# Project on images generation: Adaptive Instance Normalization

Hippolyte PILCHEN and Lucas GASCON (MVA ENS Paris-Saclay)

Generative modelling for images project – 03/25/2024

**Abstract**

In this paper, we explore the application of Adaptive Instance Normalization (AdaIN) [1] in the domain of image style transfer, specifically its utility in infusing a content image with the artistic style of a target image, while preserving the original content integrity. AdaIN is proposed as an efficient alternative to overcome the limitations of prior style transfer methods, particularly in terms of computational demand and the scope of applicable styles. We conducted a comparison between AdaIN and existing normalization procedures, such as Batch Normalization, to demonstrate that AdaIN is more suitable for style transfer applications. A comparative analysis with the style transfer framework by Gatys *et al.* [2] is also provided. Through numerous experiments, this study assesses the impact of different model parameters on the results produced by the generative model. Moreover, we explore the benefits of integrating skip-connections into the model architecture specifically designed for AdaIN-based style transfer, with the goal of improving the quality of the images produced by style transfer.

## 1 Introduction

Image style transfer consists in altering an image while preserving its content and aligning it with the artistic style of a designated target image. A substantial leap forward in this domain occurred with the introduction of neural style transfer in the seminal work done by Gatys *et al.* ([3], [2]). As explored during the first class, textures synthesis is a challenging task which has been mostly solved in [3]. Indeed, in their paper Gatys *et al.* were able to generate particular textures by comparing Gram matrices from features extracted at several layers of a pretrained VGG-19 with the texture as input and another one with white noise as input. Similarly, they designed a style transfer method in [2]. The image is updated via an optimization. One shortcoming of this approach is that it requires to solve one slow and memory-consuming optimization problem per content and per style images. Other methods tried to solve this issue by training feed-forward convolutional networks as described in [4]. Even though such processes greatly accelerate the style transfer ($\times 100$ faster than Gatys *et al.*), they are usually limited to one style per network.

In this context, the paper of interest [1] presents a method based on deep neural networks that facilitates both rapid style transfer and the application of this transfer to any arbitrary style. Their approach is built upon the premise that style information can be extracted from the second-order statistics of features (outputs of a layer of a Convolutional Neural Network (CNN)) of a style image, as demonstrated by Li *et al.* [5].

This method relies on feed-forward networks and Instance Normalization (IN) [6], an alternative to batch normalization where samples are normalized individually across channels. Inspired by the conditional instance normalization introduced by Dumoulin *et al.* [7], where IN is utilized to align features with the second-order statistics of a particular style, Huang

and Belongie developed adaptive instance normalization (AdaIN). AdaIN allows for the utilization of the speed of feed-forward methods while not being constrained to a predefined set of styles. As demonstrated through our experiments, this novel method has proven to be effective and capable of generating satisfactory style-transferred images.

We have focused on several aspects to assess the method and explore potential enhancements. Firstly, we examined the trade-off between style transfer and precise content reconstruction by tuning various parameters ($\alpha$ and the style weight). Secondly, we investigated different architectures incorporating skip-connections, as discussed in the perspective of the studied paper (Huang et al., 2017). Lastly, we conducted a comprehensive analysis of AdaIN results compared to Gatys *et al.*'s [2]. All our code is available on GitHub[1].

# 2 Adaptive Instance Normalization

## 2.1 Former normalization methods

In Deep Learning, normalization is ubiquitous. It is a simple calculation which consists in modifying the distribution's statistics of the output of a layer, following this formula:

$$N(x) = \gamma(\frac{x - \mu(x)}{\sigma(x)}) + \beta \tag{1}$$

Batch Normalization, a groundbreaking technique pioneered by Sergey Ioffe and Christian Szegedy in 2015 [8], improved the training of deep neural networks. Before its introduction, training deep models faced hurdles like the internal covariate shift problem, hindering convergence. Batch Normalization addressed this by normalizing activations across mini-batches. It accelerates training by enabling higher learning rates and stabilizes the optimization process. With Batch Normalization, each channel of the output of the layer is normalized according to the **statistics of the mini-batch** (1) and $\gamma$ and $\beta$ are learned parameters. A running mean and standard deviation have been calculated during learning and are used for inference.

