Image Style Transfer using Deep Learning

Francesco Pio Barone, Daniele Ninni, Paolo Zinesi
Department of Physics and Astronomy "Galileo Galilei", University of Padua, Italy
{francescopio.barone, daniele.ninni, paolo.zinesi}@studenti.unipd.it

***Abstract*—The task of drawing pictures taking inspiration from other sources, e.g. artworks or photos, is rather challenging. For humans, it requires a great effort of creativity and dexterity. In a machine-oriented paradigm, it is complex even to simply formalize the problem. If we restrict to an Image-to-Image formulation, the Style Transfer problem is rephrased in this way: we would like to generate a new picture which represents some given content, but using a style taken from an authorial picture. This branch of research has experienced a great boost in the latter years, thanks to advanced Deep Learning architectures and, in particular, a seminal work of *Gatys et al.* dated 2016.**

In this report, we implement Gatys' original work, which makes use of a VGG-19 Convolutional Neural Network to extract high-level features from a style picture and a content picture. Next, an optimization framework is set up to create a new figure that minimizes a suitable trade-off loss.

While we commend the originality and effectiveness of this algorithm, we recognize its limitations. In particular, it is not suitable for real-time applications. Thus, in the last part of this work we explore a different approach, proposed by *Huang et al.* (2017), which aims to trace out the complexity of an optimization problem for every couple of input pictures. Instead, the computationally expensive task is addressed by training an encoder-decoder architecture.

***Index Terms*—Neural Style Transfer, Deep Learning, Convolutional Neural Networks, Instance Normalization.**

I. Introduction

The process of producing images following the styles of existing artworks is limited by the problem of properly defining what the style of an artwork is. In a broad sense, the style of an image can be defined as the set of correlations between points in the paint or pixels in the image. The ability to extract the style from an image independently from its content allows one to synthesize an image by reverting this process with different content. Although this seems reasonable, the explicit process of disentangling the content and the style of an image still requires a lot of research effort.

A first proposed solution is to directly sample the pixels of an image from those of a given source texture [1]. This method relies on the transferring of low-level features with a non-parametric model, and despite its simplicity it achieves good results. More recent approaches make use of the ability of Convolutional Neural Networks (CNN) to encode image features at increasing level of abstraction along the network depth. In his seminal work, Gatys et al. [2] demonstrate that a CNN that extracts the semantic content from an image can efficiently inform a texture transfer algorithm. A suitable combination of semantic content and texture yields an image where the content is preserved and the style transferred from another image. This work started the research branch of Neural Style Transfer (NST) using the object recognition abilities of CNNs. Researchers have long focused on careful tuning of the network architecture and hyperparameters to improve the original model.

In this report, we present the results of an investigation of the model of Gatys (we refer only to the first author for brevity), starting by developing it from scratch and then focusing on the tuning of its parameters to maximize performances. We improve the first model of Gatys by adding control on the color of the synthesized image and on its resolution, as suggested in [3], while keeping the underline structure intact. We also explore the possibility of obtaining similar results using the more flexible AdaIN-based architecture suggested by Huang & Belongie [4]. Our goal is to find the simplest yet effective solution to synthesize an image with a desired style by comparing different approaches.

In synthesis, we present the results of our work focused on:

- **Implementation of the base model of Gatys**. The base model of the original paper is implemented, and the various hyperparameters are tested.
- **Implementation of color control and high-resolution**. More control of color is added to the synthesized images, and an efficient method to synthesize high-resolution images is presented.
- **Real-time arbitrary style transfer**. The model of Huang & Belongie is implemented to efficiently transfer arbitrary styles without the optimization required by the model of Gatys.

The structure of this report is as follows. In Sec. II we describe in more detail the literature concerning the problem of style transfer. We describe the model of Gatys and its implementation in Sec. III and its results in Sec. IV. The improvements of this model are presented in Sec. V. In Sec. VI the model of Huang is presented and implemented, while the comparison of results are in Sec. VII. We conclude the report with some final remarks in Sec. VIII.

II. Related Work

Early examples of algorithms that perform a processing similar to style transfer are image analogies [5] and image quilting [1]. In the first algorithm, the relation between an input image A and the desired filtered image A' is learned and transferred to an arbitrary image B to obtain the filtered B', using an algorithm that finds the best matching of a pixel in A w.r.t. the pixels in A'. This pattern is then repeated to transform B into B'. In image quilting, texture transfer is constrained by a correspondence map, which is a spatial

Content "Starry Night" "Shipwreck" "The Scream"

Fig. 1: Examples produced with Gatys' optimization. The first row shows the original images from which the styles are derived.

map of some corresponding quantity over both the texture source image and a controlling target image. Interestingly, in this algorithm, there appears a weighted loss between texture matching (similar to a *style* loss) and matching of the correspondence map to the target image (similar to a *content* loss). The results obtained are remarkable, considering that these image processing algorithms are designed ad hoc and no neural network is used to transfer the texture. However, these patch-based texture synthesis algorithms are limited by the small sizes of textures that can be transferred.

The structure of the CNNs greatly improves the performances of style transfer procedures, allowing to go beyond the simple texture transfer algorithms. Most of the researches in the field of style transfer strongly depend on the ability of CNNs to encode features of the content image at increasing levels of abstraction, which can efficiently inform a style transfer by constraining the reproduction of those high-level features. The work of Gatys et al. [2] has sparked interest in the field by demonstrating the feasibility of decoupling content and style from an image. Successive work focused on improving the architecture to speed up image synthesis and improve the quality of the results. Li & Wand [6] propose a combination of generative Markov Random Fields with CNNs to enforce local patterns, but the symmetries of the synthesized images are generally not conserved. Gatys et al. proposed, briefly after their first paper, a possible way to preserve the color of an original image [7]. Then, in another paper, they described how to add more control over color preservation, spatial location, and scale of style transfer [3].

