Neural Style Transfer Project

Xinzhu Wang(xw2581) Ying Jin(yj2453) Jincheng Xu(jx2365) Xudong Guo(xg2305)

Abstract

In this project, we explored an interesting application of image visualization in deep learning, image style transfer. According to a related paper published in 2016[1], we built the content and style representations based on a pre-trained VGG-19 model. Then we defined the weighted loss, ran Adam optimization algorithm over the output image and compared results in different parameters. Finally, we applied our final model to different content and style images, resulting in some lovely stylish content images.

1 Introduction

Among all the project topics the professor mentioned in lecture, data visualization has always caught our eyes. Later in the semester, when we read the paper[1], our group immediately decided to do our project on neural style transfer. It is interesting to see how the style of a masterpiece works on different pictures of modern architecture, cartoon characters, a random street view in Brooklyn Image style transfer has been a popular topic for years, but it's until recent years that researchers have overcome a large obstacle, extracting and transferring semantic image context. The paper presents a new way to build the model that trains independently on semantic image content and style picture. With this new approach and 19-layer pre-trained VGG model, we can achieve the transfer that both captures the general texture information of style image and keeps the objects and features of content image. We can also customize our loss function to play with different combinations of style and content.

Thus, the goal of this project is to add a painting filter to a photo based on our own model, which transfers the style of a painting to a picture. The main focus of our work is to turn the paper's methods into realization in Python, and at the same time, to test different parameters and compare their results. We also want to practice our deep learning skills and coding skills, especially in Tensorflow.

2 Methods

2.1 Preprocessing

Before putting our images into the model, we first did data preprocessing, where we resized and normalized images and created a noise image as input_img to start our model.

First, we resized content and style images so they appear in the same size. This step makes sure the consistency, which makes sure that both images have similar effects on the loss function.

Then we standardized them, for which we center the pixels in three color channels, RGB. To standardize images, we subtract content and style images by mean_pixel = [123.68, 116.779, 103.939], which is the mean per channel calculated over all images of VGG. Since we are going to share and combine information of two images, beyond their size, it's also necessary that they are on similarly-ranged feature values. Otherwise, the relative weight won't work as we desired.

Preprint. Under review.

Also, for preparing the following modeling, we generate a noise image based on the content image and randomly generated noises from np.random.uniform. Based on this noise image, we train our model towards content and style images. We set the noise ratio to be 0.6.

2.2 Modeling

After preprocessing images, we used the pre-trained VGG-19 model, extracted the feature maps of images from the layers. For content, we extracted "conv4_2". And for style, we extracted ["conv1_1", "conv2_1", "conv3_1", "conv4_1", "conv5_1"]. The layer extraction is a realization of the paper to represent the style and reconstruct the content. Then we defined loss function and ran Adam gradient descent optimization algorithm to train our model. Featuring on some important choices we made during the modeling process below.

2.2.1 VGG: Visual Geometry Group

We used the Visual Geometry Group (VGG) pre-trained model to reduce the time spent on training the network and omits the need for a large amount of training data. VGG is a rising neural network in object recognition that is based on AlexNet but with deeper layers and higher accuracy. To achieve better results, we chose VGG-19, the deepest layer in VGG groups, which greatly fastened and improved the training procedure with higher computational power.

2.2.2 Optimization function: Adam

An activation function is an important part of the model. In our case, we choose Adam as our activation function, since it combines some good properties of AdaGrad and RMSProp. It has several advantages which makes it a popular choice in deep learning recently. The most critical one is that it converges faster and more effectively compared to other traditional activation functions. So we used Adam as our activation function.

2.2.3 Loss function

As for loss function, we implemented the exact loss function that is presented in the paper. Total_loss = content_loss_weight * content_loss + style_loss_weight * style_loss. The benefit of using this loss function is to consider the style and semantic content independently but collaboratively. More specifically, the content_loss function we used is content_loss = tf.reduce_sum(tf.square(F - P)) / 2. The style_loss function is style_loss = sum(style_layers_weights[i] * E[i] for i in range(n_layers)). Also, these loss functions allow us to customize the proportion of style and content that we would like to present in our combined picture. For relative weight, we will elaborate later. Below we displayed the mathematical loss representation shown in the paper to give some insight behind our implementation.

$$L_{content}(\overrightarrow{p}, \overrightarrow{x}, l) = \frac{1}{2}\sum_{i,j}(F_{ij}^{l} - P_{ij}^{l})^2$$

$$E_l = \frac{1}{4N_l^2 M_l^2}\sum_{i,j}(G_{ij}^{l} - A_{ij}^{l})^2$$

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

$$L_{total}(\overrightarrow{p}, \overrightarrow{a}, \overrightarrow{x}) = \alpha L_{content}(\overrightarrow{p}, \overrightarrow{x}) + \beta L_{style}(\overrightarrow{a}, \overrightarrow{x})$$

2.2.4 Pooling layer

Pooling layers reduce the dimensions of the data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Max pooling uses the maximum value from each of the clusters of neurons in the prior layer. Average pooling uses the average value from each of the clusters of neurons in the prior layer. Here We used Average pooling because the author of the paper suggested that it will perform better.