| | Mean | Standard deviation |
|---|---|---|
| BatchNorm | $\frac{1}{NHW}\sum_{n,h,w=1}^{N,H,W} x_{nchw}$ | $\sqrt{\frac{1}{NHW}\sum_{n,h,w=1}^{N,H,W}(x_{nchw} - \mu_c(x))^2 + \epsilon}$ |
| IN | $\frac{1}{HW}\sum_{h,w=1}^{H,W} x_{nchw}$ | $\sqrt{\frac{1}{HW}\sum_{h,w=1}^{H,W}(x_{nchw} - \mu_c(x))^2 + \epsilon}$ |

Table 1: Normalization Parameters

Later, Ulyanov *et al.* discovered Instance Normalization (IN) in [9]. It consists in performing a normalization with the **statistics of the sample** directly instead of the mini-batch, enhancing style transfer results. The calculated statistics are summarized in Table 1. This method also avoids discrepancies between training and inference since the formula remains exactly the same during both process in comparison with Batch Normalization. Then, [7] improved this process with Conditional Instance Normalization (CIN). A set of parameters $(\gamma^s, \beta^s)_{s\in\text{Styles}}$ are learned. By inferring and normalizing with the associated pair of parameters $(\gamma^{s^*}, \beta^{s^*})$ the associated style can be transferred to a content image.

Through several experiments, it is demonstrated in our paper of interest [1] that IN leads to a *style normalization* of the image. Therefore, reducing and centering channel-wisely pixels

[1] https://github.com/lucasgascon/arbitrary-style-transfer

of an image in order to apply the statistics of a different style should enable to perform the desired style transfer. This would explain the shortcomings with batch normalization since the statistics of a mini-batch are the statistics of several styles. Batch normalization cannot normalize the style of a single image.

## 2.2 AdaIN for image style transfer

As the name suggests, AdaIN is a variant of instance normalization. In this case, $\gamma$ and $\beta$ are not learned; instead, these variables are set to correspond to the standard deviation and the mean of a style input. Intuitively, the method is akin to CIN, but here parameters are not learned. Therefore, style transfer is not confined to a fixed set of styles.

From our perspective, the promising results achieved by AdaIN are not coincidental. In Conditional Instance Normalization (CIN), a network has the ability to learn an appropriate mean and standard deviation (std) $(\gamma^s, \beta^s)$ for normalizing input to transform an image into a specific style. It appears logical that these two parameters, which determine the mean and std of an input, encapsulate information about the desired style. Therefore, the mean and std of the target style seem to be suitable indicators for conveying style information. Moreover, this process is achieved by adjusting an input's mean and standard deviation.

## 2.3 Architecture and training pipeline of AdaIN

### 2.3.1 Architecture

The style transfer network presented in this paper accepts a content image and a style image as inputs and produces an output image that retains the content of the former while adopting the style of the latter. The model employs an encoder-decoder architecture, with the encoder comprising the initial layers (up to relu4_1) of a pre-trained VGG-19 network. The content and style images are encoded using this encoder, followed by the application of the AdaIN layer introduced earlier. A decoder is then trained to map the output of the AdaIN layer back to the image space, functioning similarly to the encoder but substituting downsampling layers (i.e., pooling layers) with upsampling layers (i.e., nearest upsampling).

In our initial approach, we utilized the pre-trained VGG-19 model from the torchvision library as the encoder. However, we observed improved performance with a variant of the VGG-19 model that was normalized, following the methodology outlined in a GitHub project [10] related to AdaIN. This variant involved a normalization process applied to the pre-trained VGG-19, aiming to standardize the network's weights so that each convolutional filter's average activation equals one, irrespective of the image or position. The key benefit of this normalization is that it makes the magnitudes of losses, calculated from features at different network layers (like the Gramian-based style losses), directly comparable. The process starts by determining the mean activations for each filter from all images, followed by a layer-wise normalization from the lower to the higher layers.

To normalize the activations, a straightforward approach would involve modifying the weights and biases of the $i$-th filter in the $l$-th layer by dividing them by the filter's average activation over all $N$ images of the dataset, setting the average activation to one. This assumes that the activations from the previous layer remain unchanged from the original, unnormalized network. However, this assumption holds only for the first convolutional layer in the normalized network. To overcome this, researchers [11] implement a reverse scaling technique for each subsequent layer, correcting for the previous layer's scaling before applying new adjustments, thereby maintaining consistent normalization throughout the network.

