Neural Style Transfer: An End-to-End Computation

Shuyi Huo (sh3989)
Siqi Chen (sc4647)
Weihan Chu (wc2688)

Abstract

Image style transfer has been one of the most discussed and potential markets with the development of deep learning techniques. It is easy to apply in the real business scenario, monetize faster than other complicated deep learning projects, and with great visualization and interpretation advantages for non-experts, especially for investors. The main purpose of this paper is to establish a neural-style transfer implementation. We will explain mathematical backgrounds of the technique, present step-wise results, and explore different parameters of the model.

1 Introduction

Image style transfer is a technique that can learn the image feature (painting, photography, etc.) from one input called style image to apply that feature on another input called content image and compute the 'merged' image as the output. It is similar to the filter on our smartphone camera but free with fixed styles, which means that we can make any style of images we want rather than choosing certain template filters from mobile apps.

This paper will explore how to implement a high-performance Image Style Transfer algorithm from scratch and apply it to establish a transferred photo from different content and style inputs. Our work is mainly inspired by the paper *Image Style Transfer Using Convolutional Neural Networks* (Gatys, Ecker, and Bethge 2016, 2414-2423). To compute a high-performance algorithm, our team utilized a famous image processing pre-trained CNN named VGG-19. For output validation, we use different content pictures (portrait, scene etc.) and different style pictures (famous paints from variate artistic genres) to compute our target images. Since there is no absolutely quantified method to evaluate the performance of the model, our team made a brief inference and conclusion based on our aesthetic.

Our team selected 2 images each for content and style as input. We try to combine one style image and one content image to achieve texture transfer.

Figure 1: Style Image 1

Figure 2: Style Image 2

Figure 3: Content Image 1

Figure 4: Content Image 2

2 Related Work

As we have mentioned above, our work's structure mainly comes from the paper *Image Style Transfer Using Convolutional Neural Networks* (Gatys, Ecker, and Bethge 2016, 2414-2423). It discusses the image representations derived from Convolutional Neural Networks optimized for object recognition and the neural algorithm that can separate and recombine the image content and style of images. It provides us with the theoretical and mathematical backgrounds, which we will specify later in this paper.

We also had other researches on the VGG models. For example, in the paper *Very Deep Convolutional Networks for Large-scale Image Recognition* (Simonyan and Zisserman 2014), the authors investigated the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting. It showed that 16–19 weight layers improve the prior-art configurations.

After understanding the theoretical foundation of the neural style transfer, we consulted a few coding examples online to code our project. From the neural style transfer tutorial on the Tensorflow website and source code from Keras-team, we refer to these as the starting point and learn how to resize images and build infrastructure.

3 Methods

Our problem setups have the following steps: preprocessing data, setting up pre-trained image classification network, constructing loss function, and model implementation.

3.1 Preprocessing data

We defined a function to load an image and limit its maximum dimension to 512 pixels. We loaded two content images and two style images in this step.

3.2 VGG Network

Building a neural network requires large amounts of training and validation data, along with high-performance computing machines. Therefore, we used a pre-trained neural network model by the Visual Geometry Group (VGG). Simonyan and Zisserman (2014) examined the effect of the convolutional network depth on its accuracy. They promoted a significant improvement on the prior-art configurations by pushing the depth of the model to 16–19 weight layers.

Hence, in our implementation, we used VGG19 network architecture. The pre-trained image classification network is below.