However, the optimization-based algorithm of Gatys is slow and requires some minutes to synthesize an image on GPU. An approach to synthesize images faster is to train a feed-forward neural network to minimize an objective function. Ulyanov et al. [8] show that the speed of synthesis can be increased by two orders of magnitude by dedicating the majority of the computational time to training. This method has been improved in a subsequent article of the authors by introducing Instance Normalization (IN) layers instead of the usual Batch Normalization (BN) layers [9]. However, the method proposed by Ulyanov et al. and developed in many other works [10], [11] is not able to produce an output style not observed during training, and the maximum number of reproducible styles is

fixed by the architecture. Chen & Schmidt [12] introduce a style swap layer that enables the transfer of arbitrary styles in a feed-forward framework. The new layer replaces the content features with the closest-matching style features patch-by-patch. Despite the fact that it allows to transfer arbitrary styles with a feed-forward network, all the improvements of the computation are lost in the swap layer.

Huang & Belongie [4] propose to overcome the issues of both the Gatys optimization-based approach and the feed-forward approaches by introducing the so-called Adaptive Instance Normalization (AdaIN) layer and by using a convolutional encoder-decoder architecture to address the style transfer task.

III. Implementation of Gatys' architecture

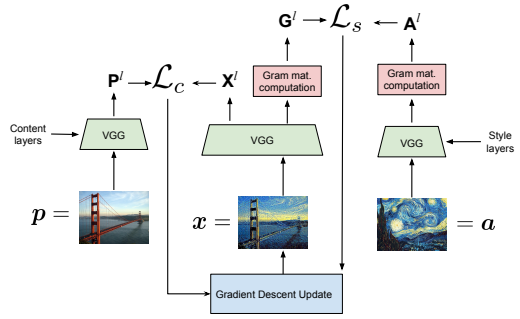


Fig. 2: Overview of Gatys' Style Transfer algorithm.

In this section we review the architecture of Gatys et al. and define the model used to perform the transfer of style from an image to another. The algorithm extracts the features of an image using a pre-trained VGG network used to perform object recognition and visualization [13]. These features are obtained as the layer outputs at a desired depth. Actually, only the convolutional and pooling layers of the original 19-layers VGG model are considered in determining the features of an image. The layers of the network are defined as `convi_j`, where i defines the depth of the layer block and j defines the index of the layer in a given block. A layer block is defined as the ensemble of layers that are inserted between two pooling layers. All convolutional layers terminate with a ReLU activation function with a similar naming convention of the preceding convolutional layer. No Batch Normalization layer is used.

When an image is given as input to the VGG model, the image features are encoded with increasing complexity along the network depth. The features encoded in the first layers mostly reflect the content of small patches of pixels, while along the network structure the recognized features organize into more complex patterns that are assembled starting from the patterns of the previous layer. The set of filter responses to an input image encodes enough information about the original image to allow its reconstruction via an optimization procedure. Considering a layer defined by the multi-index $l = l(i, j)$, the responses of that layer to an image $\boldsymbol{x}$ can be stored in a tensor $\boldsymbol{X}^l \in \mathbb{R}^{N_l \times H_l \times W_l}$, where N_l is the number of feature maps of layer l and (H_l, W_l) are the dimensions of a single feature map at that layer. The tensor elements are denoted as $[x_{ihw}]$ or $[x_{ij}]$ if the dimension indices h, w are compacted into a single index j.

The algorithm of Gatys et al. takes as input the content image $\boldsymbol{p}$ and the style image $\boldsymbol{a}$ and produces an image $\boldsymbol{x}$ by minimizing a loss function. The features of the content image and of the generated image are indicated, respectively, with $\boldsymbol{P}^l$ and $\boldsymbol{X}^l$. Starting from these quantities, the NST framework can be defined as the combination of content and style representations, that separately contribute to the overall loss.

- **Content representation.** Since the filter responses $\boldsymbol{P}^l$ and $\boldsymbol{X}^l$ encode the spatial structure and content of the images, it is possible to enforce the content transfer from $\boldsymbol{p}$ to $\boldsymbol{a}$ by requiring the responses $\boldsymbol{P}^l$ and $\boldsymbol{X}^l$ to be as close as possible. We can thus define a squared distance loss between the two feature representations at layer l,
$$\mathcal{L}_c(\boldsymbol{p}, \boldsymbol{x}, l) = \frac{1}{2} \sum_{ihw} \left(\boldsymbol{P}^l_{ihw} - X^l_{ihw}\right)^2 = \frac{1}{2} \left\|\boldsymbol{P}^l - \boldsymbol{X}^l\right\|^2 . \tag{1}$$
In this way, depending on the depth of the layer l, the minimization of the loss function $\mathcal{L}_c$ allows to constrain the semantic content of an image without fixing the exact value of pixels.
- **Style representation.** The style of an image can be defined as the correlations between different filter responses. A possible metric that encodes the correlation (thus the style) of an image is the matrix of scalar products of the feature representations, named Gram matrix,
$$\boldsymbol{G}^l_{ij} = \sum_{hw} X^l_{ihw} X^l_{jhw} . \tag{2}$$
This matrix $\boldsymbol{G}^l \in \mathbb{R}^{N_l \times N_l}$ has on its diagonal the norms squared of the filter responses and off-diagonal the correlations (up to a multiplicative factor) between different filter responses. Therefore, the style of an image $\boldsymbol{a}$ can be induced in an image $\boldsymbol{x}$ by minimizing the distance between the two Gram matrices, defined respectively as $\boldsymbol{A}^l$ and $\boldsymbol{G}^l$. This leads to a contribution of the layer l to the total loss function of the form
$$\begin{aligned} \mathcal{L}_s(\boldsymbol{a}, \boldsymbol{x}, l) &= \frac{1}{4N_l^2 H_l^2 W_l^2} \sum_{ij} \left(A^l_{ij} - G^l_{ij}\right)^2 = \\ &= \frac{1}{4N_l^2 H_l^2 W_l^2} \left\|\boldsymbol{A}^l - \boldsymbol{G}^l\right\|^2 . \end{aligned} \tag{3}$$

The best synthesized image that merges the content and style of two images is found by optimizing the total loss function
$$\mathcal{L}_{tot}(\boldsymbol{p}, \boldsymbol{a}, \boldsymbol{x}, \lambda) = \alpha\, \mathcal{L}_c(\boldsymbol{p}, \boldsymbol{x}, \lambda) + \beta \sum_l \omega_l \mathcal{L}_s(\boldsymbol{a}, \boldsymbol{x}, l), \tag{4}$$
where ω_l are weighting factors that indicate the importance of each layer's Gram matrix in the optimization w.r.t. the style.