### 2.3.2 Training pipeline

To train our models, we utilized 40,000 content images from MS-COCO and 20,000 style images from WikiArt, inspired by the datasets selected in the referenced paper. We employed the Adam optimizer, with a batch size of 8, and an adaptive learning rate akin to the original implementation. This learning rate starts at $1 \times 10^{-4}$ and decreases by a factor of $1 + 5 \times 10^{-5} \times n_{\text{iter}}$ at each iteration, where $n_{\text{iter}}$ represents the number of iterations since the start of model training.

Only the decoder requires training, for which the following loss function is used:

$$\mathcal{L} = \mathcal{L}_c + \lambda \mathcal{L}_s \tag{2}$$

This represents a weighted combination of a loss on the content images and a loss on the style images. Specifically, the content loss is the Euclidean distance between the target features (the output of AdaIN) and the features, obtained by passing this output through the decoder and then encoded again.

The style loss is

$$L_s = \sum_{i=1}^{L} \|\mu(\phi_i(g(t))) - \mu(\phi_i(s))\|_2 + \sum_{i=1}^{L} \|\sigma(\phi_i(g(t))) - \sigma(\phi_i(s))\|_2 \tag{3}$$

, where $\mu$ is the mean, $\sigma$ the standard deviation, $\phi_i$ denotes a layer in VGG-19 (relu1_1, relu2_1, relu3_1, relu4_1 layers), $g$ the encoder, $t$ the AdaIN output and $s$ the style image.

### 2.3.3 Normalization of input

After finalizing our chosen architecture and overall pipeline, we experimented with two settings. The pre-trained VGG model was trained on the ImageNet dataset, which follows a specific distribution of pixels across each channel (mean $= (0.485, 0.456, 0.406)$, standard deviation $= (0.229, 0.224, 0.225)$). Therefore, we considered that normalizing inputs of our model with the statistics of the dataset used for pre-training could be advantageous. This normalization could enhance the comparability of outputs from layers used for statistical comparison in terms of magnitude.

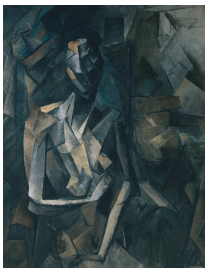
However, as we used weights of a normalized pre-trained VGG, normalizing inputs leads to rather the opposite. Content and style losses are globally 5 to 10 times bigger than the one without normalization of the inputs. The generated image seems darker than in the non pre-processed case (in Figure 7 in the appendix) degrading the content and not respecting the colors of the style image. Furthermore, the training of the version without normalization benefits from a better stability compared to the version with normalized inputs as shown in appendix in Figure 9.

# 3 Our experiments

## 3.1 Compromise between content and style

Here, we outline the experiments we conducted on the $\alpha$ and style weight parameters to assess their impact on the trade-off between content and style they enable. Figure 1 presents the content and style images used in the experiments of the following sections on style weight (Section 3.1.1) and $\alpha$ parameter (Section 3.1.2).

(a) Content image



(b) Style image

Figure 1: Content and style images

### 3.1.1 Style weight

The style weight plays a role in the model's training loss function, as defined in eq. 2, and allows control over the degree of style transfer. The higher it is, the more emphasis is given to style in the calculation of the loss.

(a) sw=1.0 (b) sw=5.0 (c) sw=10.0 (d) sw=15.0 (e) sw=20.0

Figure 2: Images generated with different style weights (sw), and $\alpha$=1

This outcome is indeed observable in Figure 2. With a style weight of 1, the content from the content image is accurately captured, yet only the color from the style image is applied. As the style weight increases, more stylistic features from the style image are transferred to the content image, such as the speckled effect and it becomes increasingly challenging to distinguish the stylistic elements from the content image.

### 3.1.2 Parameter $\alpha$

Similar to the style weight, the $\alpha$ parameter allows a balance between content and style, though it comes into play during the inference phase. $\alpha$ acts as an interpolation parameter between the encoder's feature map and the output of AdaIN. The interpolated result is then fed into the decoder. As $\alpha$ approaches 0, the output increasingly resembles the original content image. Conversely, as $\alpha$ approaches 1, the output more closely aligns with the model's expected outcome, incorporating the content of the content image and the style of the style image.

