Comparison of Neural Style Transfer Algorithms

Sharandeep Singh
(A20514069)
ssingh136@hawk.iit.edu

Tejeshwar Singh
(A20517095)
tsingh16@hawk.iit.edu

Srujan Ramesh
(A20520149)
sramesh19@hawk.iit.edu

May 2, 2023

Figure 1: Neural style transfer on a dog content image with the style image as a painting.

1 Introduction

Neural Style Transfer is an optimization approach wherein a style reference image (such as a piece of art by a well-known painter) and a content image are combined to create an output image that resembles the content image but has been "painted" in the manner of the style reference image. This is done by optimizing the output image to match the style reference image's and the content image's statistics for both content and style. A convolutional network is used to extract these data from the images.

2 Problem Description

We have tried to understand, implement, analyze and compare 2 popular neural style transfer algorithms - Image style transfer using CNN (Gatys et al) and Arbitrary style transfer (Li et al).

3 Neural Style Transfer - Gatys et al

A major limiting factor for many approaches of neural style transfer are the lack of image representations that explicitly represent semantic information and, thus, allow for the separation of image content from another image's style. Most previous texture transfer algorithms use only low-level image features of the target image to perform the texture transfer.

In this paper, they have utilized Convolutional Neural Networks (specifically, VGG19) to obtain image representations that are tailored for recognizing objects, thereby capturing significant information in images. To achieve a successful style transfer, the algorithm needs to extract the meaning behind the image content of the target image (such as the objects and overall scene), and use that information to apply the desired style from the source image. To do this, it's crucial to have image representations that can distinguish between variations in the semantic content of the image and its style.

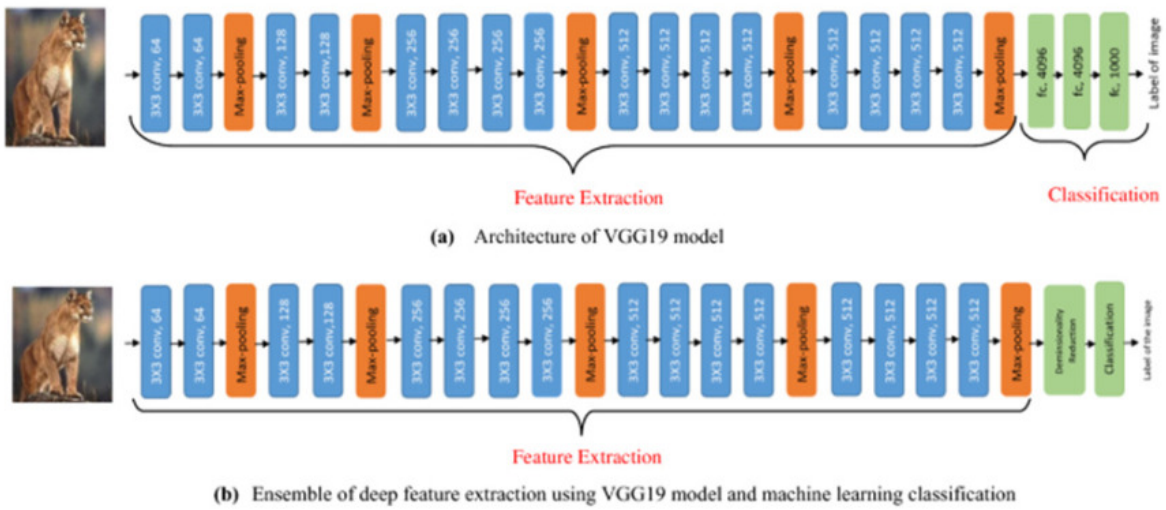


Figure 2: Information about VGG19 parts

3.1 VGG19

The progress made in Deep Convolutional Neural Networks (particularly VGG19) has been significant, due to the ability to train them with ample labeled data for specific purposes such as identifying objects. As a result, these networks can extract complex image information and represent it in a general feature format. This feature format enables the separate processing and modification of the content and style of natural images.

The outcomes presented below were created utilizing the VGG network, which was trained to recognize and localize objects. We utilized the feature space produced by a standardized version of the VGG network's 16 convolutional and 5 pooling layers, which were normalized by adjusting the weights to make the mean activation of each convolutional filter over images and positions equivalent to one. This normalization can be carried out without altering the VGG network's output, as it solely includes rectifying linear activation functions and no pooling or normalization over feature maps. We disregarded any of the fully connected layers. When generating images, we discovered that using average pooling instead of maximum pooling resulted in slightly more attractive results, which is why the images demonstrated were created using average pooling.

3.2 Architecture

The proposed method for image style transfer involves extracting and storing the content and style features from the input images. The style image is passed through the network to compute its style representation at all layers, while the content image is passed through the network to obtain its content representation in a single layer.

