GR 5242 Final Project: Neural Style Transfer

Rachel Wu (jw3663), Haiqing Xu (hx2259), Siwei Liu (sl4224), Qingyu Zhang (qz2351)
Department of Statistics
Columbia University
New York, NY 10027
jw3663@columbia.edu, hx2259@columbia.edu
sl4224@columbia.edu, qz2351@columbia.edu

Abstract

The purpose of this study is to implement a neural style transfer algorithm, which refers to the process of extracting the style of one image and applying it to a content image using Convolutional Neural Networks. Here we use a pretrained 19-layer VGG to generate content and style representations. The algorithm will produce a blended image after minimizing the total loss function, which is a weighted average of content loss and style loss. This paper further studies the impact of different parameters on the synthesized image, and the results offer great insights on what the most important factors are. We also explore applying two styles instead of one single style, producing an output image that exhibits characteristics of both styles.

1 Introduction

Neural style transfer is an optimization algorithm used to adopt the style of a specific image (the "style image") and apply the style to another image of our interest (the "content image"). Essentially, the style of some artwork is transferred onto our own picture, and with the help of this technique, people will be able to render any image into the style of any piece of artwork. The neural style transfer algorithm provides opportunities to deliver appealing blended images using neural networks, and the process could be done in just minutes.

The objective of this project is to implement our own neural style transfer algorithm to mix content pictures with style pictures. Gatys et al. has proposed a method that allows one to separate content from style in images [1]. Using the pretrained 19-layer VGG network, the aim is to get a representation of the picture in terms of content and style. This paper further studies the impact of different parameters, including number of training iterations, different random seeds, and relative weight of content and style loss in the loss function, on the result of neural style transfer.

2 Methods

2.1 Content & style representations

The intermediate layers within the pretrained VGG-19 network are used to build the content and style representations. Each layer contains some information about the original picture. After feeding an image to the neural network, the first few layers will preserve most details in the image. As the image moves deeper into the network, each layer will represent higher-level content, such as major objects and their relative positions in the image [1, 2]. For these reasons, a higher layer is chosen to generate the content representation. For style representation, multiple layers are chosen in order to further calculate the feature correlations.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

2.2 Loss function

As previously mentioned, neural style transfer is essentially an optimization algorithm, therefore we need to define a loss function. There are two components contributing to the total loss function: content loss and style loss [1].

Content loss Suppose we have a content image $\vec{c}$ and the generated image $\vec{o}$, as well as their representation in layer l, C^l and O^l, the content loss is defined as [1]

$$\mathcal{L}_{content}(\vec{c}, \vec{o}, l) = \frac{1}{2} \sum_{i,j} (C_{ij}^l - O_{ij}^l)^2 \tag{1}$$

Style loss To define the style representation of an image, first we use the Gram matrix to capture the correlations between different feature maps as in Eq. (2).

For both the style image $\vec{s}$ and the generated image $\vec{o}$, their respective Gram matrices are computed as S^l and O^l. The style loss of layer l is defined in Eq. (3).

Then we can define the total style loss, where w_l denotes layer l's relative contribution to the total style loss in Eq. (4).

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l \tag{2}$$

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (S_{ij}^l - O_{ij}^l)^2 \tag{3}$$

$$\mathcal{L}_{style}(\vec{s}, \vec{o}) = \sum_{i=0}^{L} w_l E_l \tag{4}$$

Total loss The total loss function we are going to minimize is a weighted average of content loss and style loss

$$\mathcal{L}_{total}(\vec{c}, \vec{s}, \vec{x}) = w_c \mathcal{L}_{content}(\vec{c}, \vec{x}) + w_s \mathcal{L}_{style}(\vec{s}, \vec{x}) \tag{5}$$

where w_c and w_s define the weights for content and style representation, respectively [1].

2.3 Gradient descent

In the project, the generated image is initialized with the content image, and gradient descent is performed in each training iteration to minimize the total loss function.