In the subsequent experiment (Figure 2, where the $\alpha$ parameter was set to 1), the images generated adopted the style of the style image, with varying degrees of intensity. However, the style elements one might expect to be most readily applied, such as color, did appear as such. By adjusting the $\alpha$ parameter, we can significantly diverge from the model-generated stylized image by reducing this parameter, moving closer to the original content image. Thus, the impact of these two parameters is clearly different.

Nevertheless, when $\alpha$ is set to 0, as shown in the appendix in Figure 8, the model's ability to reconstruct the content image is influenced by the style weight parameter during training. Specifically, the higher the style weight, the more emphasis the model places on style in the

generated images to reduce the style loss letting the content loss stop at higher values. To accurately predict the content image becomes increasingly challenging for the decoder.

### 3.2 Architecture with skip-connections for AdaIN

In computer vision, more advanced architecture have been designed in order to perform particular tasks. Since we have worked on a segmentation task for another project, we learned about U-Net [12] architecture for medical images segmentation. As briefly mentioned in [1], AdaIN could be inserted in more complex models, architectures using skip-connections for instance. Therefore, we implemented several models, which are deeply inspired by U-Net, to insert skip-connections in our AdaIN model. They restore spatial information that may have been lost during downsampling. Therefore, this could lead to an even more accurate style or content reconstruction.

Skip-connections have been implemented between the three first encoder layers and three last decoder layers. First, we have chosen to use additive skip-connections instead of concatenation ones to reduce computational complexity and due to its simpler implementation. Three different types of connections have been tested: adding the output of the encoder's layer which take the image content as input, the style content as input, or the sum of both feature maps.

During learning, in Figure 10 in the appendix, adding the style image seems to minimize the style loss compared to adding the content image or both. Moreover, the added feature map during skip-connections does not impact significantly the content loss. The overall loss has values close to the one without any skip-connections. Yet the visual results is disappointing as shown in Figure 3.

Style feature map | Content feature map | Both feature maps

Figure 3: Results of our AdaIN with additive skip-connections trained during 30 epochs with the style weight set to 10 and $\alpha$ to 1. For the rest the setting is the one described in section 2.3.

Unlike what we could expect from the learning losses, the best visual result is obtained by adding the content feature map. In this case, the style-transferred image has a slightly larger contrast but less the fragmented effect. No improvements compared to the model without skip-connections can be observed.

Subsequently, the observed subpar outcomes with additive skip-connections appear to be attributable to the nature of our task. In contrast to a segmentation task, where the objective involves retrieving specific features from both content and style images, our goal here is to isolate only certain features. Introducing additive skip-connections requires the decoder to discern and separate these desired features, which poses a significant challenge.

In light of this, we reasoned that concatenating feature maps for skip-connections could mitigate the complexity imposed on the decoder. By employing concatenation, we provide the decoder with a clearer indication of the features to retrieve, facilitating the preservation

of spatial information lost during downsampling without altering the feature map generated by preceding layers. Throughout the learning process, we observed a similar decrease in losses as with additive skip-connections (as illustrated in Figure 11 in the appendix).

Content concatenated Style concatenated

Figure 4: Results of our AdaIN with concatenation skip-connections trained during 30 epochs with the style weight set to 10 and $\alpha$ to 1. For the rest the setting is the one described in section 2.3.

As depicted in Figure 4, concatenating the feature maps generated from the style image yields a highly stylized output, albeit at the cost of content fidelity. Notably, the content of the original content image is overshadowed by that of the style image, resulting in a transparent appearance of the content image. However, when concatenating the content image, intriguing results emerge. The style is faithfully reproduced, akin to AdaIN without skip-connections. Furthermore, the content appears to be rendered more accurately; crucial features such as the eyes remain unaltered. Additional style-transferred results can be found in Figure 12 in the appendix, where the outcomes demonstrate promise.

