STATGR5242
ADVANCED MACHINE LEARNING
FINAL PROJECT REPORT

LINJUN HUANG (UNI: LH2985)
JIARUI CHANG (UNI: JC5268)
MUBAI LIU (UNI: ML4407)
YI ZHANG (UNI: YZ3681)

Columbia University
Department of Statistics

Contents

GR5242

Linjun Huang
lh2985@columbia.edu

Jiarui Chang
jc5268@columbia.edu

Mubai Liu
ml4407@columbia.edu

Yi Zhang
yz3681@columbia.edu

Abstract

The research of style transfer expands a lot in recent years due to the evolution of Convolutional Neural Networks. In this report, we build a neural style transfer network based on a pre-trained VGG-19 model that will capture the style and content from the image. Choosing some specific pre-trained layer from the CNN model and then applying gradient descent, the blending of image style and image content makes this neural network an artist. There are three major improvements based on the current image style transfer paper [1]. The first idea developed from the paper is utilizing a blurred version of the content image as the initial inputs. Secondly, we preprocessed images before taking them into the different choice of CNN layers from the paper. And lastly, the noise of the output image has been optimized.

1 Introduction

1.1 Background Information

In this project, we are going to use a pre-trained image classification network (VGG-19) to compose images in the style of another image, this process is known as neural style transfer. Neural style transfer is an optimization technique used to take three images: a content image, a style reference image (such as an artwork by a famous painter), and the input image you want to style — and blend them together such that the input image is transformed to look like the content image with “painted” in the style of the style image. To sum up, the key point is we won’t have a traditional so-called "accuracy" in the process. We’ll take the base input image, a content image that we want to match, and the style image that we want to match to put them together. By minimizing the content and style distances (losses) with back-propagation, we create a new and perceptually meaningful image.

1.2 Related Works

A novel structure was introduced by the paper in 2016 using VGG-16/VGG-19 Convolutional Neural Networks to capture features and contents from images. As it has mentioned in the paper, the perceptual aspects of "style" are conceptually hard to tell for non-artists, it might be meaningful to analyze the CNN representation for each layer [1]. In a model like VGG-19, there are a total number of 16 different Convolutional layers with each layer getting distinct information from input pictures. A logical implementation would be through utilizing different layers will produce various types of results to fulfill different requirements. For the coding perspective, we take a tutorial page [4] from the Tensorflow and build upon it by changing its parts that is not our interests or modify the body part of the code. But it is still based on the paper that was introduced in 2016 mentioned above.

1.3 Improvements and Adjustments

Due to the curiosity of how would the output image changes if we modify the initial input image, we decide to implement some adjustment that is not mentioned in the paper. For the purpose of comparison, we test three unique starting images: 1. white noise image; 2. content image; 3. blurred version of content image. And eventually, we would like to test out how the initial setting could affect the final outcomes and which initial setting outperforms the others.

Another aspect of our model is the weight for each part of the loss. By setting various values of weight, we can figure out how much impact of the weights will produce on the final output. In that way, we can adjust our own weight to fit our own requirements. (i.e. appear to have more original content or more similar style compare with style image) From the Tensorflow tutorial [4], besides the content loss and style loss, we also added total variation loss to the total loss. Total variation loss could help with making the outcome image become smoother.

It is trivial to see that there are many noise points appear on our final result image based on the procedure from the 2016's paper. A new way of reducing such effect would be adding a blurring process to remove the noise points after each epoch. But this may lose some information in the graph, so in order to retain the resolution, a sharpening process is also included right after the blurring effect. We believe this set of the procedure will work very well for our neural transfer structure and the results are shown in the results section.

The final idea that comes to mind is we'd like to try other CNN models to see if it could make an improvement or not. We've decided to implement MobileNet as our second choice of CNN model. It also has a similar structure overall as mentioned above to include the variation loss as well as the noise reduction option. We'll discuss the results in the last section.