Content | No control | Histogram match | Luminance | Histogram from style

Fig. 3: Effect of different color controls on Gatys' architecture. The style considered is the one of "Starry Night".

We highlight how a single layer λ is used to compute the content loss of the synthesized image. The choice of λ depends on the desired level of content preservation, i.e., a deeper λ constrains less the content of an image than a shallower λ. The parameters α and β respectively tune the importance of the content and the style in the produced image. When a small value of the ratio α/β is selected, the optimization procedure focuses more in minimizing the differences in style than the differences in content. To simplify the treatment, $\beta = 1$ is always fixed so that α assumes the role of relative importance of the content.

IV. Early results

In this section we present some results obtained with the architecture of Gatys et al. (2016) [2]. The discussion of the results obtained with different hyperparameters is also reported here. As can be seen from Fig. 1, the optimization procedure converges successfully to a mixture of style and content in the synthesized image.

A. Pre-processing

The processing of the image starts by resizing the content and style images to a common height of 512 pixels while keeping the original aspect ratios fixed. The resized images are then transformed into PyTorch tensors. To fully exploit the object recognition capabilities of the VGG network, we also shift the channel means of the tensor to the values $[0.485, 0.456, 0.406]$ and change their standard deviations to $[0.229, 0.224, 0.225]$ as suggested in the VGG model documentation in PyTorch.

B. Image synthesis

At the end of image processing, these transformations have to be reversed to obtain a synthesized image with consistent colors. The images in Fig. 1 are obtained in 600 iterations of gradient descent by using as an initialization image the original content image summed with a white noise image. The contribution of white noise enforces some randomness in the optimization process, without being destructive. The optimizer used in this work is L-BFGS, because it converges faster than Adam, and the learning rate used is $\eta = 1$. The relative content weight is $\alpha = 10^{-2}$ and the weighting factors $\omega_l = 1/5$ for each of the layers `conv1_1`, `conv2_1`, `conv3_1`, `conv4_1`, `conv5_1`. The other layers do not contribute to the total style loss. The only layer considered in the content loss is `conv4_2`. This choice of hyperparameters leads to the visually appealing results of Fig. 1. The chosen hyperparameters are related to the ones in the original paper, but some of them are different from the original ones due to the different definition of some parameters. This difference may also be due to the qualitative nature of the "best architecture". It is not, in fact, possible to define in a mathematical sense which synthesized image is closer to the (undefined) ground truth. A detailed discussion about the choice of the hyperparameters and a comparison with other hyperparameters will be performed in a broader scenario in the next section.

V. Improvement of Gatys model

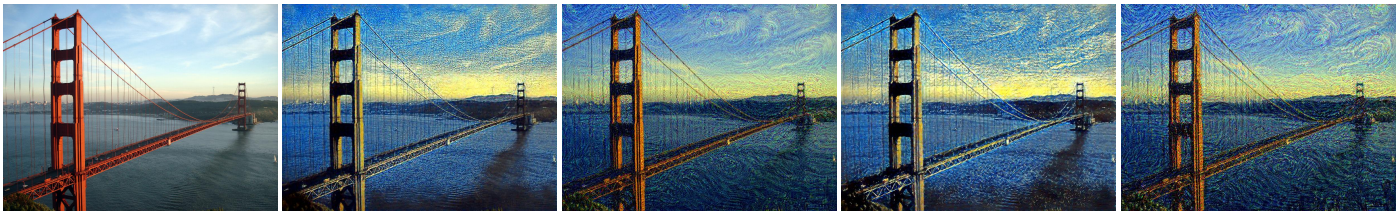
Although the images produced using the method described above are visually appealing, one might be interested in constraining also the colors of the content image while leaving intact the style structure that is transferred. Moreover, since the synthesis of a high-resolution image is more expensive than the synthesis of a lower-resolution one, a more efficient procedure would allow to save time in the computation. To perform these improvements we exploit the methods presented in a subsequent paper of the authors [3].

A. Color control

A simple method to preserve color consists in transforming the colors of the style image to match the colors of the content image, and then performing the style transfer algorithm as before. Only linear methods are used in this work to transform the image, for simplicity. Let $\boldsymbol{\rho}$ be the pixel of an image that encodes the 3 R-G-B values. A linear transformation in the RGB color space transforms this pixel as $\boldsymbol{\rho}' = \boldsymbol{A}\boldsymbol{\rho} + \boldsymbol{b}$, where $\boldsymbol{A}$ is a 3×3 matrix and $\boldsymbol{b}$ is a 3-dimensional vector. The transformation we choose is the one that aligns the mean $\boldsymbol{\mu}_s$ and covariance matrix $\boldsymbol{\Sigma}_s$ of RGB values of the style image to the mean $\boldsymbol{\mu}_c$ and covariance matrix $\boldsymbol{\Sigma}_c$ of the content image.

Content $\eta = 10^{-2}$ $\eta = 10^{-1}$ $\eta = 10^{0}$ $\eta = 10^{1}$

Fig. 4: Effect of different learning rates on Gatys' architecture. The style considered is the one of "Starry Night".



Content `conv1_1` `conv5_1` `conv1_1, conv2_1` `conv4_1, conv5_1`

Fig. 5: Effect of different style layers in the $\mathcal{L}_{tot}$ of Gatys' architecture. The style considered is the one of "Starry Night".

