Neural Style Transfer

Gloria Isotton
gloria.isotton@studenti.unipd.it

Sara Munafò
sara.munafo@studenti.unipd.it

Abstract

This report outlines our recreation of Gatys et al.'s Neural Style Transfer method, known as Neural Style Transfer. Using images recognized by specialized Convolutional Neural Networks, we aimed to clearly capture vital details in pictures. Our main goal was to separate and blend content and style from natural images, creating visually high-quality images merging arbitrary photograph content with artwork aesthetics.

1. Introduction

Neural style transfer is an artificial system based on a Deep Neural Network that creates artistic images of high perceptual quality [5]. Starting with two images, a content image, and a style image, this algorithm creates a new picture that combines the content of the content image with the style of the style image. While the global arrangement of the original photograph is preserved, the colors and local structures that compose the overall scenery are provided by the artwork. The system uses neural representations of the fed images (feature maps) to separate and recombine the content and style of pictures.

This study focuses on advancing the understanding of how human comprehend and generate artistic imagery. The representation of style in neural networks involves the computation of correlations among neurons in different layers within the network. This process mirrors a biologically plausible computation observed in the primary visual system (V1), specifically by complex cells. These complex cells, found in V1, naturally engage in the extraction of correlations between neurons. This connection between the neural style transfer mechanism and biological processes highlights the alignment of computational strategies in artificial neural networks with the functions observed in the human visual system.

In this report we achieve our goal to implement the Neural Style Transfer algorithm as proposed by Gatys et.al. [5], and analyze its performance. The source code is available in the GitHub repository at https://github.com/isottongloria/Vision-Cognitive-Systems_NST

1.1. Related Work

Since the mid-1990s, the art theories behind the appealing artworks have been attracting interest not only among artists but also among numerous researchers in the field of computer science. A plethora of studies and methodologies have emerged, exploring how to automatically turn images into synthetic artworks [7]. Within the computer vision community, style transfer is usually studied as a generalised problem of texture synthesis, which is to extract and transfer the texture from the source to target [2], [3], [4], [1]. In the late 2000's Hertzmann and colleagues [6] introduced a framework called "image analogies" to execute a generalized style transfer, acquiring knowledge from analogous transformations observed in pairs of unstylized and stylized images.

In recent times, drawing inspiration from the power of Convolutional Neural Networks (CNNs), Gatys and collaborators [5] first studied how to employ a CNN for creating artistic imagery by separating and recombining image content and style. This work is usually regarded as a gold standard in style transfer algorithms. The application of CNNs to transform a content image into diverse styles, commonly referred to as Neural Style Transfer (NST), has gained significant prominence over time in both academic discourse and industrial applications, attracting growing interest. Numerous approaches have been put forth to enhance or broaden the original NST algorithm [8].

1.2. Dataset

For this particular task, the need for an extensive training set is avoided, as we leverage the pre-trained VGGNet19, excluding the fully-connected layers, which has been trained on the ImageNet dataset. ImageNet is a visual database containing more than 20,000 categories, each comprising several hundred images. Consequently, there is no substantial training involved for this neural network; rather, the emphasis lies in refining the initialized image to achieve the desired outcome.

For our experiments, we considered a total of 5 style images and 3 content images, sourcing them from rawpix-

els.com. A selection of these images will be showcased in the experiments section.

Before subjecting these images to the model, it is important to undergo a preprocessing stage. The initial step involves the conversion of images from an array with values in the range [0, 255] to a torch.FloatTensor of shape (channel x height x width), specifically (3 x 512 x 512) in our case, ensuring the output lies within the range [0.0, 1.0]. Following this conversion, we tested two different approaches: in the first approach we kept the output in the [0.0, 1.0] range, while in the second we tried carrying out a different normalization, using the mean values and the standard deviation from the ImageNet dataset, which is also a common practice when working with visual recognition tasks. The results obtained displayed a much clearer output image when the range of the images was kept between 0.0 and 1.0, so we chose to keep that range four our pixel values.
Additionally, image dimensions were standardized to facilitate efficient computation of the loss function during the subsequent training phase.