2 Method

Generally speaking, our starting point is [1]. We have done three major improvements based on [1]. Besides the initial image is set to be white noise or original content image, we introduce a blurred version of the content image as the initial image. It has less bias to the content image compare to staring with original content image. Also, its loss converge much faster than starting with the white noise. Secondly, what we have done in advance is to do some prepossessing step on both style and contact image, change the layer for gradient decent to see the difference of final output. Looking deep inside of how the weight and layer can affect the final result. Lastly, we de-noised the output image via blurring and sharpening process after each epoch.

2.1 Model

We used two different model VGG-19 model and mobilenet model to compare the speed of convergence for loss function.

2.1.1 VGG Neural Network

We choose to use VGG-19 network for train step. VGG-19 contains 16 CNN layers with ReLU as activation function. The model separate them with 5 different blocks and followed by a max pooling layers in each block. In both [1] and [2], they choose block4Conv2 as content representation, and block1Conv1, block2Conv1, block3Conv1, block4Conv1 and block5Conv1 as the representation for style image. We first follow that standard but find out some different types of content or style may performs really bad, so we change the choice of layers, and able to achieve a more flexible output, we will discuss this more in the result part.

2.1.2 Mobilenet model

The basic idea for VGG-19 is using CNN layers with ReLU activation function to capture various parts information from the original image. The differences between these two models are Mobilenet model is composed by two types of special layers named depthwise convolutional filter and pointwise convolutional layer. From [5], the main propose for depthwise filter is to cut the dimension down to 1, use a single filter for original input. However it only do the filter part, we need connect all dimensions

to get the features from image. So depthwise filter usually followed by a pointwise layer i.e. 1*1 convolution.

2.2 Loss Function

Our loss function is composed by three parts, content loss, style loss and variation loss. Content loss and style loss are multiplied by the weight we set ahead and divided by the number of layer we choose. Moreover, we add a total variation loss, which represent the sum of the absolute differences for neighborhood pixel values for the input. Adding this loss could make the output graph looks more smoother. At last the loss function comes out to be

$$L = \alpha * l_{style} + \beta * l_{content} + l_{variation}$$

We can get totally different image by changing the value of α and β.

2.2.1 Content Loss

Content loss is calculated by using the most direct way – calculating the mean squared loss between the original content image and generated image.

$$L_{content} = \frac{\alpha}{l} \sum_{l} \sum_{i,j} (p_{i,j}^{content} - p_{i,j}^{generated})^2$$

In this formula, $p_{i,j}^{content}$ and $p_{i,j}^{generated}$ represent the pixel values for content and generated image. l represents the number of layer we chosen for content loss. By implementing this, we can get how much different between the content image and output image.

2.2.2 Style Loss

Style loss is little bit difference from content. We introduce the gram matrix for loss calculation.

$$G_{ij}^{l} = \sum_{k} F_{ik}^{l} F_{jk}^{l}$$

Gram matrix is based on dot product, due to the fact that dot product can reflect the similarity between two vectors or matrix. Applying this idea to image, the larger the result is, the more similar these two vectors are. And again similar to the content loss, we calculate the mean squared loss between the gram matrix result for style part,

$$L_{style} = \frac{\beta}{l} \sum_{l} \sum_{i,j} (G_{i,j}^{style,l} - G_{i,j}^{generated,l})^2$$

And from this step we can capture the information about how much information we reserve by CNN layers i.e. lines, pattern or texture from original style image.

2.2.3 Variation Loss

Variation loss is a regulation method used to eliminate the noise generate on the boundary. The formula for variation loss is

$$L_{variation} = \sum_{i}^{m} \frac{1}{m} (p_{(i+1),j} - p_{i,j})^2$$

$$L_{variation} = \sum_{j}^{n} \frac{1}{n} (p_{i,(j+1)} - p_{i,j})^2$$

m, n represent the size of the image, the two functions shown above represent the horizontal and vertical variation loss for a image. We can even add them up for to maintain the smoothness for the output.

2.3 Prepossessing Image Noise Elimination

For the prepossessing part, we choose several ways to change the input image. Blurring, sharpening and adjusting the contrast of the original content image. We have three starting images: 1. white noise image; 2. content image; 3. blurred version of content image. We generate our white noise image by randomly select value from normal distribution with mean 0 and variance 0.5. We also generate the blurred version of the content image via a blurring process.

