we addressed is embedding. Embedding is a powerful data processing technique that has been applied to a variety of machine learning tasks. In the context of image processing, embedding can be used to transform high-dimensional input data into a lower-dimensional space, while preserving important characteristics of the data. One application of embedding is in the field of style transfer, which involves transforming an input image to match the style of a particular painting or artwork.

In this context, building a separate neural network for each painting is inefficient and impractical, since many paintings share common visual textures, color palettes, and semantic structures. Instead, a better ap-

(a) Style image n°1: "Clouds over Bor, 1940"

(b) Style image n°2: "Towers, 1916"

Fig. 16: Style images used for testing β and γ parameters.

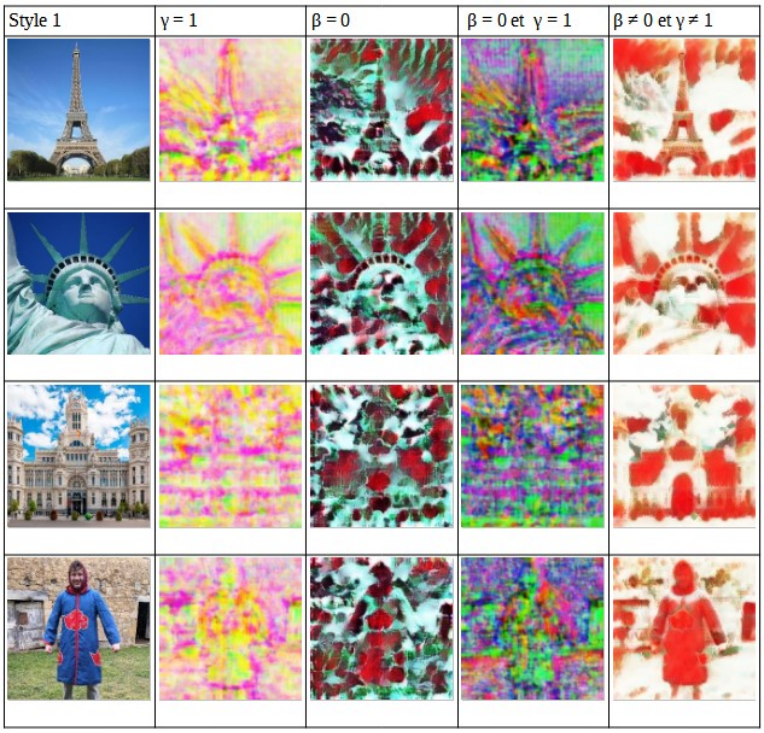


Fig. 17: Representation of different values of β and γ on different content images and with style 1.