This request results in a set of constraints for $\boldsymbol{A}$ and $\boldsymbol{b}$ in terms of the four RGB statistics,

$$\begin{aligned} \boldsymbol{b} &= \boldsymbol{\mu}_c - \boldsymbol{A}\boldsymbol{\mu}_s, \\ \boldsymbol{A}\boldsymbol{\Sigma}_s\boldsymbol{A}^T &= \boldsymbol{\Sigma}_c. \end{aligned} \tag{5}$$

A family of solutions is obtained using the Cholesky decomposition $\boldsymbol{\Sigma} = \boldsymbol{L}\boldsymbol{L}^T$ for each covariance matrix of style and content, eventually setting

$$\boldsymbol{A}_{chol} = \boldsymbol{L}_c\boldsymbol{L}_s^{-1}\ . \tag{6}$$

Another possibility is to perform an eigenvalue decomposition of the covariance matrix $\boldsymbol{\Sigma} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$, take its square root $\boldsymbol{\Sigma}^{1/2} = \boldsymbol{U}\boldsymbol{\Lambda}^{1/2}\boldsymbol{U}^T$ and set

$$\boldsymbol{A}_{eigv} = \boldsymbol{\Sigma}_c^{1/2}\boldsymbol{\Sigma}_s^{-1/2}\ . \tag{7}$$

In our experience, the eigenvalue variant of linear transform has proved to be effective for the purpose of color control in neural style transfer. We call this color control method "Histogram matching", as it is modulated on the statistics of the image RGB channel values. Later on, we will discuss how this method can be "swapped" to force the content image to be stylized using the colors of the original style artwork, i.e. "Histogram matching from style".

Another approach for color control is to project the input images (both the content and style) on a color space which stores the luminance information as a single component (usually called Y) and chromatic information in the other two components. YIQ is suitable for this purpose, as suggested by the authors [3]. Nevertheless, we used the YCbCr space for simplicity. After the luminance channel is extracted for both the style and the content image, the style transfer algorithm is executed only on the luminance channel. By doing so, the algorithm is able to capture the style information without taking into consideration the color patterns. Eventually, the generated image is color-restored by merging the generated luminance channel (Y) with the color channels from the original image (CbCr). This color control method is named "Luminance".

B. High-resolution

The Gatys' algorithm leads to optimal stylized images (Fig. 1) in a matter of few hundred iterations. The execution of gradient descent takes approximately a minute on modern high-end GPUs, whereas the execution on CPU is much slower, typically in the order of 15/20 minutes with the same parameters. After stating the remarkable advantage of smaller computation time on GPUs, we now wish to briefly discuss the memory requirements.

Gatys' algorithm takes roughly 4GB of video RAM to generate a short-edge 512 pixels image. The memory requirement scales up to 6GB for 720 pixels, to 10GB for 1080 pixels. This rapid scaling in the memory requirement is the main limit for the generation of higher resolution images. An obvious solution is to move to a distributed GPU framework, or demand to a CPU (thus the system RAM) the optimization process. We have noticed that the generation of higher resolution images typically takes more iterations to converge towards optimal results. Also, each iteration itself is noticeably slower.

Even though we cannot address the problem of VRAM requirements, we can adopt a strategy to help the optimization process at higher target resolutions. First, we apply the NST algorithm to generate a low-resolution image, with the framework described in the previous sections, including color control. Instead of returning the generated imaged after n iterations, we upscale the generated image to the (higher) target resolution. Similarly, the content and style image are re-sampled from the source files at the target resolution.

Eventually, the NST algorithm is executed with the higher-resolution images for n^* iterations.

In synthesis, we first run NST on low-resolution images; we execute it again at higher resolution, initializing it using the upscaled generated image from the first step. This strategy reduces the number of overall steps that one should perform to generate the high-resolution image from scratch. The first $n \sim 600$ iterations are executed faster, leading to an optimal low resolution blueprint of the stylized image. The final $n^* \sim 100$ iterations take more time individually, but we need just few of them to make the algorithm converge to the higher resolution image. Therefore, the efficiency of the algorithm increases at higher resolution. A side advantage of this procedure is that low-level noise, which is typical for neural image synthesis, is reduced.

C. Discussion

We now proceed to discuss the effectiveness of the hyperparameters chosen in the implementation of the style transfer, together with the effectiveness of the color transfer approaches described above. Whenever not specified, the hyperparameters used are the same ones described in Sec. IV.

In Fig. 3 different controls on the color of the synthesized image are presented. At first glance, the images obtained without imposing any control on colors reflect the color distribution of the style image (second column). By applying the color transformation from the content image to the style image and then repeating the style transfer procedure, the style is successfully transferred to the content image while preserving the content colors. Depending on the method used to enforce the desired color distribution (either histogram matching or luminance-only transfer), the results show different brightness behaviors (third and fourth column of Fig. 3). The choice of one method or the other needs to be evaluated according to the desired features of the result. The flexibility of the color transfer method allows it to also be applied in the reversed direction, from the style image to the content image. The results obtained in this case are hardly distinguishable from the results obtained without any color control (last column of Fig. 3).

In the following, we discuss particular choices of hyperparameters in the synthesis of images. Figure 4 shows the effects of different learning rates on image synthesis at a fixed number of iterations. Images are produced in 600 iterations using the style of "Starry Night" and histogram matching color control. We immediately notice that images produced with lower learning rates ($\eta = 10^{-3}$) struggle to reach convergence in the fixed number of iterations, whereas higher learning rates ($\eta = 10^{-1}, 10^{0}$) allow a faster convergence. For even higher learning rates ($\eta = 10^{1}$), the style transfer algorithm explores pixel values outside the $[0, 255]$ RGB range, and thus the synthesized images present many black regions where the RGB scale is fully saturated. In fact, the optimization is performed over the (limited) pixels space of the synthetic image and any gradient descent step may move the image outside of this space if the norm of the loss gradient is not properly limited. However, by fixing the values of α, β in the total loss, the optimal learning rate range can also be fixed once for all.