One of the idea we came up with is fitting the style image using calligraphy and ink painting. since they only get black and white color, so we add an black and white transfer to our content image to see what the outcome would be. Also, through each epoch, we force the output image to be black and white.

The noise elimination part was very challenging since there are no paper discussed about this part. It also plays a vital role on the final output images. We first used a blurring process to remove the noise points after each epoch. However, we found out that the final outcome image might be too blur if we only using the blurring process. In order to retain the resolution, we also introduce an image sharpening process. To code up those two process is a very tough task for us. We spent lots of time on familiarizing the structure of tensorflow image in order to write the blurring and sharpening functions. The combination of those two process works very well so far on noise eliminating.

The intuition behind Gaussian blur is similar to the basic idea of convolutional neural network, it applies the Gaussian function to each pixel. Under 2 dimension We have

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

By applying this formula to the pixel value we can generate concentric circles that are normally distributed from the center. The value of each pixel is the weighted average of the surrounding neighboring pixel values. The value of the center pixel has the largest Gaussian distribution value, so it has the largest weight. As the neighboring pixels are farther away from the original pixel, their weights become smaller and smaller.

The idea for sharpen has the opposite direction of Gaussian blur. It also uses the convolutional technique, different with Gaussian blur, this time we use a 3*3 matrix with a high value locate in the center and surround by 8 negative values. In this way, we can decrease the correlation between the central pixel and the pixel connect to it, i.e. we can make the image more obvious rather than smooth. In our research, we do the sharpen step follows by blurring, it can perfectly eliminate noise point on our output result.