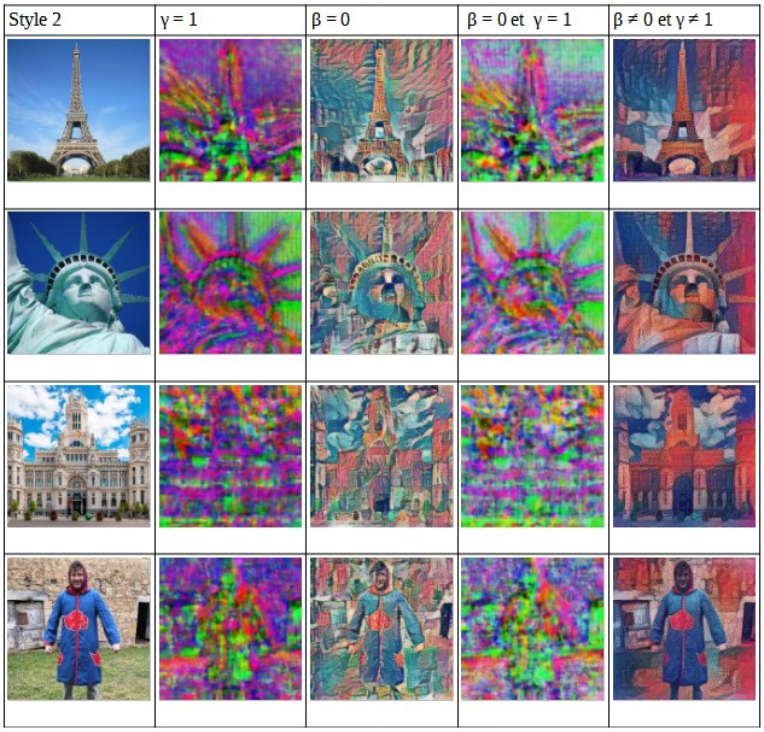


Fig. 18: Representation of different values of β and γ on different content images and with style 2.

proach is to use a single style transfer network that can learn to represent the unique style of any painting in a lower-dimensional embedding space. Figure 19 is a low dimensional representation of the embedding space of the texture and painting datasets that were used during training of the style transfer neural network.

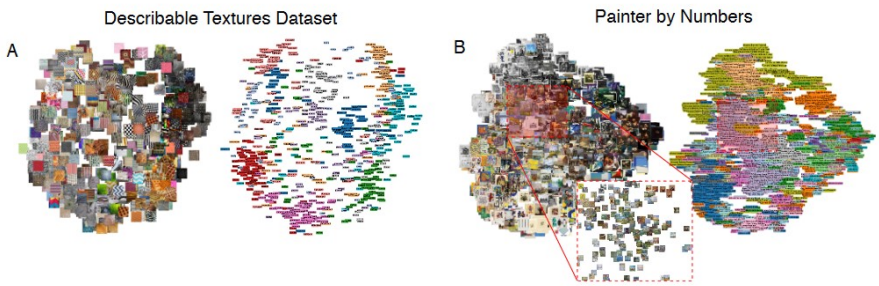


Fig. 19: Low dimensionnal representation of the embedding space of the DTD and PbN datasets.

This embedding space is constructed using conditional instance normalization, which involves normalizing each unit's activation in a way that is specific to the painting style being represented. Each unit's activation z is normalized as follows:

$$z = \gamma_s \left(\frac{x - \mu}{\sigma} \right) + \beta_s \tag{10}$$

Here, μ and σ refer to the mean and standard deviation across the spatial axes in an activation map. γ_s and β_s are the learned mean and standard deviation of the unit.

The resulting embedding vector S = $\{\gamma_s,\beta_s\}$, which contains information about the artistic style of a painting, can be used to transfer the style of any painting onto an input image. The performance of this approach is illustrated by testing the model on style images that it has never seen before. We chose 4 new style images and tested the model with an interpolation weight of 1 and the output can be seen in figure 20.

Another advantage of this kind of embedding is illustrated in [3]. Given a convex combination of $\{\gamma_s,\beta_s\}$, it is possible to generate an image with blended style like in figure 21.

Suppose (γ_1, β_1) and (γ_2, β_2) are the parameters corresponding to two different styles. We use $\gamma = \alpha\gamma_1 + (1 - \alpha)\ \gamma_2$ and $\beta = \alpha\beta_1 + (1 - \alpha)\beta_2$ as our new parameters to stylize an image.