A random white noise image is then passed through the network to compute its style and content features. The element-wise mean squared difference between the style features of the random image and the style representation of the style image is computed to obtain the style loss. The mean squared difference between the content features of the random image and the content representation of the content image is also computed to obtain the

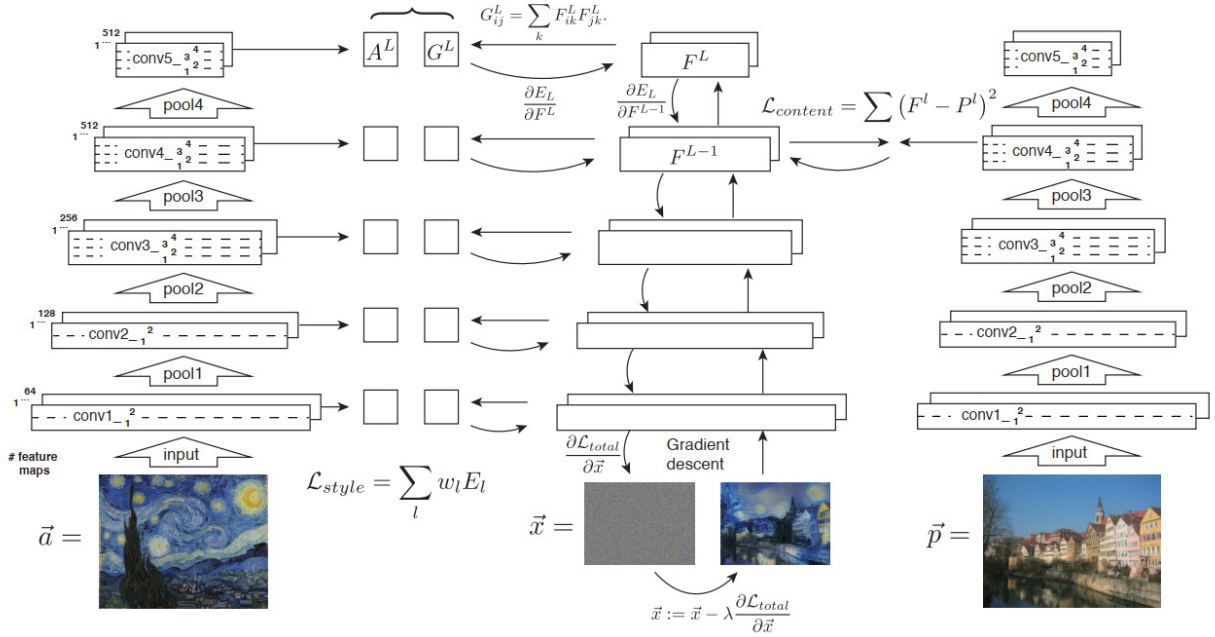


Figure 3: Architecture of NST - Gatys'

content loss.

The total loss is a linear combination of the style and content loss. The gradient of the total loss with respect to the pixel values of the random image is computed using error back-propagation. This gradient is then used to iteratively update the image until it matches both the style features of the style image and the content features of the content image.

3.3 Algorithm

To generate the output image, the authors use an optimization process to iteratively adjust the pixel values of a random noise image to minimize the difference between its content representation and the content representation of the content image, while also maximizing the similarity between its style representation and the style representation of the style image. It uses the following formulae to calculate losses

$$L_{total} = \alpha L_{content} + \beta L_{style} \tag{1}$$

A layer with N_l distinct filters has N_l feature maps each of size M_l, where M_l is the height times the width of the feature map. So the responses in layer l can be stored in a matrix $F_l \in R^{N_l M_l}$ where F^l_{ij} is the activation of the ith filter at position j in layer l.

$$L_{content}(\vec{p}, \vec{x}, \vec{l}) = \frac{1}{2} \sum (F^l_{ij} - P^l_{ij})^2 \tag{2}$$

Where,
p = Given image
x = Generated image

P_l and F_l their respective feature representation in layer l. Then,

$$L_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l \tag{3}$$

Where,
a = Given image

x = generated image

A_l and G_l are the respective style representations in layer l.

Using this loss function's derivative, we can perform backpropagation. Thus we can change the initially random image x until it generates the same response in a certain layer of the CNN as the original image p. As the network processes the input image, the representations it generates become more sensitive to the content of the image while becoming less dependent on the exact appearance of the image. Thus, higher layers, for example, 'conv4 2' (d) and 'conv5 2' (e), in the network capture the high-level content in terms of objects and their arrangement in the input image but do not constrain the exact pixel values of the reconstruction very much.

In contrast, reconstructions from the lower layers, for example, 'conv1 2' (a), 'conv2 2' (b), 'conv3 2' (c), simply reproduce the exact pixel values of the original image.