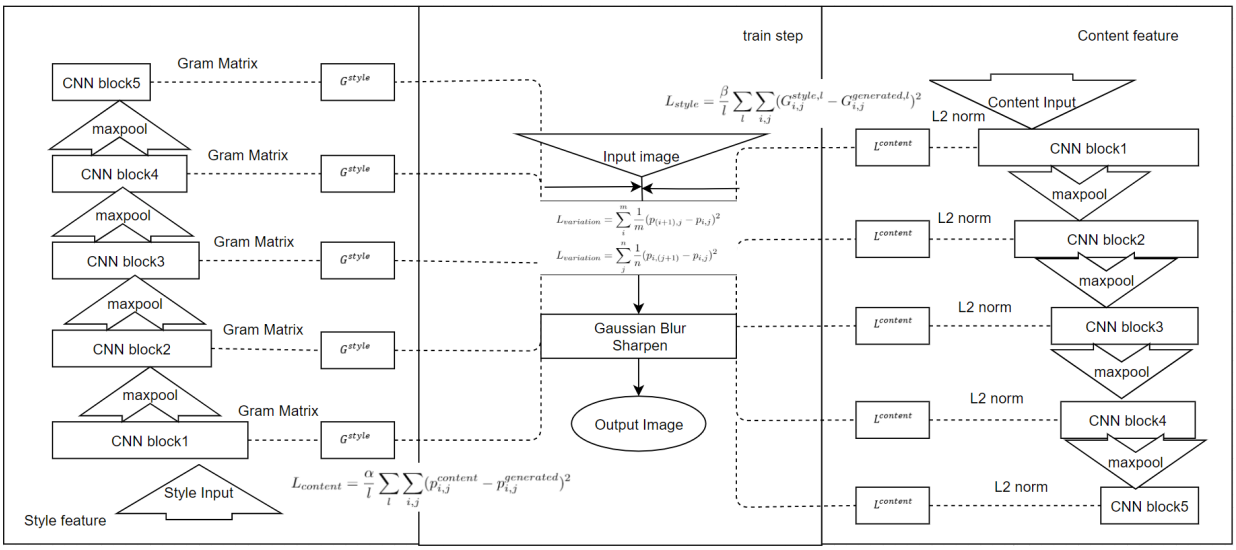


Figure 1: Implementation

3 Result

3.1 Output Image at Each Epoch Under Different Initialization

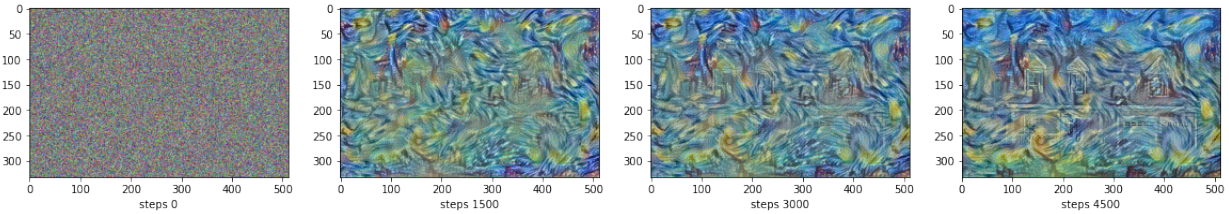


Figure 2: each epoch result for white noise initialization

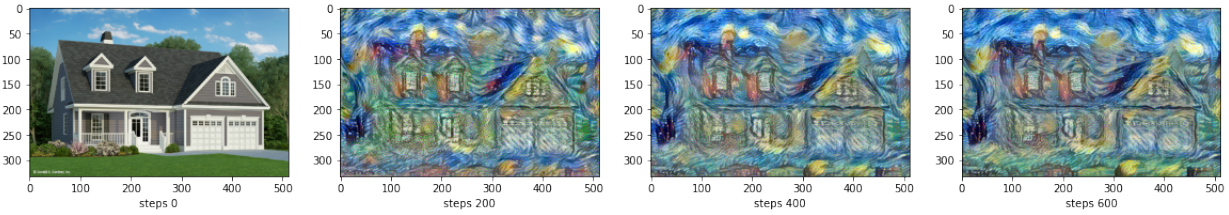


Figure 3: each epoch result for content initialization

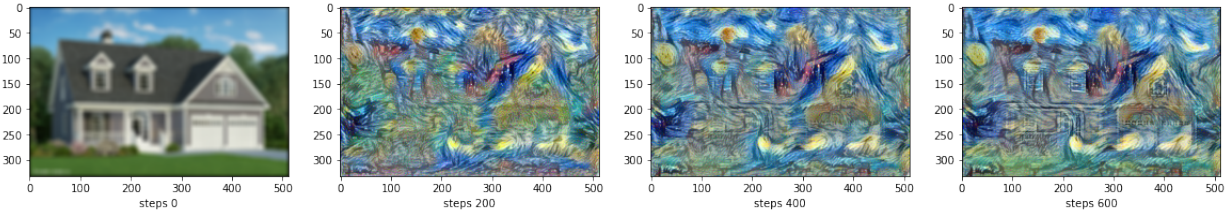


Figure 4: each epoch result for content initialization

We set the number of epoch to be 7 and the number of iteration in one epoch to be 250 for the white noise initialization and 100 for the content and blurred content initialization. The blurred content initialization and the content initialization has almost the same pattern.The speed of learning is initially very fast and slow down after the third epoch. Under the same setting, the white noise initialization has a much slow learning speed.

3.2 Denoise Implementation

Figure 5: without implement denoise method

Figure 6: implement denoise method

Figure 7: Denoise implementation

As shown above, after implementing the blurring and sharpening process after each epoch, the noise in the final output image is successfully removed.