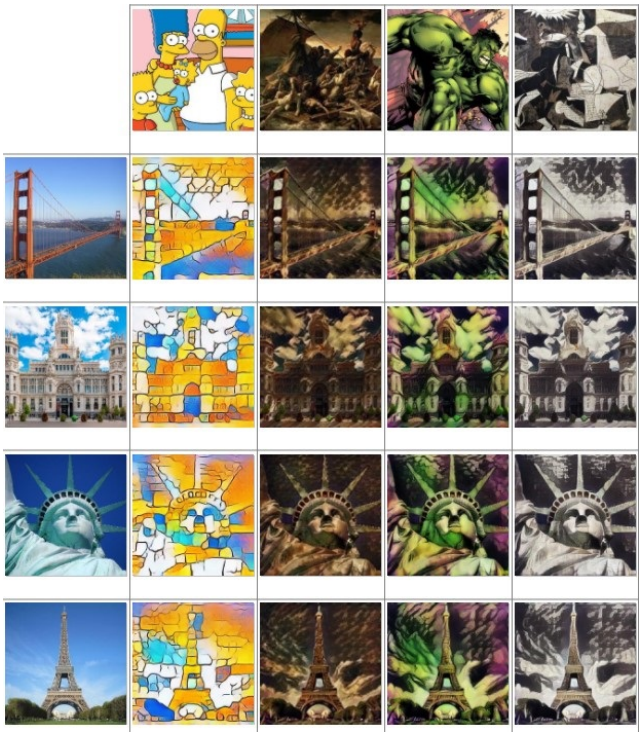


Fig. 20: Representation of how the model generalizes with style images it was not trained on.

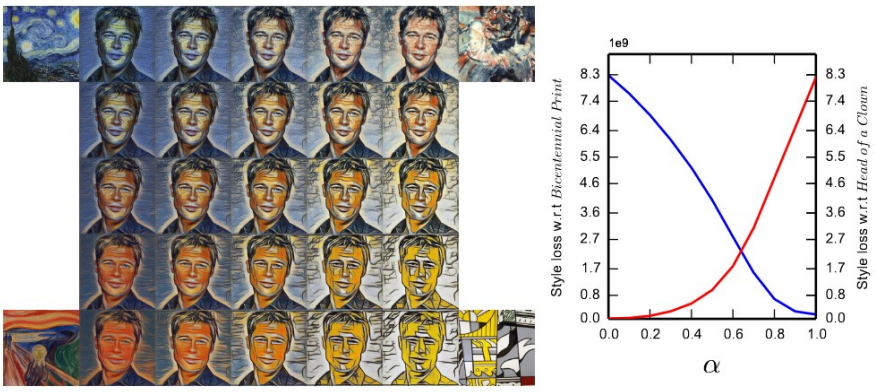


Fig. 21: Left : Pastiches generated combining four different styles (in the corners). Each stylized image corresponds to a different convex combination of the four styles' γ and β values. Right: The style loss as a function of α

6 Conclusion

The purpose of our study is to investigate context normalization in a style transfer neural network model. It is very interesting to see how small parameter such as those which induce conditional normalization can influence the output. Despite the model's excellent performance, there are still some limitations, such as low-level noise that can affect photorealism in photographs. However, this issue could potentially be resolved with post-processing techniques.

Traditionally, artistic stylization of images falls under the category of non-photorealistic rendering in computer graphics. Our approach differs in that we aim to separate image content from style, although this is not a clearly defined problem. Nevertheless, it is fascinating that a neural system trained for object recognition can learn representations that enable partial separation of image content from style.

Separating content and style is challenging since style is often a mixture of various factors such as brush strokes, color maps, dominant forms and shapes, composition, and subject matter. Nevertheless, we consider style transfer successful if the generated image resembles the style image while showing the objects and scenery of the content image.

In conclusion, the similarities between performance-optimized artificial neural networks and biological vision suggest that the ability to abstract content from style may be a fundamental feature of our visual system's powerful inference capabilities.