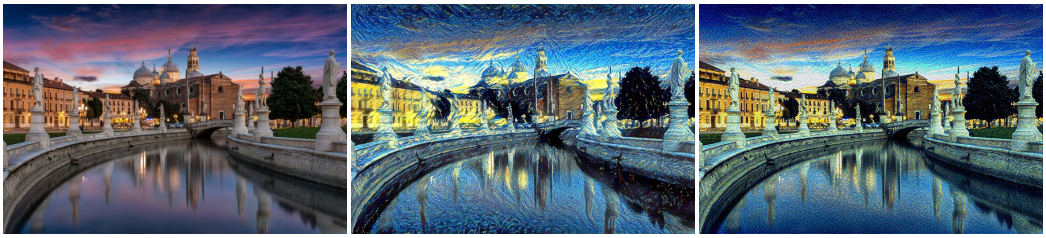
The performance of the style transfer also depends on which and how many Gram matrices are considered in the style part of the total loss function. To produce an image with a style as similar as possible to the desired style, many style layers need to enter the loss evaluation. Figure 5 shows how the number of layers considered in the loss function affects the final results. No color correction is enabled for these images. When only the lower `conv1_1` layer is used to define the style, the content image is equipped with only the lower-level statistics of the style image. The complexity of the style pattern transferred to the synthesized image increases when higher levels are considered. As expected, the best results are obtained when many layers are considered in the style loss. However, the presence of too many style layers increases the computational cost. The optimal trade-off between the quality of the results and the computational cost is found by considering only the layers `conv1_1`, `conv2_1`, `conv3_1`, `conv4_1`, `conv5_1`, in agreement with the original paper.

As a last comparison, we explore the difference in performances obtained using the Adam optimizer instead of the L-BFGS optimizer used so far. Adam is well-known for being the most efficient optimizer in the training of most networks, however, the optimization task of neural style transfer is very different from the task of parameter optimization in a neural network. In Fig. 6 we compare two images synthesized in 600 iterations using the style of "Starry Night" without any control on color. Although the image obtained with Adam only shows the lower style pattern, the result of L-BFGS also reproduces the higher-level patterns of the style image in the same number of iterations. The same comparison applied to other images always gives the same result, which experimentally confirms the choice of L-BFGS as the best optimizer for this task.

VI. An alternative approach: NST with AdaIN

We have proved so far that Gatys' algorithm is very effective in capturing the target style representation. However, it requires a slow iterative optimization process for each input image pair, which limits its practical application. For instance, it is impossible to run the algorithm as a real-time application, even on modern high-end GPUs. Several approximations based on feed-forward neural networks have been proposed to speed up neural style transfer. Unfortunately, the speed improvement comes at a cost: the network is usually tied to a fixed set of styles and therefore cannot adapt to arbitrary new styles.

In this section, we implement the simpler approach proposed by Huang & Belongie in [4]. This approach is based on the use of the so-called *Adaptive Instance Normalization* (AdaIN) layer that aligns the mean and variance of the content features with those of the style features. Our implementation allows content-style trade-off control in both training and testing. On the one hand, this method (which we refer to as *NST-AdaIN*) enables arbitrary style transfer in real-time, combining



Content L-BFGS Adam

Fig. 6: Effect of the use of different optimizers on Gatys' architecture. The style considered is the one of "Starry Night".

the flexibility of Gatys' optimization-based framework and the speed similar to the fastest feed-forward approaches. On the other hand, in some cases *NST-AdaIN* leads to results of slightly worse visual quality.

A. Adaptive Instance Normalization

NST-AdaIN is inspired by the *Instance Normalization* (IN) layer, which is surprisingly effective in feed-forward style transfer. To explain this effectiveness, Huang & Belongie propose a new interpretation of IN. In detail, given the results of their experiments, they argue that IN performs a form of style normalization by normalizing feature statistics, i.e., the mean and variance, which have been found to carry the style information of an image.

Motivated by their interpretation, they introduce a simple extension to IN, namely, AdaIN. While IN normalizes the input to a single style, AdaIN adapts it to arbitrarily given styles. In short, AdaIN performs style transfer in the feature space by transferring feature statistics. When a content input x and a style input y are given, AdaIN simply aligns the channel-wise mean and variance of x to match those of y. Unlike Batch Normalization (BN) or IN, AdaIN has no learnable affine parameters. Instead, it adaptively computes the affine parameters from the style input,

$$\text{AdaIN}(x, y) = \sigma(y)\,\frac{x - \mu(x)}{\sigma(x)} + \mu(y). \tag{8}$$

In other words, it scales the normalized content input with $\sigma(y)$, and shift it with $\mu(y)$. Similarly to IN, these statistics are computed across spatial locations.

B. Architecture

In Fig. 8 we show an overview of the *NST-AdaIN* network that we implemented according to the architecture proposed in [4]. It has a simple encoder-decoder architecture, in which the encoder f is fixed to the first few layers (up to `relu4_1`) of a pre-trained VGG-19. In short, the network takes a content image c and an arbitrary style image s as inputs, and synthesizes an output image that recombines the content of the former and the style of the latter. In detail, after encoding the content and style images in the feature space, both feature maps are fed to an AdaIN layer that aligns the mean and variance of the content feature maps to those of the style feature maps, producing the target feature maps t,

$$t = \text{AdaIN}(f(c), f(s)). \tag{9}$$

A randomly initialized decoder g is trained to map t back to the image space, generating the stylized image $T(c, s)$,

$$T(c, s) = g(t). \tag{10}$$

The decoder mostly mirrors the encoder, with all pooling layers replaced by nearest up-sampling to reduce checkerboard effects. Furthermore, reflection padding is used in both f and g to avoid border artifacts.