### 3.3 Comparison with other methods

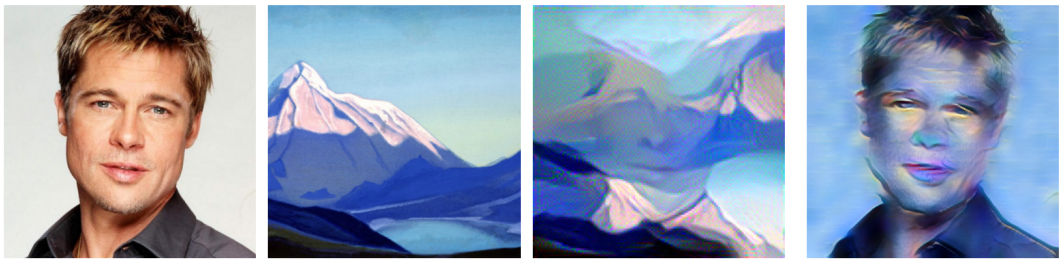
In this section, Gatys *et al.* method [2] and our implementation of AdaIN have been compared based on the quality of the style-transferred image. First, in AdaIN's paper, it is shown that AdaIN is faster than Gatys *et al.* which requires to solve an optimization problem for each style-transferred image. Furthermore, AdaIN is more flexible than other feed-forward methods since it is not limited to a set of styles. Therefore, we focused on the quality of the style transfer for various styles and images. Gatys *et al.* method has been implemented according to PyTorch tutorial with comparison of Gram matrices.

Content image Style image Gatys AdaIN

Figure 5: Comparison of the results of our AdaIN and of Gatys *et al.* method. Gatys algorithm has been run for 1000 iterations. The content image is styled with *Fanfare* (1968) by Miriam Schapiro.

In Figure 5, the image produced by Gatys *et al.*'s model appears to exhibit a style transfer with smoother strokes and colors that blend more seamlessly. In contrast, the image generated by the AdaIN model showcases more vibrant and pronounced colors and textures.

AdaIN adjusts the normalization of content features to match those of the style, potentially leading to a style transfer that more assertively captures the essence of the reference image's style. However, this can sometimes result in outcomes where the content features are less defined or less true to the original image.



Content image | Style image | Gatys | AdaIN

Figure 6: Results of our AdaIN and of Gatys *et al.* method on a content which has been used in both papers but on a new *hard style*. Gatys algorithm has been run for 5000 iterations. The content image is styled with *Tibet* (1933) by Nicholas Roerich.

By exploring the results on the validation dataset, we found that some styles are harder to transfer than others (*hard styles*). This especially occurs for Gatys *et al.* method. Indeed, we highlight in Figure 6 this by using a content image on which both algorithm can transfer certain styles. Then, we illustrate that on a particular style Gatys *et al.* fails while AdaIN keeps giving satisfying results. Failure is evaluated by the fact that the content image is hardly recognizable and that we can see content from the style image in the style-transferred image, which should not occur.

## 4 Conclusion and perspectives

In conclusion, our experimentation with AdaIN has demonstrated its capability to successfully transfer an arbitrary number of styles to various images in a feed-forward manner. The achieved results are as satisfactory as the method proposed by Gatys et al., and in some cases, even surpass them. However, it's essential to meticulously tune parameters such as $\alpha$ and the style weight based on the desired outcome, a process we extensively investigated.

Additionally, we explored more advanced architectures, particularly involving skip-connections, to train and infer from the model. Unfortunately, the results did not meet the expectations suggested by the studied paper. With additive skip-connections, style-transferred images failed to capture the desired style and did not surpass those without skip-connections. However, employing concatenation of feature maps for skip-connections yielded intriguing results comparable to those presented in [1]. Importantly, we implemented this new architecture without increasing the model's parameter count, ensuring no compromise was made on the flexibility of possible styles and the quality of style-transferred images. Moving forward, there is potential for further enhancements to this technique, such as utilizing AdaIN on the concatenated feature maps to refine the style transfer process.

While AdaIN has demonstrated impressive results, it's worth noting that other image style transfer methods have emerged. For instance, in [13], a similar architecture is employed, but Optimal Transport maps and interpolation on the Wasserstein geodesic are utilized for style transfer instead of the adaptive instance normalization block.

Moreover, AdaIN has found applications beyond style transfer. As discussed in class, AdaIN has been leveraged to develop StyleGans [14], a Generative Adversarial Network (GAN) capable of generating realistic images with mixed styles.

## References


[1] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization, 2017.

[2] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423, 2016.

[3] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks, 2015.

[4] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images, 2016.

[5] Yanghao Li, Naiyan Wang, Jiaying Liu, and Xiaodi Hou. Demystifying neural style transfer, 2017.

[6] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization, 2017.

[7] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A learned representation for artistic style, 2017.