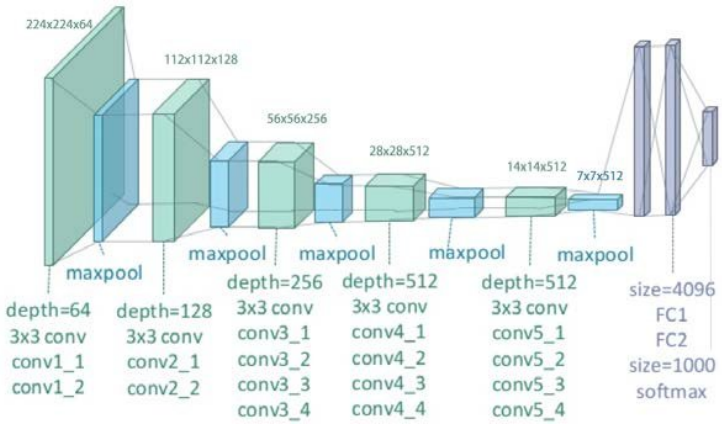

Figure 5: Network Architecture of VGG-19 Model

3.3 Loss function

3.3.1 Content Loss

From the paper (Gatys, Ecker, and Bethge 2016, 2414-2423), we know that a given input image $\vec{x}$ is encoded in each layer of the Convolutional Neural Network by the filter responses to that image. 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 a layer l can be stored in a matrix $\mathrm{F}_l \in \mathrm{R}^{M_l \times M_l}$ where F^l_{ik} is the activation of the i^{th} filter at position j in layer l.

Let $\vec{p}$ and $\vec{x}$ be the original image and the image that is generate, and P_l and F_l their respective feature representation in layer l. We then define the content loss as following:

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

The content loss measures the l_2 loss between output graph and the input content image.

3.3.2 Style Loss

To obtain a representation of the style of an input image, Gatys, Ecker, and Bethge (2016, 2414-2423) uses a feature space designed to capture texture information. These feature correlations are given by the Gram matrix $\mathrm{G}_l \in \mathrm{R}^{N_l \times N_l}$, where G^l_{ij} is the inner product between the vectorised feature maps i and j in layer l.

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

Let $\vec{a}$ and $\vec{x}$ be the original image and the image that is generated, and A_l and G_l their respective style representation in layer l. The contribution of layer l to the total loss is then:

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

The style loss on each layer is a normalized Euclidean distance between the style representation and output image. And the total style loss is the weighted average of individual style losses. Let w_l be the weighting factors of the contribution of each layer to the total loss. Then the total style loss is:

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

3.3.3 Total Variation Loss

The total variation loss is not a concept proposed in the paper(Gatys, Ecker, and Bethge 2016, 2414-2423). However, we noticed that one downside to the basic implementation above is that it produces a lot of high frequency artifacts. Hence, to decrease this, we used an explicit regularization term called the total variation loss on the high frequency components of the image (TensorFlow).

We computed the "total variation" as the sum of the squares of differences in the pixel values for all pairs of pixels that are next to each other (horizontally or vertically). And then we summed the total-variation regualarization for each of the 3 input channels:

$$L_{tv} = \sum_{c=1}^{3}\sum_{i=1}^{H-1}\sum_{j=1}^{W}(x_{i+1,j,c} - x_{i,j,c})^2 + \sum_{c=1}^{3}\sum_{i=1}^{H}\sum_{j=1}^{W-1}(x_{i,j+1,c} - x_{i,j,c})^2$$

3.3.4 Total Loss

The total loss $\mathcal{L}_{total}$ is then a linear combination of the content, style and total variation loss:

$$\mathcal{L}_{total} = \alpha\mathcal{L}_{content} + \beta\mathcal{L}_{style} + \theta\mathcal{L}_{tv}$$

where α, β, and θ represent the weights of the loss accordingly.

4 Results

4.1 Initial Result

The first sample output here is overlay the style image of Van Gogh's starry night on a content picture of New York City.

Figure 6: Style Image 1

Figure 7: Content Image 1

The model parameters used to generate the result are shown below:

Table 1: Pre-defined model parameters

Parameter Name	Parameter Value
Content Weight	0.5
Style Weight	0.5
Total Variation Weight	1
Adam Learning Rate	0.02
Adam β_1	0.99
Adam β_ϵ	0.1
Epochs	20
Steps of Each Epoch	500

With the input parameters specified above, we have the output picture shown below:

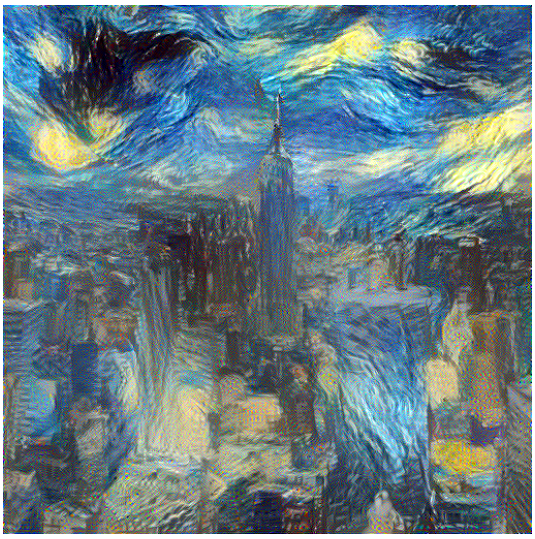

Figure 8: Output Picture 1

We can see that we successfully rendered the new york city photo with the Van Gogh Style. To better understand how the model works in the interactions, We plotted the training process every 500 steps:

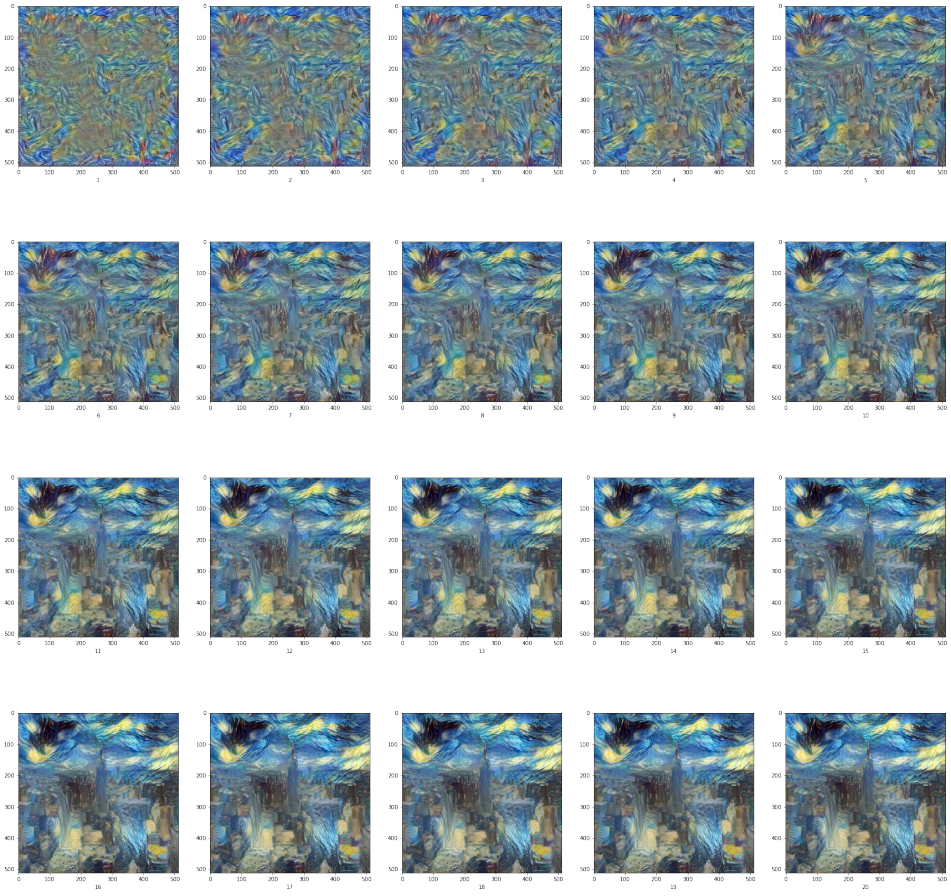

Figure 9: Training Process at Every 500 Steps

From the plot, we can see that the output picture does not change much after 5000 steps. We can also check the convergence through the loss plot:

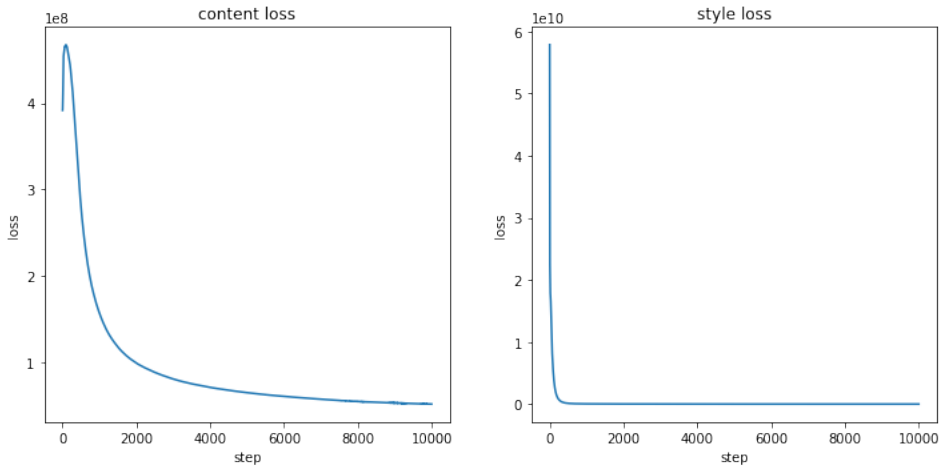

Figure 10: Content and Style Loss

4.2 More Results

With more style and content inputs, we can generate various art pictures. We have our second style and content image below:

Figure 11: Style Image 2

Figure 12: Content Image 2

We tried a few more combinations of different style and content images. We exported the following output pictures:

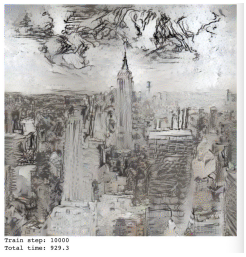

Figure 13: Output Picture 2

Figure 14: Output Picture 3

Figure 15: Output Picture 4

We also took a further look at the step-wise modelling of the output2:

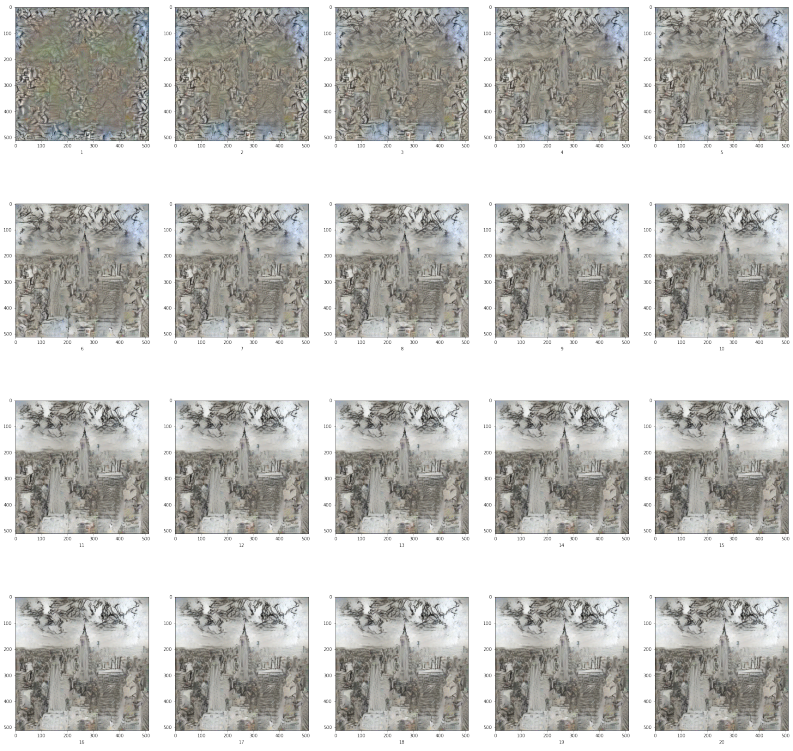

Figure 16: Training Process at Every 500 Steps

We may reach the same conclusion that the output picture does not change much after 5000 steps.