As we will see in Sec. VI-C, the degree of style transfer can be controlled during training by adjusting the style weight λ. In addition, *NST-AdaIN* allows content-style trade-off at test time by interpolating between feature maps that are fed to the decoder, i.e., by adjusting the weight α,

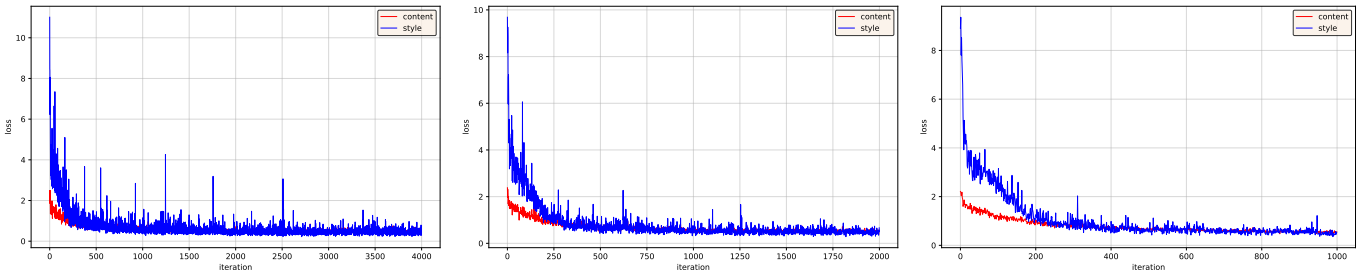
$$T(c, s, \alpha) = g((1 - \alpha) f(c) + \alpha \text{AdaIN}(f(c), f(s))). \tag{11}$$

Note that this is equivalent to interpolating between the affine parameters of AdaIN. The network tries to faithfully reconstruct the content image when $\alpha = 0$, and to synthesize the most stylized image when $\alpha = 1$.

Another important architectural choice is whether the decoder should use instance, batch, or no normalization layers. In fact, IN normalizes each sample to a single style while BN normalizes a batch of samples to be centered around a single style. Both are undesirable when we want the decoder to generate images in vastly different styles. Thus, the network does not use normalization layers in the decoder. Indeed, Huang & Belongie proved in [4] that IN/BN layers in the decoder hurt performances.

C. Training

As in the original implementation, we use the first few layers (up to `relu4_1`) of a pre-trained VGG-19 as encoder. The pre-trained Torch model `vgg_normalised.t7` (Lua language) is available at the repository AdaIN-style, which contains the official code for [4]. Since we work with PyTorch, we converted it to the PyTorch model `vgg_normalised.pth` using the Python script `convert_torch.py` available at the repository convert_torch_to_pytorch.



batch = 8, n_iter = 4000 batch = 16, n_iter = 2000 batch = 32, n_iter = 1000

Fig. 7: *NST-AdaIN*, training curves of $\mathcal{L}_c$ and $\mathcal{L}_s$.

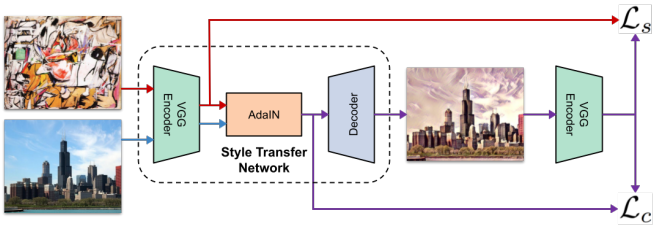


Fig. 8: Overview of *NST-AdaIN*. The first layers of a pre-trained and fixed VGG-19 network are used to encode the content and the style images. An AdaIN layer is used to perform style transfer in the feature space. A decoder is trained to invert the AdaIN output to the image spaces. The same VGG encoder is used to compute a content loss $\mathcal{L}_c$ and a style loss $\mathcal{L}_s$.

We train the *NST-AdaIN* network using PASCAL VOC [14] (11,530 images) as content dataset and ArtBench [15] (60,000 images) as style dataset. This is a deviation from the original paper implementation by Huang & Belongie, where they use WikiArt as style dataset and MS-COCO as content dataset. As in the original implementation, we use the Adam optimizer with a learning rate of 10^{-4} and $\beta_1 = \beta_2 = 0.9$. Furthermore, we use the scheduler `ReduceLROnPlateau` to reduce the learning rate when the total loss stops decreasing. Following Huang & Belongie, we use batches of 8 content-style image pairs. In addition, we also try to use batches of 16 and 32 image pairs to explore the effect of batch size on both training stability and visual quality.

Regarding the pre-processing of the input images:

- *training*: we first resize the smallest dimension of both images to 512 while preserving the aspect ratio, then randomly crop regions of size 256×256;
- *testing*: we resize the style image to the content image, which can have any size since the network is fully convolutional.

We use the pre-trained VGG-19 to compute the loss function $\mathcal{L}$ used to train the decoder,

$$\mathcal{L} = \mathcal{L}_c + \lambda \mathcal{L}_s, \tag{12}$$

which is a weighted combination of the content loss $\mathcal{L}_c$ and the style loss $\mathcal{L}_s$ with the style loss weight λ. As in the original implementation, we use $\lambda = 0.1$.

The content loss $\mathcal{L}_c$ is the Euclidean distance between the target features and the features of the output image. We use the AdaIN output t as the content target, instead of the commonly used feature responses of the content image. The authors claim that this leads to slightly faster convergence and also aligns with their goal of inverting the AdaIN output t,

$$\mathcal{L}_c = \|f(g(t)) - t\|_2 . \tag{13}$$

The style loss $\mathcal{L}_s$ only matches the mean and standard deviation of the style features, since the AdaIN layer only transfers these statistics. Although the authors find that the commonly used Gram matrix loss can produce similar results, they match the IN statistics because it is conceptually cleaner,

$$\begin{aligned} \mathcal{L}_s = &\sum_{i=1}^{L} \|\mu(\phi_i(T)) - \mu(\phi_i(s))\|_2 + \\ &+ \sum_{i=1}^{L} \|\sigma(\phi_i(T)) - \sigma(\phi_i(s))\|_2 , \end{aligned} \tag{14}$$

where $T = g(t)$ and each ϕ_i denotes a layer in VGG-19 used to compute the style loss. As in the original implementation, we use `relu1_1`, `relu2_1`, `relu3_1`, `relu4_1` layers with equal weights. The training curves of $\mathcal{L}_c$ and $\mathcal{L}_s$ are shown in Fig. 7. We can see that, as expected, the losses become more stable as the batch size increases.