1.3. Method

The methods used in our work are strongly related to the NST algorithm proposed by Gatys et.al. in [5], and involve the use of the 16 convolutional and 5 pooling layers of the VGG19 network. In order to produce an image which is the combination of a content image and a style image, we need to feed the CNN three images: the content image $\bar{p}$ from which the network will have to extract important features, the style image $\bar{s}$ from which the network will have to extract recurrent stylistic features, and a target image $\bar{t}$ which will have to be optimized by the CNN in order to obtain the desired result.
As outlined in the original paper, defining the Content and Style representations is crucial for measuring the "distance" between content and style images and the target image $\bar{t}$. When the network is fed an image, in each layer of the CNN a series of filters is applied to the image in order to detect important features; the result of this series of convolutions is encoded in the so called Feature map, whose size depends on the depth of the layer and the number of filters applied. Each feature map encodes information about the important features of the input image at a given stage in the network, and it becomes more and more abstract as the image moves through the layers of the CNN, retaining feature information and discarding spatial information.
We can exploit the feature maps to measure the overall loss, which is a measure of the 'distance' between the target image and both the style and content images :

$$\mathcal{L} = \alpha \mathcal{L}_{\text{content}} + \beta \mathcal{L}_{\text{style}} \tag{1}$$

where α and β are appropriate weights which we will discuss about later.

1.3.1 Content representation

As for the content representation, we can associate the content of an image with its corresponding feature map. Given T^l and F^l the feature maps of $\bar{t}$ and $\bar{p}$ at layer l in the CNN, the content loss is computed as the mean squared distance between the two feature representations:

$$\mathcal{L}_{\text{content}} = \frac{1}{2} \sum_{i,j} (F_{ij} - T_{ij})^2 \tag{2}$$

1.3.2 Style representation

In order to define the style loss, we need the appropriate style representation. When we search for stylistic characteristics in an image, we are not interested in the presence of any single specific feature, rather in co-occurrences between pairs of feature, which define the style. This can be measured by the Gram matrix, which is the dot product between the feature map and its transpose; given the feature map F^l at layer l, the Gram matrix in layer l is defined as:

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

where k is the index referring to the different filters applied to the image up to layer l.
Calling Gs^l style gram matrix and Gt^l the target gram matrix in layer l, the contribution of layer l to the style loss is given by:

$$E_l = \frac{1}{N} \sum_{i,j} (Gs^l_{ij} - Gt^l_{ij}) \tag{4}$$

where N is the normalization constant of the gram matrices at layer l (the product of the dimensions of the feature map). The style loss can be computed as:

$$\mathcal{L}_{\text{style}} = \sum_l w_l E_l \tag{5}$$

where w_l are the weights associated to each layer.

1.3.3 Loss regularization

The goal of the style transfer algorithm, then, is minimizing the overall loss. By doing so, we ensure that the distance between the final generated image (meaning the optimized target image) and both the style and content images is minimum, granting a good resemblance to both. This is done via a backpropagation algorithm, where the gradient of the loss with respect to the target image is computed, and the pixels of the target image are therefore updated to minimize the value of the loss. In order to regularize the loss and to encourage spatial smoothness in the output image, we added

an extra term to the overall loss [8], which is the total variation loss. This regularizing term is computed on the target image $\bar{t}$. For a generic image I, the total variation loss is defined as follows:

$$\mathcal{L}_{\text{total\_variation}}(I) = \sum_{i,j} \left(|I_{i+1,j} - I_{i,j}| + |I_{i,j+1} - I_{i,j}| \right) \tag{6}$$

This extra term is also multiplied by an appropriate weight tv_{weight}, so the final formula of the loss is given by:

$$\mathcal{L} = \alpha \mathcal{L}_{\text{content}} + \beta \mathcal{L}_{\text{style}} + tv_{\text{weight}} \mathcal{L}_{\text{total\_variation}} \tag{7}$$

Not all implementations of the neural style transfer algorithm make use of a regularization technique, but for our implementation we noticed a significant improvement in the smoothness of the image by adding the total variation loss.