4.3 Parameter Tuning Results

We tried different optimizers to see if we can have a better result. Except for the Adam we used before, we selected Adagrad and RMSprop. The results are shown below:

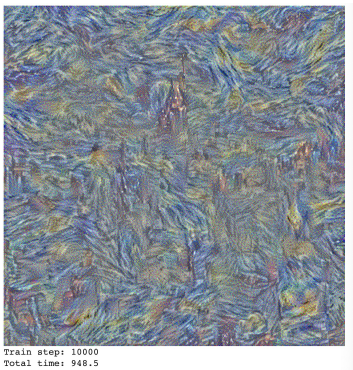

Figure 17: Optimizer: Adagrad

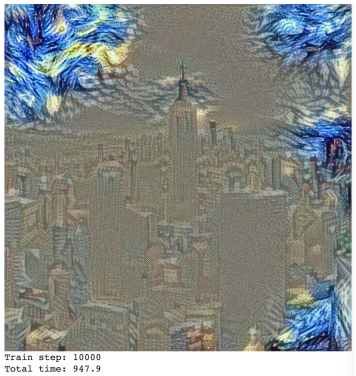

Figure 18: Optimizer: RMSprop

We can clearly see that neither of the two optimizers performed a better result than Adam.

Additionally, we examined how different content and style weights in the total loss function influence the results. α represents content weight and β represents the style weight. We can see that when the

content weight is larger, the result picture has more content, whereas when the style weight is larger, the result picture is more abstract.

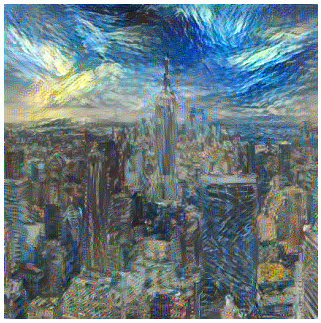

Figure 19: $\alpha = 1$, $\beta = 20$

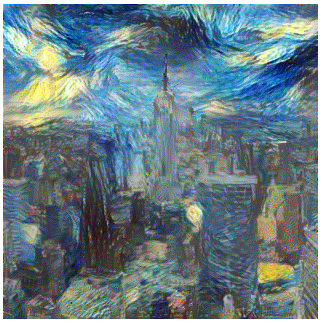

Figure 20: $\alpha = 1$, $\beta = 1$

Figure 21: $\alpha = 20$, $\beta = 1$

5 Conclusions and Future Work

From this project, our team went through the process of an end-to-end neural style transfer computation. Some important insights and future work concerns are listing below:

Model convergence speed is one of our biggest obstacles. It takes more than 10 minutes to generate a picture using Google Colab GPU through our laptop. We tried to create a web application that allows users to upload their style and content pictures and then generate new work. But we failed to do so because it is too slow to run the algorithm without the GPU in Google Colab. It takes around 700 minutes to run the web application on our computers. In the future, our team will focus on creating a doable web front-end by a faster neural style transfer algorithm (GAN, RNN, etc.), necessary parameter fixation, and functionality elimination.

We also think that the selection of optimizer matters. Our team tried Adagrad and RMSprop but they performed not so well as Adam. In the future, our team will try more optimizers such as stochastic gradient descent for faster computation.

Parameter tuning is needed. In our work, we used the pre-trained model from the TensorFlow hub, but there is a possibility for improvement in performance with parameter optimization. Different content and style loss ratios will lead to different results by focusing either on content or style. Results can be adjustable according to the user's arts aesthetics or needs. Additionally, different style or content pictures may have different ratios to achieve the best visualization. In the future, our team will focus on concluding some fashion content and style weight ratios for the public.

There are also broad applications of neural style transfer. This algorithm can do more than just image style transfer. It is similar to a voice style transfer, where you can talk like someone else by expressing your sentences in another person's voice. Neural style transfer could also be applied as a video style transfer. For example, when YouTubers are live streaming, the scenes could be changed dynamically according to selected styles.

References

Gatys, Leon A., Alexander S. Ecker, and Matthias Bethge. 2016. "Image Style Transfer using Convolutional Neural Networks.".

Keras-Team. "Neural Style Transfer with Keras.", `https://github.com/keras-team/keras/tree/fcf2ed7831185a282895dda193217c2a97e1e41d`.

Simonyan, Karen and Andrew Zisserman. 2014. "Very Deep Convolutional Networks for Large-Scale Image Recognition." arXiv Preprint arXiv:1409.1556.

TensorFlow. "Neural Style Transfer.", `https://www.tensorflow.org/tutorials/generative/style_transfer#total_variation_loss`.