2.4 Effect of parameters

Following are the factors that may influence the neural style transfer process and contribute to how the resulting image looks like.

Training iterations In each training iteration, all three images (content image, style image and the generated image) are passed through the VGG-19 network, to get content and style representations. Then we calculate the derivative of the total loss function with respect to the pixels in the current generated image. In each iteration, pixels are updated in effort to minimize the total loss.

Random seeds Tensorflow is used in our code implementation to perform minimization on the total loss function. There is randomness in this process, so a random seed needs to be specified if one hopes to reproduce the same resulting image.

Relative weight of content and style loss in the loss function If we define w_c and w_s as the weights for content and style, respectively, then the ratio w_c/w_s indicates whether we'd like to focus more on style or content reconstruction. A larger ratio means a higher emphasis on the content part. Therefore, the resulting image will look more like the original photograph, and vice versa.

2.5 Neural style transfer with two styles

Inspired by a Github project, we have further explored applying two styles to one content image [3]. This could be achieved by incorporating two style losses in the total loss function, as we see in Eq. 6.

$$\mathcal{L}_{total}(\vec{c}, \vec{s_1}, \vec{s_2}, \vec{x}) = w_c \mathcal{L}_{content}(\vec{c}, \vec{x}) + w_{s1} \mathcal{L}_{style}(\vec{s_1}, \vec{x}) + w_{s2} \mathcal{L}_{style}(\vec{s_2}, \vec{x}) \tag{6}$$

where $w_c + w_{s1} + w_{s2} = 1$, and $w_{s1} = w_{s2} = \frac{1}{2} w_s$

(a) (b) (c)

(d) (e)

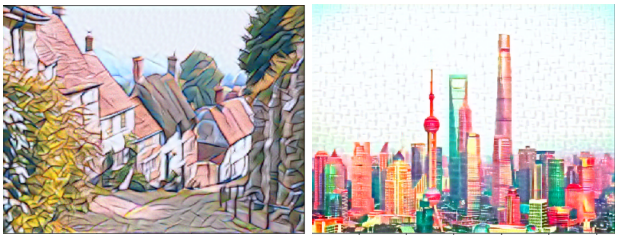
(f) (g)

Figure 1: The photographs that serve as the content for the resulting images are: (a) *Shanghai: skyline* by Charles Chen. (b) *Village* by John Gomez. (c) *NYC* by 10best.com. The art pieces that provide the style for the resulting images are also shown. (d) *Organic Wind* by Teresa Young, 2015. (e) *Shanghai Skyline Illustration* by Animexz94 (favpng.com). (f) *Little Village* by Megan Duncanson (g) *Mosaic* by Zhiyuan He.

3 Results

Fig. 1 displays a set of the content and style images used in this paper. Fig. 2 shows some examples of the resulting images when testing our neural style transfer algorithm. Due to limited space, only two images are displayed. Compared with the original photographs, the resulting images adapt the style of the artwork pretty well.

Training iterations Based on Fig.3, after 50 training iterations, most of the details in the content pictures, such as the contour and shape of major objects, are still preserved in the generated image. Meanwhile, although the style of the generated image is not strong yet, there is a clear change in the overall color of the image. At 200 iterations, one can see a strong shift in terms of the colors. At 400 training iterations, the color pattern in the generated picture looks very similar to that in the



(a) (b)