1.3.4 Implementation

The implementation of our code relies on PyTorch, a machine learning framework based on the Torch library.
From 'torch.nn' we imported the pretrained VGG19 neural network, and selected all its layers up to layer 36 (neglecting the last fully-connected layers).

Table 1: Convolutional layers of VGG-19 adapted for 3x512x512 input. Layers used for computing the content loss are underlined in red, while those used for calculating the style loss are underlined in blue.

Layer	Output Size	Parameters
Input	3x512x512	-
Conv1_1	64x512x512	3x3, 64
Conv1_2	64x512x512	3x3, 64
Pool1	64x256x256	2x2 max pooling
Conv2_1	128x256x256	3x3, 128
Conv2_2	128x256x256	3x3, 128
Pool2	128x128x128	2x2 max pooling
Conv3_1	256x128x128	3x3, 256
Conv3_2	256x128x128	3x3, 256
Conv3_3	256x128x128	3x3, 256
Conv3_4	256x128x128	3x3, 256
Pool3	256x64x64	2x2 max pooling
Conv4_1	512x64x64	3x3, 512
Conv4_2	512x64x64	3x3, 512
Conv4_3	512x64x64	3x3, 512
Conv4_4	512x64x64	3x3, 512
Pool4	512x32x32	2x2 max pooling
Conv5_1	512x32x32	3x3, 512
Conv5_2	512x32x32	3x3, 512
Conv5_3	512x32x32	3x3, 512
Conv5_4	512x32x32	3x3, 512
Pool5	512x16x16	2x2 max pooling

For the initial experiments we compute feature reconstruction loss at layer `conv4.2` and style reconstruction loss at layers `conv1.1, conv2.1, conv3.1, conv4.1` and `conv5.1` of the VGG-19 network as highlighted in table 1. The weight of each style loss was $1/5$ for all layers.

The loss functions were implemented using the 'torch.nn.MSELoss()' function. This function was employed to compute the mean squared loss between feature maps for the estimation of the content loss and the mean squared loss between gram matrices for the style loss.

The target image was generated as random uniform noise between -90 and $+90$, normalized, and converted into a tensor using 'torch.from_numpy'. Before the training we employed 'requires_grad_(requires _grad=True)' method on the target tensor. This ensures that the tensor's gradient is stored in an attribute called 'grad', guaranteeing that the image gets updated during training.

Throughout the training process, the generated image undergoes iterative updates aimed at minimizing the combined content and style loss. This optimization is realized by adjusting the pixel values of the target image through gradient descent. We use Adam [9] optimizer with a learning rate of 10^{-1}. The gradients necessary for this optimization are computed via loss backpropagation through the network by using the '.backward()' method. This enables the model to learn and enhance the generated image progressively, aligning it more closely with the desired content and style. During training, we also tried using both the content and style images as target images. Training takes roughly 1 hour on an AMD Ryzen 7 5800H with 8 cores, and a maximum frequency of 4.40 GHz.

1.4. Experiments

1.4.1 Preliminar tests

After implementing the algorithm, we started performance testing. In the initial tests, we excluded the regularization term from the loss function and initialized the target image with random white noise. The content and style images from Figure 1 were employed. The style loss was computed over all 5 layers [`conv1.1, conv2.1, conv3.1, conv4.1, conv5.1`].
We tested the network on 1000 iterations. After verifying that the total loss was consistently decreasing, we examined the output images, which were saved every 50 iterations.