[8] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[9] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[10] naoto0804. Github: pytorch-adain, 2017.

[11] corleypc. Github: vgg-normalize, 2019.

[12] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[13] Youssef Mroueh. Wasserstein style transfer, 2019.

[14] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks, 2019.

## Supplementary figures

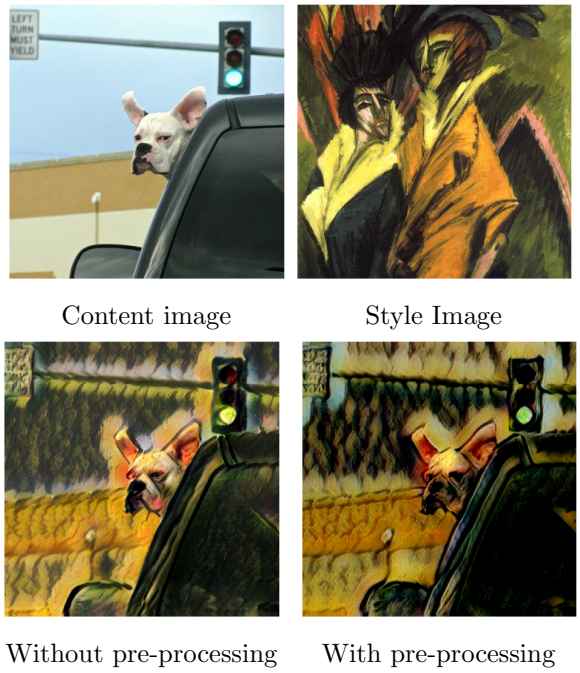



Figure 7: Results of our AdaIN implementation trained with and without pre-normalized content and style images. The decoder has been trained for 30 epochs with the training pipeline described in section 2.3. The content image is styled with *Two Women in the Street* (1914) by Ernst Ludwig Kirchner.

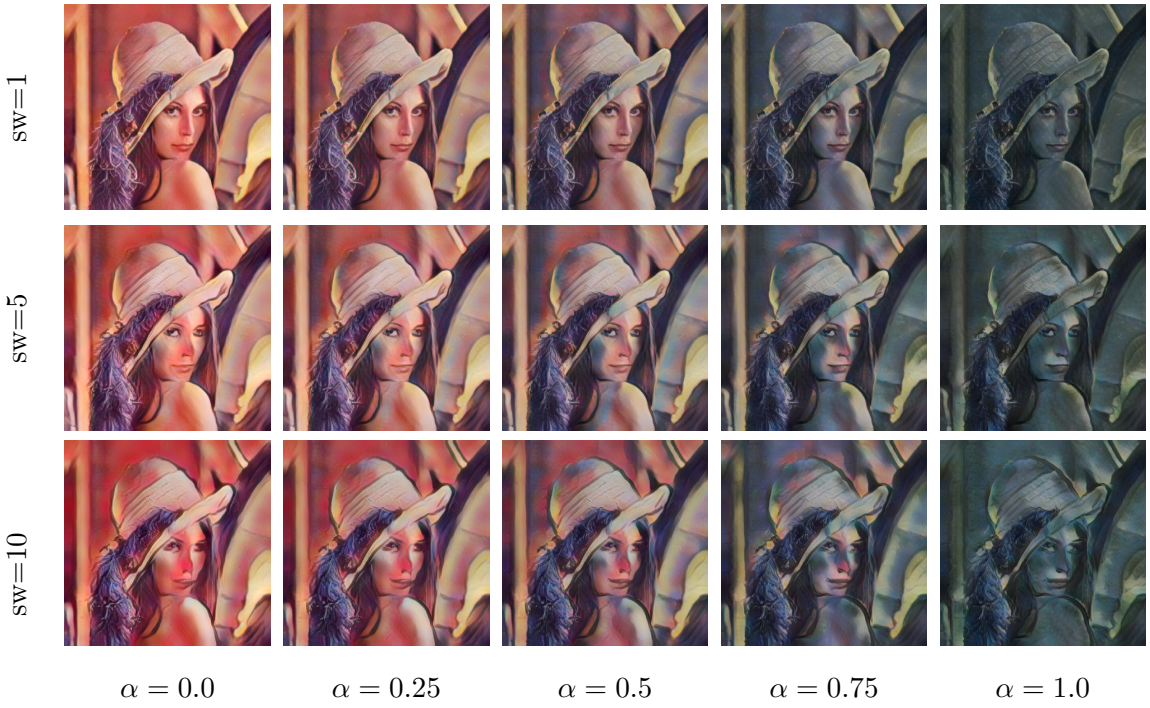



Figure 8: Images generated with different style weight (sw) and $\alpha$ values.