3 Sample Results

(a) temple (b) Van Gogh Starry Night

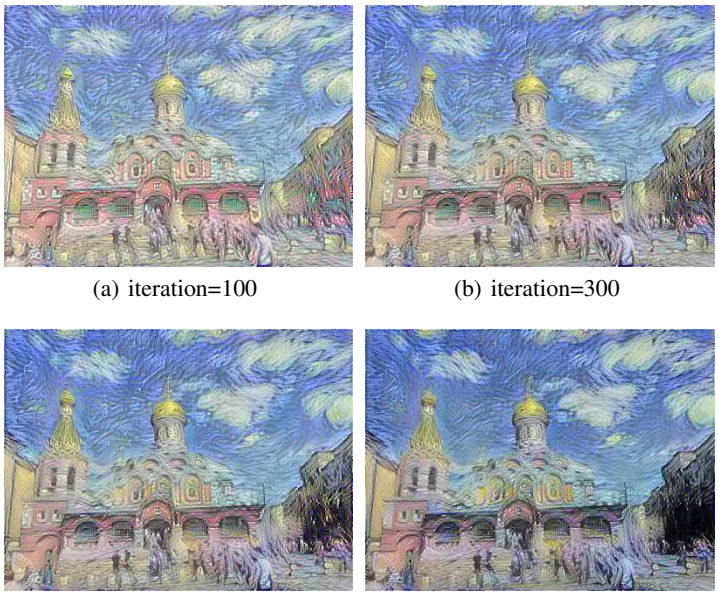
Figure 1: Raw input of temple and Van Gogh painting

3.1 Results with different parameters

We chose one style image: *Starry Night* and one content image: *Temple*, shown in Figure 1, to test model results. Accordingly, we determined the relatively best model parameters for our desired combination. Aside from the mentioned parameters below, we also tried different style-layer weights. It is concluded that style-layer weights hardly impact the results.

Here we list out four parameters to compare their outputs.

- **Iterations:** Training iterations, without batches, indicate the number of times the model is updated. Larger iterations usually lead to a better model result, but in a trade-off with computational run time. To test the influences of training iterations in our model, we ran four kinds of iterations: 100, 300, 500 and 1000. We can clearly see from Figure 2 that, the more iterations we run, the better stylish content picture we get. It may perform better if we try higher iteration, but the computational cost is too high, so we decided to keep iteration = 1000 for our final model, which already provided the desired result.



(a) iteration=100 (b) iteration=300

(c) iteration=500 (d) iteration=1000

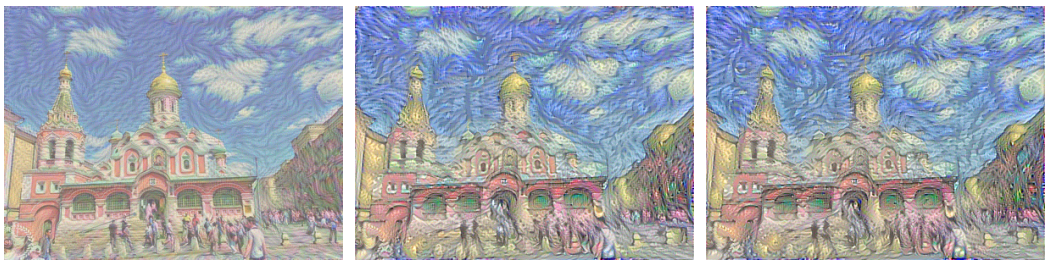
Figure 2: Results with different iterations

- **Random seeds:** To test the stability of our model, we run our model on various random seeds, 0, 2019, and 5242. Under the same relevant weight and iteration, we can see from Figure 3 that there are no obvious differences among different seeds. So we conclude that our model is stable, and we can use any random seed in our final model, and we went with default seed=2019 for our final model.

(a) seed=0 (b) seed=2019 (c) seed=5242

Figure 3: Results with different random seeds

- **Learning Rate:** Learning rate has always been an important hyperparameter in deep learning, which affects both the required time for model to converge and the performance of final result. If the learning rate is too small, our model will improve towards convergence slowly and requires a long run time to optimization. On the other hand, if the learning rate is too large, the result may swing between optimal point's left and right, and cannot reach the global optimal. In our case, we started with 0.2, which is a common choice for learning rate in deep learning. But we soon found out that neural style transfer requires a much larger learning rate to provide a better transfer. Like in Figure 4, we tried learning rate 0.2, 2.0, 3.0. And from the result, we can see that learning rate = 2.0 and 3.0 have very similar performance. Since some other content and style images originally match less on color and contents, we decided to keep the learning rate = 2.0 in case.