It's important to highlight that a decrease in the loss function does not necessarily mean a high-quality output image: in the initial tests, the generated images, while exhibiting some variations with increasing iterations, remained essentially random white noise images. The parameters which we could tune in order to influence the output were the weights of each loss contribution (α, β) and the learning rate of the optimizer (lr); we quickly discovered a key issue with our parameter settings – the learning rate, initially set at $lr = 10$, was too high, leading to overshooting. By testing different values of the learning rate, we found the best

(a) Content image

(b) Style image

Figure 1: Images used as content and style for first tests.

results with a rate of $lr = 0.1$.
Once we adjusted the learning rate, the images started displaying features of both the content and the style images, however we observed persistent noise in the images. To address this, we decided to include the total variation loss term, with a weight of $tv_{\text{weight}} = 10^{-4}$. The the total variation loss weight was determined by comparing its magnitude with the content and style loss terms.

1.4.2 Exploring style weight and layer variations

By comparing all the loss terms, we found that for the specific choice of the input images in Figure 1, a well-balanced contribution from style, content, and regularization terms in the total loss is best attained by setting $\alpha/\beta \sim 10^{-6}, 10^{-5}$ and $tv_{\text{weight}} \sim 10^{-5}, 10^{-4}$.
After identifying a range of values that yielded satisfactory results, we proceeded to experiment with varying the style weight and observed its impact on the final outcome. As expected, a higher style weight produces an image where more stylistic features are present, while with a lower style weight the original content image is much more present in the final target image.
To further investigate, we also changed the subsets of layers on which the style loss term was computed. Incorporating style features from higher layers of the network results in an observed increase in complexity for the image structures represented by the style. The results of these tests are shown in Figure 2.
From this point on, we observed that after approximately 300-400 iterations both the output image and the loss function stopped displaying significant changes. In response to this observation, we decided to reduce the total number of iterations to 500.

1.4.3 Analyzing style and target image variations with a fixed content image

One of the main advantages of implementing a neural style transfer algorithm by exploiting a pre-trained network is that, upon identifying a robust set of parameters, the algorithm is expected to perform effectively in the transfer task, even when the content or style images are altered.
To evaluate this, we fed the network the same content image paired with various styles. During this experimentation, we observed that for specific styles the regularization term had a negative impact on the final result, so we decided to set $tv_{\text{weight}} = 0$ to enhance the overall result. The impact of the total variation weight is further examined in Section 1.4.4.
Contextually, we also decided to test the performance of the algorithm with a biased input target image, by initializing the target image to be the content image itself.
To achieve the results shown in Figure 3, we had to re-tune all the parameters: since in this case the final target image was much more biased towards the content than the style, the α/β ratio had to be significantly lower in order to achieve comparable results.

1.4.4 Exploring Total Variation Loss Impact

We conducted experiments to examine the influence of the regularization term 6 in the total loss. After choosing a different content and style, we aimed to understand the effects of reducing the contribution of the total variation loss on the generated image.
The results, showcased in Figure 4, aligned with expectations: reducing the contribution of the total variation loss significantly increases the noise in the image. However it's important to notice that there is a trade-off when using regularization terms like the total variation loss. If the weight is too low, the final image exhibits excessive noise. Conversely, if the weight is too high, the image becomes overly smoothed, potentially leading to the loss of stylistic and spatial features.

1.5. Conclusion

In our study, we successfully generated visually pleasing images by combining content and style images. Best results are achieved when using the content image as the target. However, the process requires manual tuning of loss weights for each input image, lacking a quantitative evaluation method: the assessment remains subjective and relies on human judgment. Additionally, the NST algorithm exhibits varying performance with different styles, influenced by the characteristics of CNN-based image reconstruction. Future work could explore experimenting with diverse content images beyond houses and various style images. With our implementation, we also noticed that no more than 300-400 iterations were needed for convergence, which might suggest that the algorithm gets stuck in a local minimum; another improvement could involve testing the algorithm at a larger scale (i.e., 5000-6000 iterations) to deepen our investigation into its convergence. Furthermore, we de-