3.3 Content Layer Selection

Figure 8: content image NYC with style image star

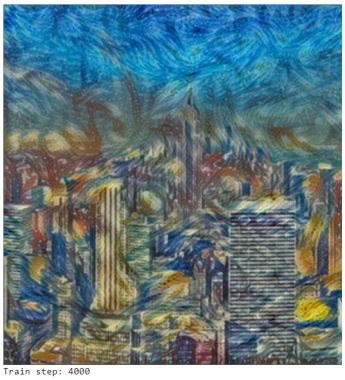


Figure 9: using content layer block1 cov2

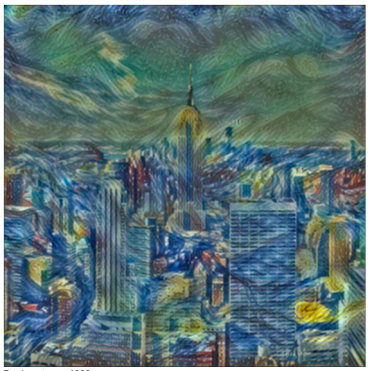


Figure 10: using content layer block2 cov2

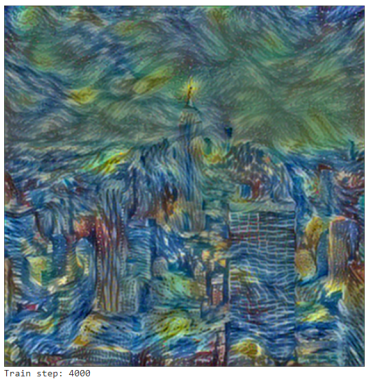


Figure 11: using content layer block3 cov2

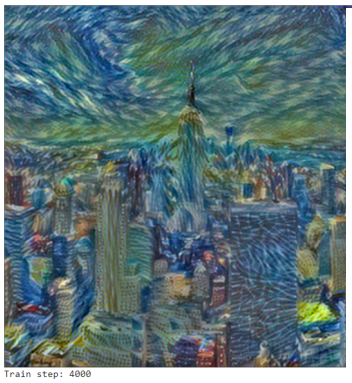


Figure 12: using content layer block4 cov2

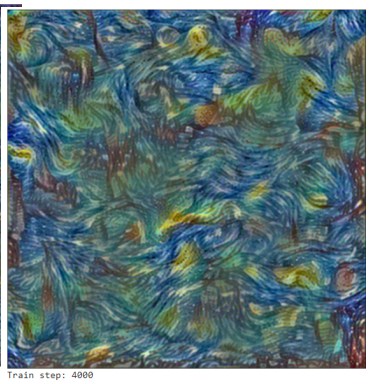


Figure 13: using content layer block5 cov2

Figure 14: content layer selection

Through our experiment, similar to the result conduct by Justin Johnson[3], if the choosing layer is closer to the first block the result tend to capture more original content including the shapes, colors, object and structure for the input content image. On the contrary, if our choice is approach to the last several layers, the result tend to capture more on the information about color and really limit amount information from structure.