VII. *NST-AdaIN* Results and Comparison with Gatys

A. Comparison with Gatys' algorithm

In this section, we compare example NST results generated using the following methods:

1) *NST-AdaIN* (our implementation)
2) *NST-AdaIN* (original implementation)[1]
3) Gatys et al. (our implementation)

In Fig. 10 we show examples of NST results generated by the compared methods. Note that all the test content and style images are *never* observed during the training of both *NST-AdaIN* implementations. In some cases (e.g., rows

[1]We use the pre-trained Torch model `decoder.t7` provided by the authors at the official repository AdaIN-style and we convert it to a PyTorch model using the Python script `convert_torch.py`.

Fig. 9: *NST-AdaIN*, effect of batch size on the same content-style image pair.

1,2,3) the quality of the images generated by both *NST-AdaIN* implementations is quite competitive with that of Gatys' results. In some other cases (e.g., rows 4,5), both *NST-AdaIN* implementations are slightly behind Gatys' quality. As claimed by Huang & Belongie, this is not unexpected, as there is definitely a three-way trade-off between speed, flexibility, and quality. Furthermore, we can see that in some cases (e.g., rows 4,5) our *NST-AdaIN* implementation leads to evidently worse results than the original implementation. This could be due to the fact that the original decoder model was trained for more iterations and on different datasets, larger than those used by us (roughly 80,000 images each).

B. Batch size

In Fig. 9 we show example *NST-AdaIN* results from the same content-style image pair, generated using decoder models trained with different batch sizes. In detail, we explore the following configurations:

- $\texttt{batch\_size} = 8,\ \texttt{n\_iter} = 4000$;
- $\texttt{batch\_size} = 16,\ \texttt{n\_iter} = 2000$;
- $\texttt{batch\_size} = 32,\ \texttt{n\_iter} = 1000$.

On the one hand, as already shown in Fig. 7, in the training phase a larger batch size leads to more stable losses. On the other hand, in the testing phase the batch size does not seem to play a crucial role, at least in the above configurations and using our implementation.

C. Content-style trade-off

As mentioned in Sec. VI-B, *NST-AdaIN* allows the user to control the content-style trade-off at test time by adjusting the weight α. Note that this control is only applied at runtime using the same network, without any modification to the training procedure. As shown in Fig. 11, a smooth transition between content-similarity and style-similarity can be observed by changing α from 0 to 1.

VIII. Concluding Remarks

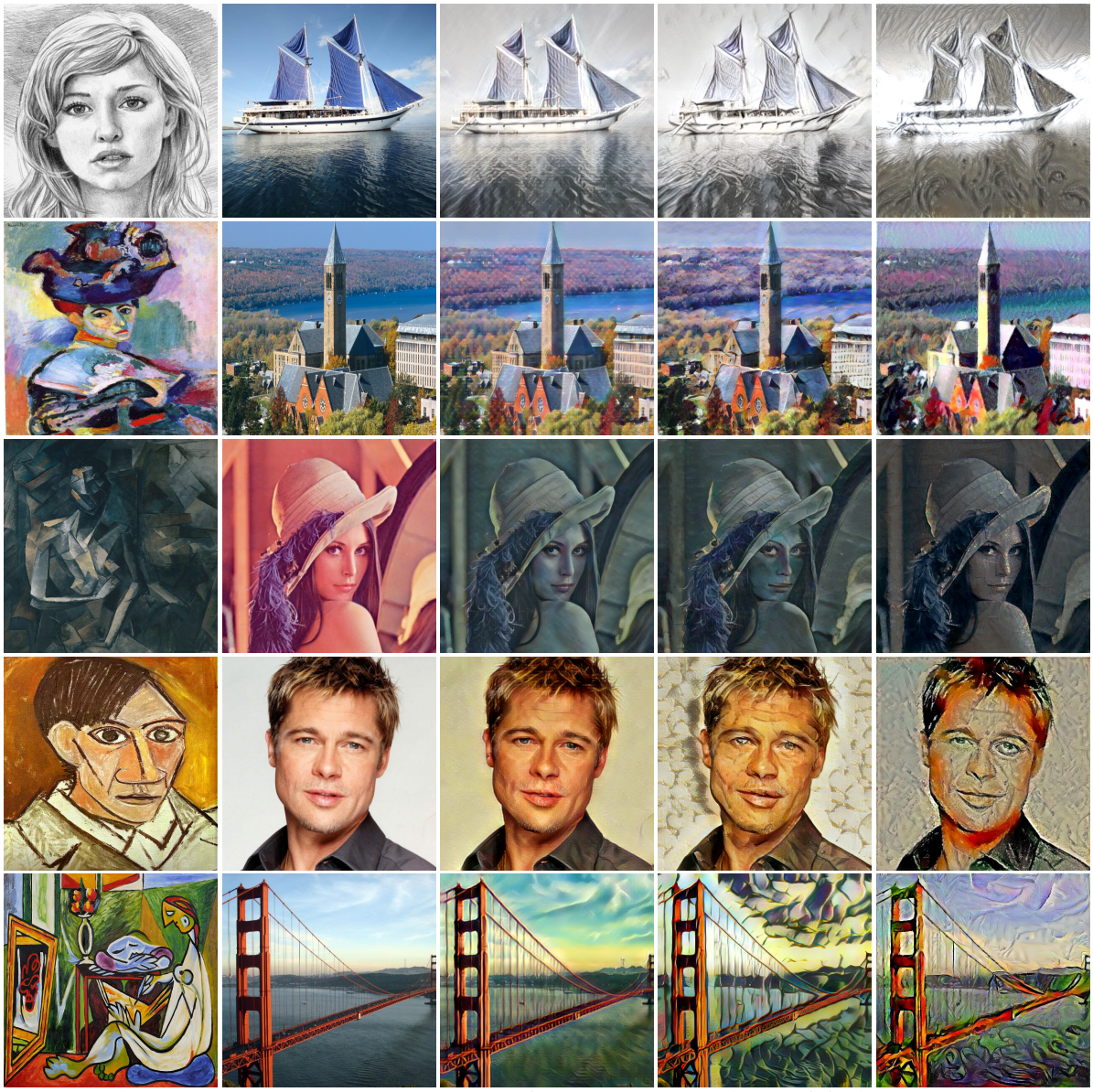
We have implemented both Gatys' and Huang's algorithms to perform Style Transfer with Deep Neural Networks. Gatys' approach has proved to be genuinely able to capture the style of many famous artworks and transfer them to photos of landscapes and portraits. The idea to use high-level features extracted by a deep CNNs is effective for those styles characterized by unique brush strokes or patterns - like "The Scream" or "Starry Night", or in general Impressionist pictures. Nevertheless, the approach is less effective for styles that involve combinations or deformation of shapes - like Cubism and Surrealism. There is still some margin of improvement, but yet the most effective way to achieve a better result is to play with the hyperparameters of the algorithm in order to find a good trade-off.