Figure 2: Detailed outcomes for content and style images in Figure 1. Each row displays the result achieved by aligning the stylistic representation across progressively larger subsets of the CNN layers. It is observed that the localized image structures encapsulated by the style representation exhibit an augmentation in complexity as style features from higher layers of the network are incorporated. This phenomenon can be explained by the growing complexity of features as we move up the processing hierarchy of the network. The columns exhibit different relative weightings assigned to content and style reconstruction. The numerical value on the top of each column represent the ratio α/β, reflecting the balance between emphasizing the photograph's content and the style of the artwork. tv_{weight} was fixed to 10^{-4} for all simulations.

cided to set the image sizes to 512x512 to achieve higher resolution images, however the VGGNet19 was trained on 224x224 images. It would be optimal to see whether working with images of the same size might yield better results.

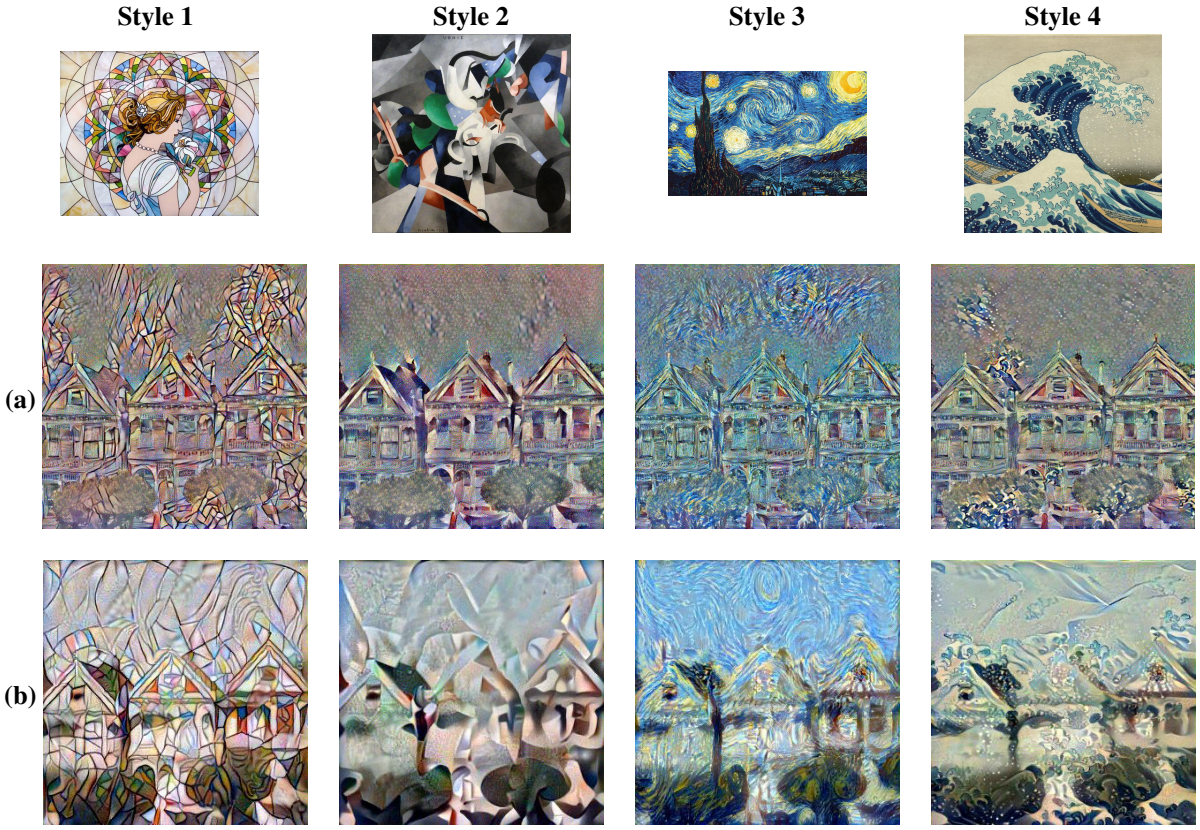

Figure 3: Results of training with different style images with: **(a)**: Random white noise image as input ($\alpha = 5.0, \beta = 6 \cdot 10^6, tv_{\text{weight}} = 0.0$). **(b)**: Content image as input ($\alpha = 1 \cdot 10^5, \beta = 5 \cdot 10^2, tv_{\text{weight}} = 0.1$).