3.4 Different α, β in the Setting

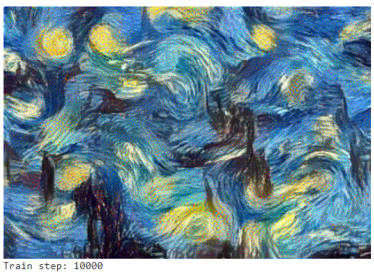


Figure 15: alpha=0.01, beta = 0.1

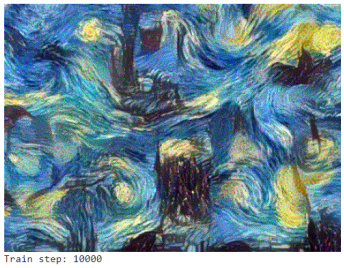


Figure 16: alpha=0.1, beta = 0.1

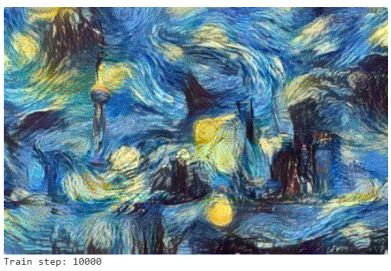


Figure 17: alpha=1, beta = 0.1

Figure 18: Result for different alpha and beta

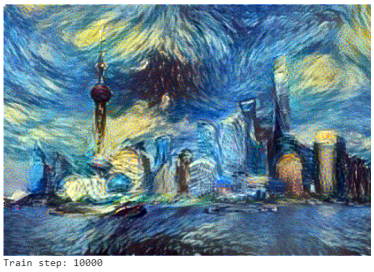


Figure 19: alpha=10, beta = 0.1

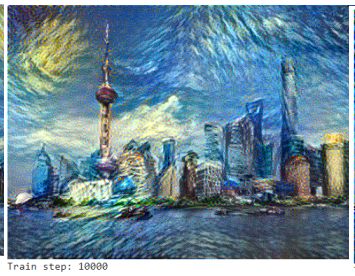


Figure 20: alpha=100, beta = 0.1

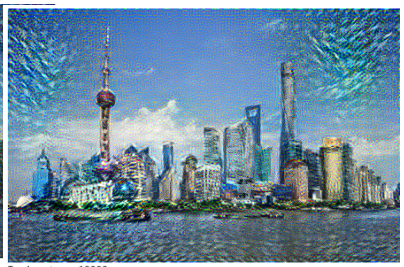


Figure 21: alpha=1000, beta = 0.1

Figure 22: Result for different alpha and beta

Described in [1], if the ratio of $\frac{\alpha}{\beta}$ is greater than one, the result image will have a strong bias toward the content image. Conversely, it will have a bias toward the style image. The optimal values in practice are $\alpha = 10$, $\beta = 0.1$.

3.5 Log Loss Plot Under Different Initialization

The total loss decreases rapidly in the first three epoch for the content image and blurred content image initialization settings. Under the same setting, the white noise initialization has a much slower convergence speed. Due to the magnitude of the loss are very big, we use a log scaling of the loss value. We implement the denoise method after each epoch. Comparing the log loss plot without denoise method and with denoise method, the style loss jumps up after each epoch while the variational loss jump down after each epoch. During the denoise process includes two part: blurring and sharpening. We use the blurring method to eliminate the noise points in the image. To retain the resolution of the image, we do a sharpen process. During the denoise process, the shapes in the image are blurred and the lines around the shapes are cleared. The blurred shapes decrease the total variational loss. Also, The lines in image increases the style loss since there are no line in the original style image. This may be one of the reason that makes those jumps. In both cases, the blurred content image initialization outperforms the other initialization settings. If the style image is less abstract saying more realistic, then the content initialization will have the best performance.