(a) learning_rate=0.2 (b) learning_rate=2.0 (c) learning_rate=3.0

Figure 4: Results with different learning rates

- **Relative weights:** Relative weights represent the proportion of style and content image in our final combined picture. The way we defined our loss function is: total_loss = content_loss_weight * content_loss + style_loss_weight * style_loss, where we keep our style_loss_weight=1, and play around our content_loss_weight. It's obvious that there is a trade-off between style image and content image, since we punish their losses differently. To explore the different combinations, we tried content_loss_weight 0.01, 0.001, 0.0001, which keep a stronger focus on style, but have a slightly different tendency. A smaller content loss weight should match a higher emphasis on style image, and from Figure 5, we can clearly see that our result matches our assumption. Relative_weight = 0.0001 has the strongest mark of our style image.

(a) relative_weight=0.01 (b) relative_weight=0.001 (c) relative_weight=0.0001

Figure 5: Results with different relative weights

3.2 More examples

After the parameter determination process, we finished our model with parameters, 1000 iterations, learning rate 2.0, and 0.0001 relative weights. Besides, we set the random seed to be 2019 to keep the consistency and reproducibility of our outputs. To further apply our model, we applied the model to three content images and two style images. The three content images are *Temple*, *Cafe*, and *One Piece*. The two style images are *The Starry night* by Van Gogh and *The Weeping Woman* by Pablo Picasso. The original content image, style images, and our final stylish content images are displayed in Figure 6 and 7.

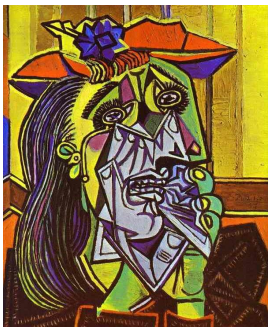
(a) temple (b) Van Gogh Starry Night (c) Overlay Van Gogh on temple

(d) cafe (e) Van Gogh Starry Night (f) Overlay Van Gogh on cafe

(g) OnePiece (h) Van Gogh Starry Night (i) Overlay Van Gogh on one piece

Figure 6: Overlay Van Gogh on temple, cafe and OnePiece

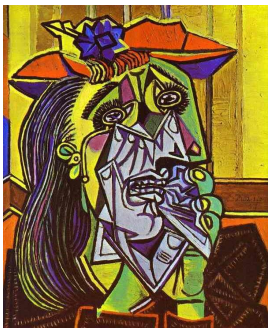
(a) temple



(b) Picasso The Weeping Woman

(c) Overlay Picasso on temple

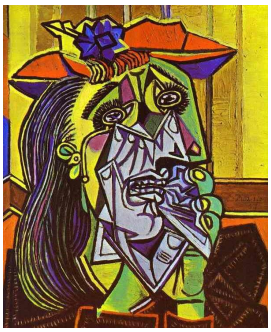
(d) cafe



(e) Picasso The Weeping Woman

(f) Overlay Picasso on cafe

(g) OnePiece



(h) Picasso The Weeping Woman

(i) Overlay Picasso on one piece

Figure 7: Overlay Picasso on temple, cafe and OnePiece

4 Conclusion

After tuning several parameters, we conclude that our model is stable since it shows similar performance under different seeds. And the preferred relative weight is 0.0001, where gives us the desired mix of style and content images. As for iteration, iteration = 1000 shows us the best combination of style and content, which keeps the general semantic content of content image while displays the style of the painting. And learning rate = 2.0 provides us the acceptable computational cost and the most desirable result. We tested our model on three different content images, *Temple*, *Cafe*, and *One Piece*. And we also applied two filters, two paintings with different styles and by different artists, *The Starry night* by Van Gogh and *The Weeping Woman*. And from the result figures below, we can see that for the same content, our model shows different filter effects of different styles while both keeping the general texture of the content image. Also, for the same filter, it shows differently on different contents, and we are able to tell which filter we applied just by looking at our final picture.

A real application of this project is to create an app that asks the user for a photo and asks him to choose a painting, then we can show the original photo under the filter of the painting.

References

[1] Gatys, L. A., Ecker, A. S., & Bethge, M. (2016) Image Style Transfer Using Convolutional Neural Networks. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). doi: 10.1109/cvpr.2016.265

[2] Nelson, Z. (2018, June 21). NELSONZHAO/zhihu. Retrieved from https://github.com/NELSONZHAO/zhihu/tree/master/image_style_transfer.

[3] haosnote.blogspot.com (2019). Temple. [online] Available at: https://2.bp.blogspot.com/-XdzgNaZBFqY/W_xQf1iQC6I/AAAAAAAAAQI/1H1BoquFXqs6U3lfDW3xj2UWbPPoG6gFwCLcBGAs/s640/%25E7%259B%25B8%25E7%2589%2587%2B2018-6-25%2B%25E4%25B8%258B%25E5%258D%25885%2B38%2B39.jpg.

[4] Vincent van Gogh. The Starry Night. 1889. Painting. Museum of Mod. Art, New York.

[5] Pablo Picasso. The Weeping Woman. 1937. Painting. Museo nacional Centro de arte Reina Sofia

[6] Sh.eastday.com. (2019). One Piece. [online] Available at: http://sh.eastday.com/images/thumbnailimg/month_1906/d1862f9ecc8745288423b04b6de6dec9.jpg.