References

1. Léon A.Gatys, Alexander S.Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. *DOI: 10.1109/CVPR.2016.265*, June 27-30, 2016.
2. Vincent Dumoulin, Ethan Perez, Nathan Schucher, Florian Strub, Harm de Vries, Aaron Courville, and Yoshua Bengio. Feature-wise transformations. *DOI 10.23915/distill.00011*, July 9, 2018.
3. Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A learned representation for artistic style. In *Arxiv:1610.07629*, 2017.
4. Vincent Dumoulin, Jonathon Shlens, Manjunath Kudlur, Honglak Lee, and Golnaz Ghiasi. Exploring the structure of a real-time, arbitrary neural artistic stylization network. In *Arxiv:1705.06830*, August 28, 2017.
5. Johnson Justin, Alahi Alexandre, and Fei-Fei Li. Perceptual losses for real-time style transfer and super-resolution. In *Arxiv:1603.08155*, March 27, 2016.
6. Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Arxiv:1409.1556*, August 28, 2017.
7. Andrea Vedaldi, Victor Lempitsky, and Dmitry Ulyanov. Instance normalization: The missing ingredient for fast stylization. In *Arxiv:1607.08022*, November 6, 2017.
8. Andrea Vedaldi, Victor Lempitsky, Dmitry Ulyanov, and Vadim Lebedev. Texture networks: Feed-forward synthesis of textures and stylized images. In *Arxiv:1603.03417*, March 10, 2016.

7 Appendix

7.1 Appendix I: Hyperparameters for the N-Style Transfer: [3]

Operation	Kernel size	Stride	Feature maps	Padding	Nonlinearity
Network – $256 \times 256 \times 3$ input					
Convolution	9	1	32	SAME	ReLU
Convolution	3	2	64	SAME	ReLU
Convolution	3	2	128	SAME	ReLU
Residual block			128		
Residual block			128		
Residual block			128		
Residual block			128		
Residual block			128		
Upsampling			64		
Upsampling			32		
Convolution	9	1	3	SAME	Sigmoid
Residual block – C feature maps					
Convolution	3	1	C	SAME	ReLU
Convolution	3	1	C	SAME	Linear
	Add the input and the output				
Upsampling – C feature maps					
	Nearest-neighbor interpolation, factor 2				
Convolution	3	1	C	SAME	ReLU
Padding mode	REFLECT				
Normalization	Conditional instance normalization after every convolution				
Optimizer	Adam (Kingma & Ba, 2014) ($\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$)				
Parameter updates	40,000				
Batch size	16				
Weight initialization	Isotropic gaussian ($\mu = 0$, $\sigma = 0.01$)				

Fig. 22: Hyperparameters used to train the Style Transfer Network described in [3]

7.2 Appendix II: Hyperparameters for the Arbitrary Style Transfer: [4]

Operation	spatial dimensions	filter depth
style image	256×256	3
Inception-v3 (mixed 6e)	17×17	768
Reduce mean	1×1	768
Bottleneck (matrix multiply)	1×1	100
Matrix multiply	1×1	2758
Optimizer	Adam ($\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$)	
Parameter updates	4M	
Batch size	8	
Weight initialization	Isotropic gaussian ($\mu = 0$, $\sigma = 0.01$)	

Fig. 23: Hyperparameters used to train the Arbitrary Style Transfer Network described in [4]

Operation	Kernel size	Stride	Feature maps	Padding	Nonlinearity
Network – $256 \times 256 \times 3$ input					
Convolution	9	1	32	SAME	ReLU
Convolution	3	2	64	SAME	ReLU
Convolution	3	2	128	SAME	ReLU
Residual block			128		
Residual block			128		
Residual block			128		
Residual block			128		
Residual block			128		
Upsampling			64		
Upsampling			32		
Convolution	9	1	3	SAME	Sigmoid
Residual block – C feature maps					
Convolution	3	1	C	SAME	ReLU
Convolution	3	1	C	SAME	Linear
	Add the input and the output				
Upsampling – C feature maps					
	Nearest-neighbor interpolation, factor 2				
Convolution	3	1	C	SAME	ReLU
Padding mode	REFLECT				
Normalization	Conditional instance normalization after every convolution				
Optimizer	Adam ($\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$)				
Parameter updates	4M				
Batch size	8				
Weight initialization	Isotropic gaussian ($\mu = 0$, $\sigma = 0.01$)				

Fig. 24: Hyperparameters used to train the Arbitrary Style Transfer Network described in[4]