4 Arbitrary Style Transfer

Fast approximations with 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 cannot adapt to arbitrary new styles.

This paper presents a simple yet effective approach that not only does style transfer in real-time but is also adapted to arbitrary new styles and not tied to a fixed set of styles. At the heart of this method is a novel adaptive instance normalization (AdaIN) layer that aligns the mean and variance of the content features with those of the style features. This is all done using a single feed-forward neural network.

One of the major drawbacks of many feed-forward methods is that each network is constrained to a limited number of styles or runs much slower than Gatys'; framework. However, a new approach, inspired by instance normalization (IN), has been developed here that is capable of transferring novel styles in real-time. This technique works by reparameterizing the data distribution at each step to ensure uniformity, which has been surprisingly effective in feed-forward style transfer.

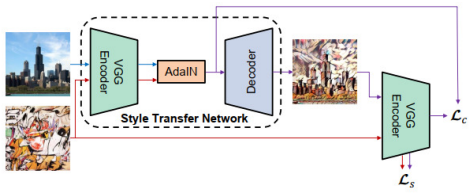


Figure 4: Architecture of AdaIN

$$IN(x,y) = \sigma(y)(\frac{x-\mu(x)}{\sigma(x)}) + \mu(y) \tag{4}$$

In essence, AdaIN transfers feature statistics, notably the channel-wise mean and variance, to accomplish style transfer in the feature space. Similar to the suggested style swap layer, the AdaIN layer serves a similar purpose. The AdaIN layer is as easy as an IN layer and adds essentially no computational overhead, whereas the style swap operation requires a lot of time and memory.

4.1 Architecture

They have encoded the content and style images using the first few layers of a fixed VGG-19 network. In the feature space, style transfer is carried out using an AdaIN layer. The AdaIN output is decoded to the image spaces using a decoder that is trained. They computed a content loss Lc and a style loss Ls using the same VGG encoder, which are given as follows:

$$L = L_content + \lambda L_style \tag{5}$$

Where, content loss is the Euclidean distance between AdaIN output target t and output image f(g(t)).

$$L_content = ||f(g(t)) - t||_2 \tag{6}$$

And,

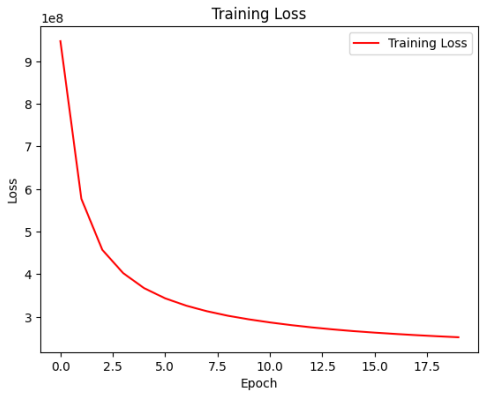


Figure 5: Training Loss graph for NST - Gatys'

$$L_{style} = \sum_{i=1}^{L} ||\mu(\phi_i(g(t))) - \mu(\phi(s))||2 + \sum^{L} i = 1||\sigma(\phi_i(g(t))) - \sigma(\phi(s))||_2 \quad (7)$$

where ϕ_i denotes a layer in VGG-19 used to compute the style loss.

5 Results

We have run the implementations on google colab free GPUs, wherein we have used the following setup in the NST - Gatys' implementation:

Optimizer = LBFGS
feature extractor = VGG19
datasets = Human Faces, Best artwork of all time
Epochs = 20

we have used the following setup in the AdaIN implementation:

Optimizer = Adam
feature extractor = VGG19
datasets = Best artworks of all time, Pascal VOC 2007
Epochs = 30

Link for the github repo: https://github.com/srujrs/Neural-Style-Transfer

(a) Content (b) Style (c) Epoch 1

(d) Epoch 5 (e) Epoch 10 (f) Epoch 15

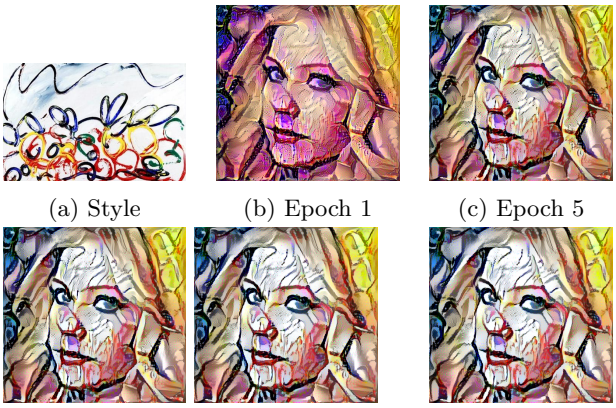
(g) Epoch 20

Figure 6: Output over few epoch for transferring first style using NST - Gatys'

6 Conclusion

The key difference between these two methods is the way they represent and manipulate the style information. Gatys neural style transfer uses a pre-trained CNN to extract style features from an image, while arbitrary style transfer directly manipulates a set of style parameters. This can result in different levels of transfer, with Gatys neural style transfer often producing more detailed and high-quality transfer results, while arbitrary style transfer can be more flexible and easier to apply to a wide range of styles.