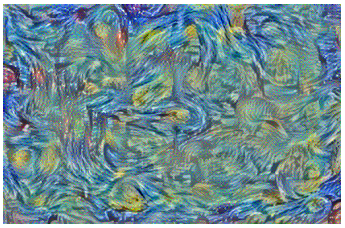


Figure 23: white noise initialization

Figure 24: content image initialization

Figure 25: blur content image initialization

Figure 26: result images from log loss plot without denoise method

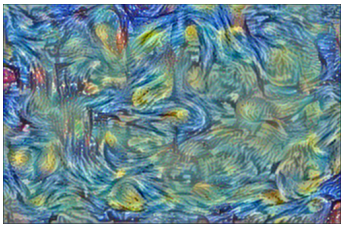


Figure 27: white noise initialization

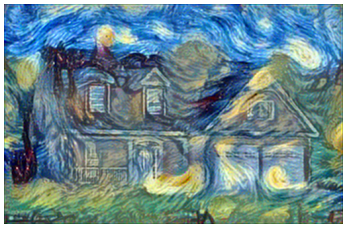


Figure 28: content image initialization

Figure 29: blur content image initialization

Figure 30: result images from log loss plot using denoise method

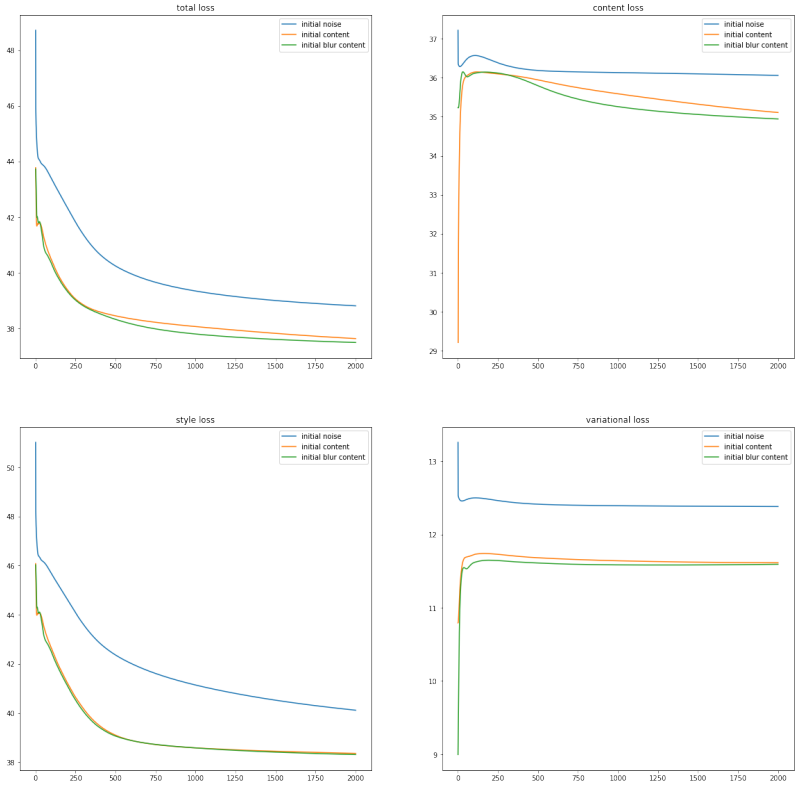


Figure 31: log loss plot without using denoise method

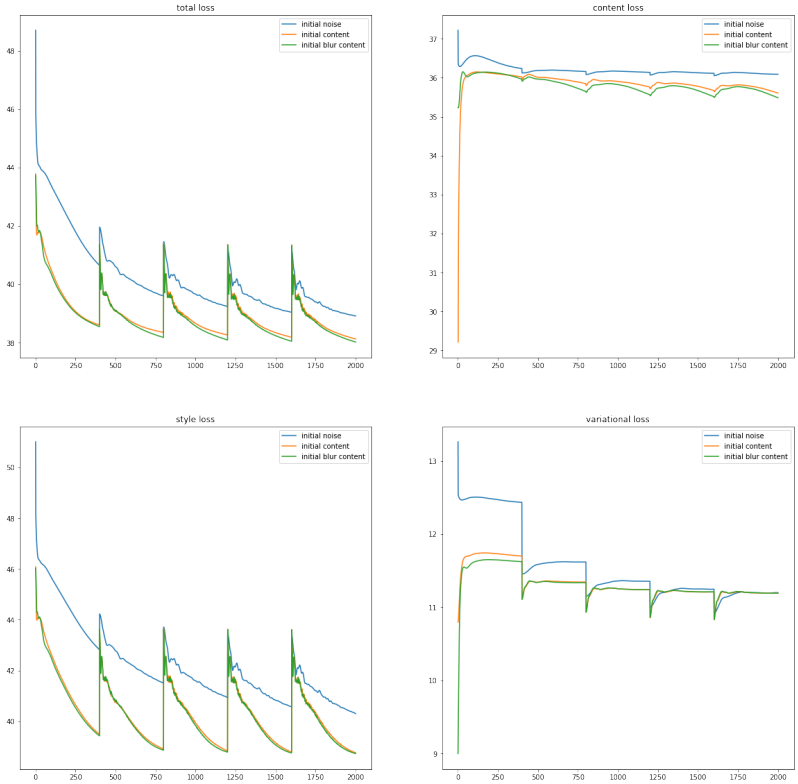
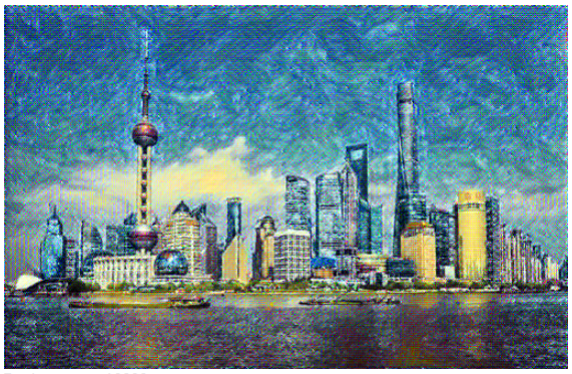


Figure 32: log loss plot using denoise method

3.6 Mobilenet Results



```
Train step: 10000
Total time: 102.5
```

Figure 33: result image from pretrained Mobilenet Model, using $\alpha = 1,\ \beta = 100$.