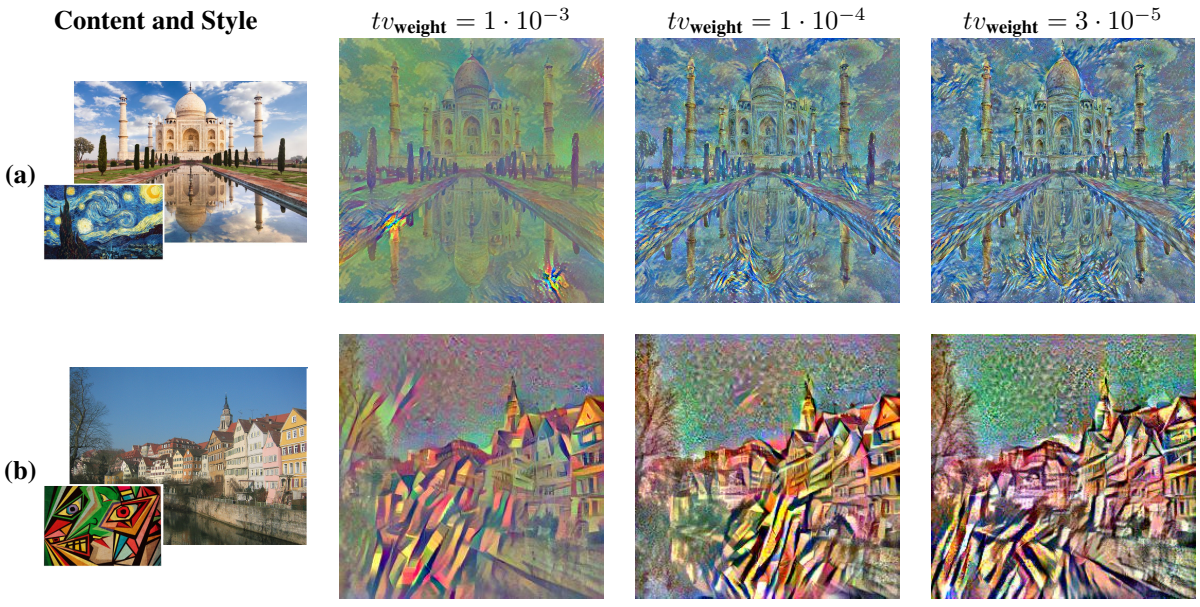

Figure 4: Results of training with different style images with: **(a)**: Results with Van Gogh style ($\alpha = 5.0, \beta = 1 \cdot 10^7$). **(b)**: Results with Picasso style ($\alpha = 5.0, \beta = 1 \cdot 10^7$).

References

[1] Iddo Drori, Daniel Cohen-Or, and Haim Yeshurun. Example-based style synthesis, 2003.

[2] Alexei A Efros and William T Freeman. Image quilting for texture synthesis and transfer, 2001.

[3] Michael Elad and Peyman Milanfar. Style transfer via texture synthesis, 2017.

[4] Olivier Frigo, Neus Sabater, Julie Delon, and Pierre Hellier. Split and match: Example-based adaptive patch sampling for unsupervised style transfer, 2016.

[5] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style, 2015.

[6] Aaron Hertzmann, Charles Jacobs, Nuria Oliver, Brian Curless, and David Salesin. Image analogies, 06 2001.

[7] Yongcheng Jing, Yezhou Yang, Zunlei Feng, Jingwen Ye, Yizhou Yu, and Mingli Song. Neural style transfer: A review, 11 2020.

[8] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution, 2016.

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.