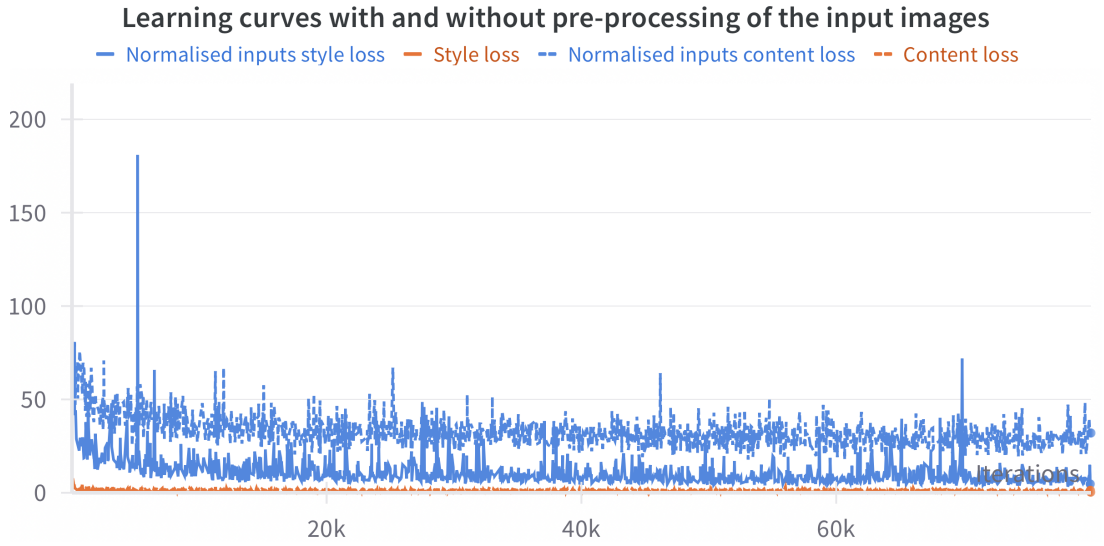



Figure 9: Content and style loss during the learning of our decoder. On the one hand, style and content images have been normalized using the statistics of ImageNet dataset. On the other hand, no pre-processing has been used. $\alpha$ has been set to 1 and the style weight to 10, all the other parameters are the same as described in the training pipeline section 2.3