4 Discussion

We are using the Google GPU to generate the images from the content images and the style images. If we extract one Convolutional layer from content image and five Convolutional layer from style image via VGG-19 pretrained model, it takes 932 seconds for 10000 training steps. From pretrained Mobilenet model, we use also use one layer as content representation and three layers as style representation. It takes 102.5 seconds for 10000 training steps.

4.1 Tuning Parameters

4.1.1 Initialization

We studied three types of initialization: 1. white noise; 2. content image; 3. blurred content image. Initially, from Figure 31, the second initialization has zero content loss at zero step and the third initialization has zero variational loss and small content loss at zero step. The third initialization setting both have a less bias toward the content compared to the second initialization setting. Also, shown in Figure 31 and Figure 32, the blurred content image initialization setting has the fastest speed of convergence. If the style image is less abstract saying more realistic, then the content initialization setting will have the best performance. Although the white noise initialization has no bias toward the content image, it takes too many step to converge to a meaningful image.

4.1.2 Convolutional Layer Selection

In [1] Section 3.1, the author uses 'block4 conv2' as the content representation layer and 'block1 conv1', 'block2 conv1', 'block3 conv1','block4 conv1','block5 conv1' as the style representation layers. According to our observation, the 'block1 conv2' could between represent the color and the structures in the content image compared to the 'block4 conv2' as shown on the Figure 14. Also, we found out that the 'block1 conv1' and 'block5 conv1' does not convey style information so we drop them to increase our training speed.

We are using the 'block1 conv2' as the content representation layer and 'block2 conv1', 'block3 conv1','block4 conv1" as the style representation layers. We set the weights to the content representation layer as 10^4 and the weights to the style representation layer as 10^{-3} to make the convergence fast.

4.1.3 Denoise Method Implementation

Our denoise method successfully remove the random noise points generated by the gradient. It makes the output image smoother and looks better.

4.1.4 Value of Loss Weights ($\alpha,\ \beta$)

In the original work [1], the authors mentioned the trade off between the weight of the content loss and the weight of the style loss. Depending on the expectation of the output image, a bias toward the content image could preserved the shape and color of the original content image while it also leave some room for the the style to be make. In general, one can adjust the ratio between α and β between content and style to create different types of images. There is no unique or right answer.

4.1.5 Optimizer

We used the Adam optimizer and it can be replaced by L-BFGS-B mentioned in [1]. We could also try different learning rate or different optimizer for further studying.

4.2 Mobilenet

From pretrained Mobilenet model we use 'conv dw 2 relu' as content representation layer and 'conv pw 1 relu', 'conv dw 5 relu', and 'conv dw 7 relu' as style representation, we could get a similiar but slight different result image compared to the pretrained VGG-19 model. Apparently, the training speed of using Mobilenet is much faster than using VGG-19.

References

[1] L. A. Gatys, A. S. Ecker and M. Bethge,(2016) *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR): Image Style Transfer Using Convolutional Neural Networks*, Las Vegas, NV, pp. 2414-2423, doi: 10.1109/CVPR.2016.265.

[2] Leon A. Gatys, Alexander S. Ecker, Matthias Bethge, Aaron Hertzmann, Eli Shechtman. (2017) *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR): Controlling Perceptual Factors in Neural Style Transfer*, pp. 3985-3993, University of Tubingen

[3] Justin Johnson, Alexandre Alahi and Li Fei-Fei. (2016) *European Conference on Computer Vision: Perceptual Losses for Real-Time Style Transfer and Super-Resolution*

[4] https://www.tensorflow.org/tutorials/generative/style_transfer

[5] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto and Hartwig. (2017) *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*

Appendix

Code Github Link

https://github.com/bingguJ/5242_project