After comparing the two models runtime, we conclude that Gaty's algorithm is simple to implement and requires one content image and one style image whereas AdaIN model is difficult to implement and requires to be trained on huge sets of content and style images. In terms of runtime, Gatys model



(a) Style (b) Epoch 1 (c) Epoch 5

(d) Epoch 10 (e) Epoch 15 (f) Epoch 20

Figure 7: Output over few epoch for transferring second style using NST - Gatys' over the first style

takes few seconds for each epoch on a GPU and can give good results in 20 epochs as seen above. However, the time taken to generate the output is not suitable for real life applications like mobile apps. AdaIN style transfer enables real-time style transfer making it suitable for interactive applications, this is because it requires the model to be trained before hand on large datasets using powerful computational resources.

In summary, Gatys neural style transfer and arbitrary style transfer are both effective techniques for artistic style transfer in images, and the choice of method may depend on factors such as the desired level of fidelity, computational resources, and available training data.

In future works, high-resolution photos can be dealt with in particular, as the current arbitrary style transfer techniques can be computationally expensive and time-consuming. To reduce computational complexity, future research might concentrate on creating more effective algorithms or parallel processing techniques.

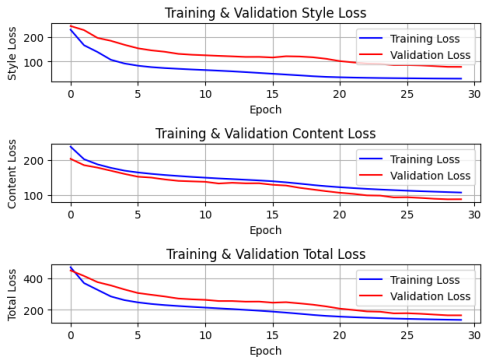


Figure 8: Training graphs for AdaIN model

7 References

- https://openaccess.thecvf.com/content_cvpr_2016/papers/Gatys_Image_Style_Transfer_CVPR_2016_paper.pdf
- https://openaccess.thecvf.com/content_ICCV_2017/papers/Huang_Arbitrary_Style_Transfer_ICCV_2017_paper.pdf
- https://openaccess.thecvf.com/content_cvpr_2017/papers/Li_Diversified_Texture_Synthesis_CVPR_2017_paper.pdf
- https://www.v7labs.com/blog/neural-style-transfer
- https://www.kaggle.com/search?q=image+style+transfer+in%3Adatasets
- https://keras.io/examples/generative/adain/

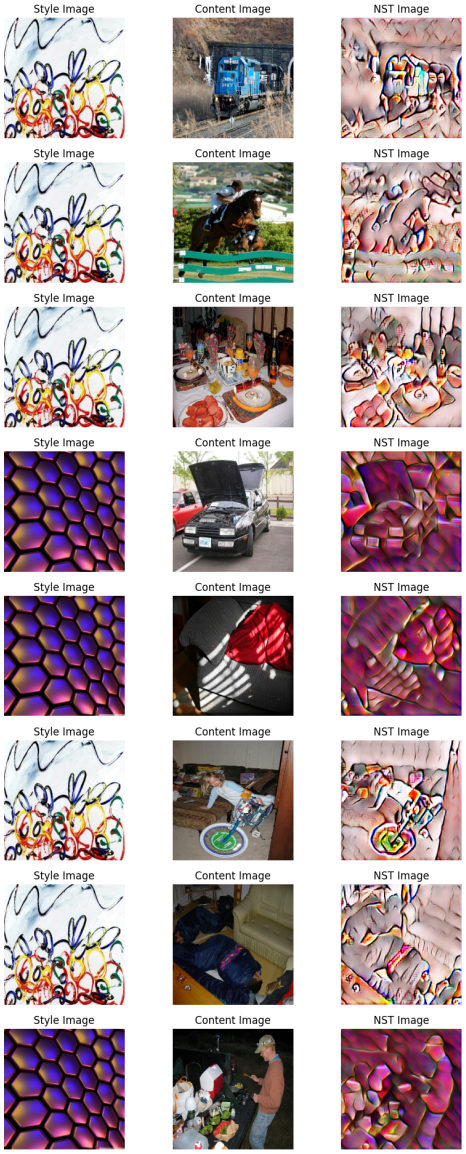


Figure 9: Outputs obtained for AdaIN for the same 2 styles used in Gatys'