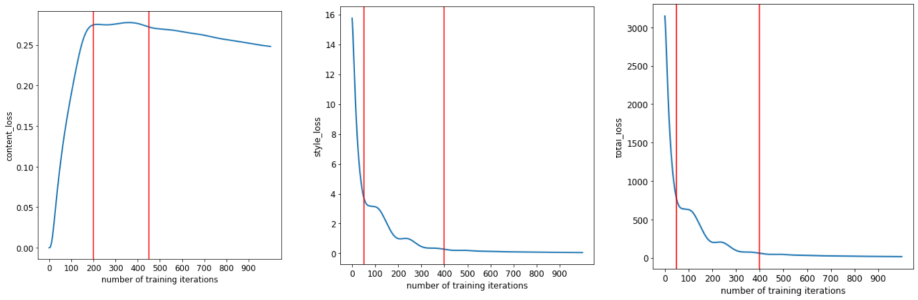
Figure 2: Examples of some resulting images from the neural style transfer algorithm. (a) Content: Fig. 1b. Style: Fig. 1g. (b) Content: Fig. 1a. Style: Fig. 1e.

style image. The tree on the left seems blurred, and it's hard to identify the leaves clearly. Compared with the output at iteration 400, the colors on the generated image at 1000 iterations do not change much. However, the resulting image now exhibits a stronger style and fewer details. There is no clear contour in the generated picture in the sense that the borderlines are vague. The distorted contours precisely reflect what we see in the style image.

(a) Iteration=50 (b) Iteration=200

(c) Iteration=400 (d) Iteration=1000

Figure 3: Resulting images with different training iterations, keeping all other parameters the same (random seed = 0, w_c/ w_s= 10^{-2}). Content: Fig. 1b. Style: Fig. 1d.



(a) (b) (c)

Figure 4: Loss functions when changing training iterations (random seed = 0, w_c/ w_s= 10^{-2}).

Fig. 4 demonstrates the change in content loss, style loss and total loss, as the number of training iterations changes. Since the generated image is initialized with the content image, at first the content loss is zero. From 0 to 200 iterations, content loss increases monotonically. There are some fluctuations in the curve between 200 and 450 training iterations, and then the curve flattens and keeps decreasing. The style loss and total loss curves follow an overall decreasing trend, which can explain the increase in stylism as the number of training iterations increases. Both curves fluctuate between 50 and 400 iterations, and become flattened after 400 training iterations. This explains what we see in Fig, 3, because after 400 iterations, there is no major change in the colors of the generated image anymore.

(a) Random seed=0 (b) Random seed=23

Figure 5: Resulting images with different random seeds, keeping all other parameters the same (iteration = 1000 , w_c/ w_s= 10^{-2}). Content: Fig. 1c. Style: Fig. 1f.

Random seeds Fig. 5 illustrates results from the neural style transfer algorithm when we adjust the random seeds with the same choice of all other parameters. Both resulting images adapt the color pattern of the style representation very well, but there's almost no difference between the two figures in terms of the texture and position of the buildings. To further explore the impact of random seeds, Fig. 6 displays the total loss functions for four different random seeds, 0, which is the default random seed selection, 23, 625, and 2019, respectively. Not surprisingly, there's no huge deviation among the four curves in general. All the four loss functions decrease drastically at first, followed by some slight differences from 100 to 300 iterations, where the four curves fluctuate the most. After 300 iterations, all curves seem to converge. Thus, compared to changing training iterations, having different random seeds does not contribute to altering the resulting images on a large scale.

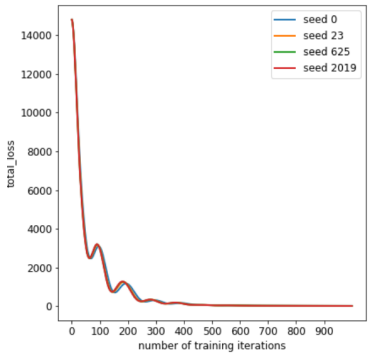
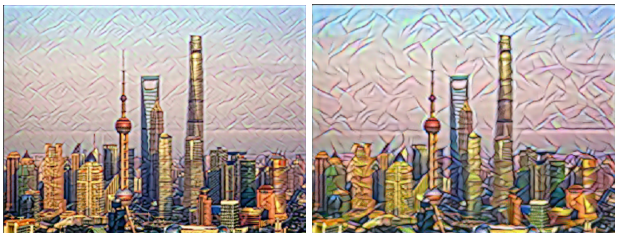


Figure 6: Loss functions when changing random seeds, keeping all other parameters the same.