A further extension of our work, for instance, could be to tackle the style transfer problem between two realistic photos. Indeed, there exists a branch of I2I literature entirely devoted to this task: Deep Photo Style Transfer [16].

Huang's algorithm has been more challenging to work out, as the results were not completely satisfying and, in general, not up to Gatys' aesthetic yield. We have explored different training datasets and tuned several optimization parameters, eventually getting closer to the original result of the authors. Nonetheless, it has been instructive to work with this specific approach, as we acknowledge its power in a perspective of a real-time application on consumer devices. In this case, the improvement area is an open book. The AdaIN layer aligns only basic statistics of the input images, i.e. channel-wise mean and variance. It is worth some further research, perhaps with more advanced architectures, such as residual networks and skip connections, or higher-order statistics.

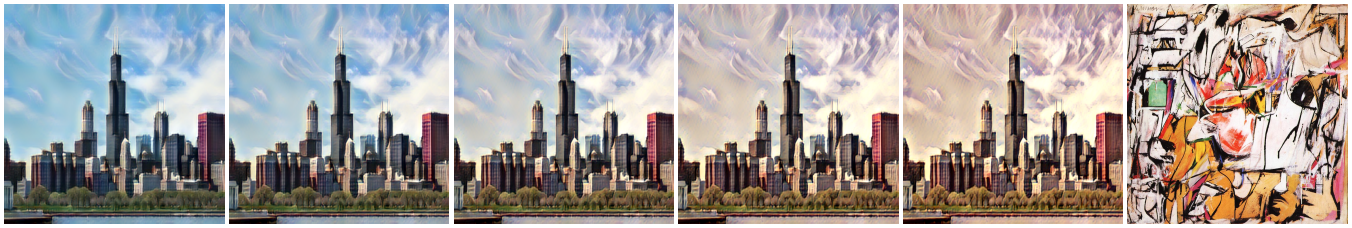
Member contributions to the project:

- **Francesco Pio Barone** and **Paolo Zinesi** have produced the code for Gatys and its improvements (color control + high-resolution). Francesco Barone has also written the *deepstyle* library, wrapping the NST algorithm with a user-friendly Python interface.
- **Daniele Ninni** has produced the code for Huang arbitrary style transfer with AdaIN.
- All the students have contributed to the coding with discussions and meetings.
- All the students have contributed to the writing of the report.



Style | Content | *NST-AdaIN* (ours) | *NST-AdaIN* (original) | Gatys et al. (ours)

Fig. 10: Example NST results. All the test content and style images are never observed by *NST-AdaIN* during training.



$\alpha = 0$ | $\alpha = 0.25$ | $\alpha = 0.5$ | $\alpha = 0.75$ | $\alpha = 1$ | Style

Fig. 11: *NST-AdaIN*, content-style trade-off. At runtime, we can control the trade-off by changing the weight α.

REFERENCES

[1] A. A. Efros and W. T. Freeman, "Image quilting for texture synthesis and transfer," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 341–346, 2001.

[2] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2414–2423, 2016.

[3] L. A. Gatys, A. S. Ecker, M. Bethge, A. Hertzmann, and E. Shechtman, "Controlling perceptual factors in neural style transfer," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3985–3993, 2017.

[4] X. Huang and S. Belongie, "Arbitrary style transfer in real-time with adaptive instance normalization," in *ICCV*, 2017.

[5] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin, "Image analogies," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, (New York, NY, USA), pp. 327–340, Association for Computing Machinery, 2001.

[6] C. Li and M. Wand, "Combining markov random fields and convolutional neural networks for image synthesis," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2479–2486, 2016.

[7] L. A. Gatys, M. Bethge, A. Hertzmann, and E. Shechtman, "Preserving color in neural artistic style transfer," *arXiv preprint arXiv:1606.05897*, 2016.

[8] D. Ulyanov, V. Lebedev, A. Vedaldi, and V. Lempitsky, "Texture networks: Feed-forward synthesis of textures and stylized images," Association for Computing Machinery, 2016.

[9] D. Ulyanov, A. Vedaldi, and V. 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*, pp. 6924–6932, 2017.

[10] V. Dumoulin, J. Shlens, and M. Kudlur, "A learned representation for artistic style," 2017.

[11] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, and M.-H. Yang, "Diversified texture synthesis with feed-forward networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3920–3928, 2017.

[12] T. Q. Chen and M. Schmidt, "Fast patch-based style transfer of arbitrary style," *arXiv preprint arXiv:1612.04337*, 2016.

[13] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015.

[14] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, pp. 303–338, June 2010.

[15] P. Liao, X. Li, X. Liu, and K. Keutzer, "The artbench dataset: Benchmarking generative models with artworks," *arXiv preprint arXiv:2206.11404*, 2022.

[16] F. Luan, S. Paris, E. Shechtman, and K. Bala, "Deep photo style transfer," 2017.