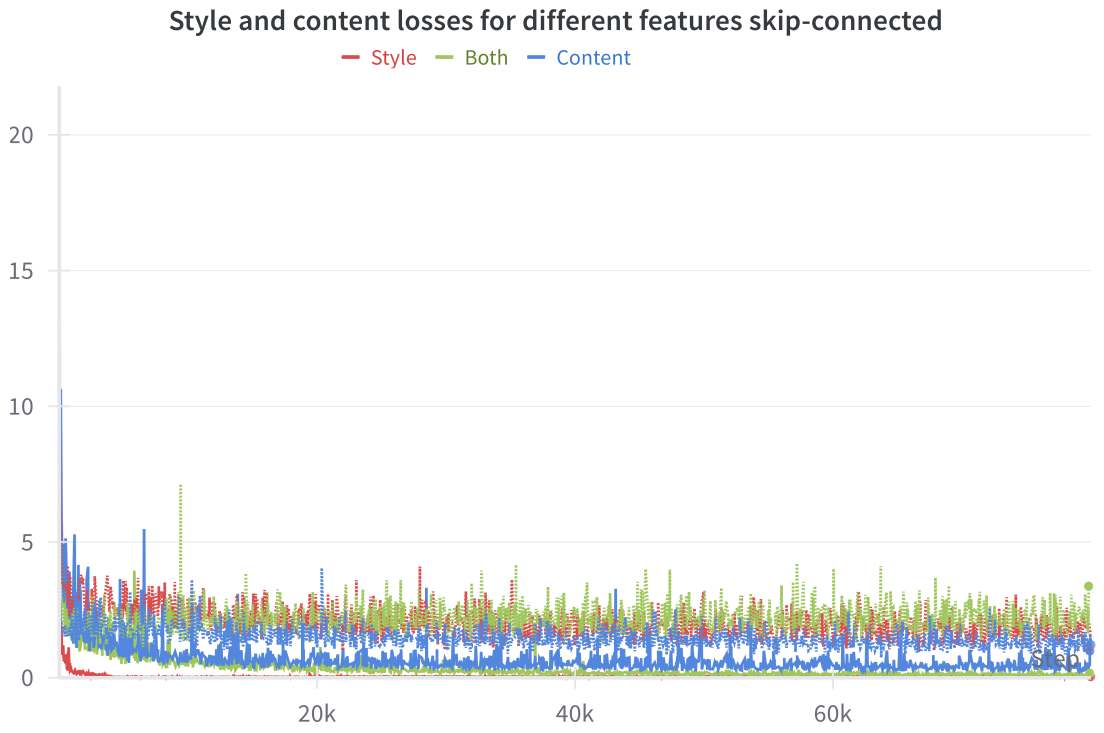



Figure 10: Content (dashed lines) and style (plain lines) loss during the learning of our decoder with additive skip-connections architecure. The skip-connections is done by adding the feature map of the encoder with the style, content or both images as input to the output of the previous decoder. $\alpha$ has been set to 1 and the style weight to 10, all the other parameters are the same as described in the training pipeline section 2.3

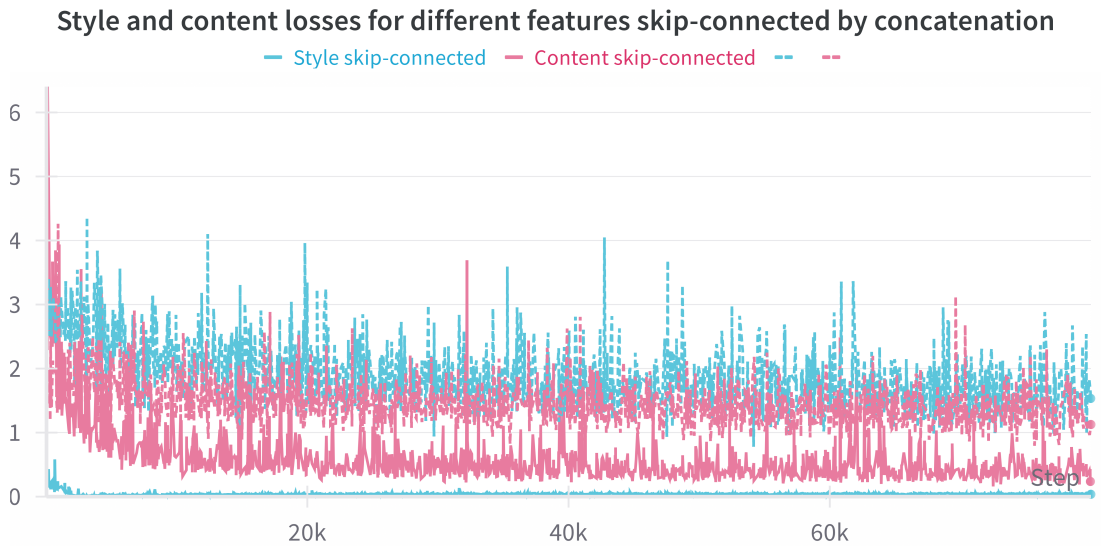



Figure 11: Content (dashed lines) and style (plain lines) loss during the learning of our decoder with additive skip-connections architecure. The skip-connections is done by concatenating the feature map of the encoder with the style, content or both images as input to the output of the previous decoder. $\alpha$ has been set to 1 and the style weight to 10, all the other parameters are the same as described in the training pipeline section 2.3

Figure 12: Results for various style on various image contents using our implementation of skip-connected AdaIN architecture by concatenating content feature maps. The model has been trained for 30 epochs. $\alpha$ has been set to 1 and the style weight to 10, all the other parameters are the same as described in the training pipeline section 2.3

Figure 13: Results for various style on various image contents using our implementation of AdaIN architecture exactly as in [1]. The model has been trained for 30 epochs. $\alpha$ has been set to 1 and the style weight to 10, all the other parameters are the same as described in the training pipeline section 2.3