Relative weight of content and style loss Fig. 7 represents the results of using various relative weights of content and style loss in the loss function. As we decrease the relative weight w_c/ w_s, where w_c represents the weight of content and w_s represents the weight of style, the resulting image will look more like the artwork in terms of the texture. However, currently there is no standard on how "good" an output image is. We can get images that look very similar to the style of some famous paintings, but it's difficult to define how well the content and style images are synthesized from a mathematical point of view. Therefore, there will be a trade-off between style and content matching, and the conclusion can be subjective. In addition, it seems that once the relative weight of w_c/ w_s reaches a threshold, the intensity of stylism in the generated image will not change drastically



(a) 10^{-1} (b) 10^{-2}

(c) 10^{-3} (d) 10^{-4}

Figure 7: Resulting images with different relative weights w_c/ w_s, keeping all other parameters the same (random seed = 0, training iteration = 1000). Content: Fig. 1a. Style: Fig. 1g.

anymore. Observing Fig. 7, as the relative weight decreases from 10^{-1} to 10^{-3}, there is a stronger emphasis on the style component. However, once the relative weight reaches 10^{-3}, there is no obvious change in stylism for the resulting image as shown when the relative weight is 10^{-4}.

Neural style transfer with two style images On top of our neural style transfer with one content image and one style image, we have further explored applying two styles to one content image. Fig. 8 shows the result and for comparison purposes, we have also included two single-style outputs. We can see that the two-style output image has successfully adopted the characteristics of two different styles. Compared with output image using style 1 only, the two-style output shows a less strong color intensity, but clearer contours. Compared with output image with style 2 only, the two-style output has less clear contours, but stronger color intensity.

(a) Styles: Fig. 1d and Fig. 1g. (b) Style: Fig. 1d (c) Style: Fig. 1g

Figure 8: Resulting images with two styles, keeping all other parameters the same (iteration = 1000 , w_c/ w_s= 10^{-2}). Content: Fig. 1b.

4 Conclusion and Discussion

In general, our algorithm has successfully transferred the style to content images. Number of training iterations is shown to have a large impact on the output image. In the first few iterations, one can observe a transition in color, in which the color pattern of the output gets increasingly similar to that of the style image. As training iterations further increase, the style of the artwork has a stronger impact on the output image, and the details from the original content image become vague. The impact of random seeds is not significant, as one can hardly tell the differences among images generated by different random seeds. The relative weight of content and style loss is also an important factor that

influences how the output image looks like. A smaller relative weight corresponds to a more intense style component. In addition, once the relative weight reaches a threshold, the intensity of style does not change much. However, note it's hard to define a good relative weight because it all depends on how the audience view the resulting image. Thus, there is definitely a trade-off between content and style when creating appealing figures.

Another thing to note is, how good the resulting image is also partially relies on the choice of content and style images. When testing the algorithm, we notice that having an artwork with lighter colors and stronger stylism will generally produce a better output image. The synthesized images may also suffer from noise and low-quality issues, but this could be solved using some resizing techniques to the original images [4].

Besides, further efforts to blend two style images with one content image produce a satisfactory result in the sense that one can clearly observe the mixture of two styles in the output image. Possible extensions to this project include performing image segmentation to specify which part of the output image shows a stronger style effect and exploring the impact of different relative weights of two style losses. h part of the output image shows a stronger style effect and exploring the impact of different relative weights of two style losses.

References

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

[2] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. *arXiv:1505.07376*, 2015.

[3] Reiichiro Nakano. Arbitrary style transfer using TensorFlow.js. *GitHub repository, https://github.com/reiinakano/arbitrary-image-stylization-tfjs*, 2019.

[4] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. In *Distill, http://doi.org/10.23915/distill*, 2016.