GR5242 Final Project: Neural Style Transfer

Sung In Cho, sc4393
Department of Statistics, Columbia University
sc4393@columbia.edu

Abstract

In this project, we will use deep learning technique to compose images in a style of another image. This method is called *Neural Style Transfer*. For this project, I mainly referred to the paper, [1]`Image Style Transfer Using Convolutional Neural Networks (Gatys et al., 2016)`. Neural style transfer blends two images, a content image and a style reference image, to make an output image that looks like the content image painted in the style of the style reference image. An optimization technique is used to the output image to match the content statistics of the content image and the style statistics of the style reference image. These statistics are extracted from the above images using a convolutional network. We will use VGG19 network in this project. From this project, we want to learn the important steps of Neural Style Transfer and find out the factors that affect an output image the most in a training process.

1 Introduction

Deep learning using Neural Network has already surpassed human level performance in tasks such as object recognition and detection. However, it had been lagging far behind in tasks like generating artistic artifacts. With the recent advancement of computer hardware, as well as the proliferation of deep learning, deep learning can now be used to create art. One of the most well-known techniques in the field is Neural Style Transfer.

Neural Style Transfer employs a pretrained convolution neural network (CNN) to transfer styles from a given image to another. This is done by defining a loss function that tries to minimize the differences between a content image, a style image, and a generated image.

In this project, we will follow the steps in the paper of [1] `Gatys et al` to learn how to 1) extract the features of the style reference images and the content images, 2) separate style and content representations of those images, and 3) merge those features using loss functions to generate a well blended image.

(a) Content image (b) Style image (c) Generated image

2 Feature Extraction

2.1 VGG19 network

I used VGG19 network, a pretrained image classification network, to get the content and style features of the image. The network's first few layer activations represent low-level features (like edges and textures). The final few layers represent higher-level features (object parts; like wheels or eyes). Therefore, I extracted a content feature from a final few layers to extract the semantic information of the content image. To extract style features from an image, we use *Gram matrix*. Style information of the image should be unrelated to semantic information of the image. Therefore, we use *Gram matrix* which shows the correlation between feature maps of each layers. In the case of style feature extraction, we must check the correlation of multiple layers. By this method, we can get stationary information of multiple layers and get a restructured image using multiple features from these layers.

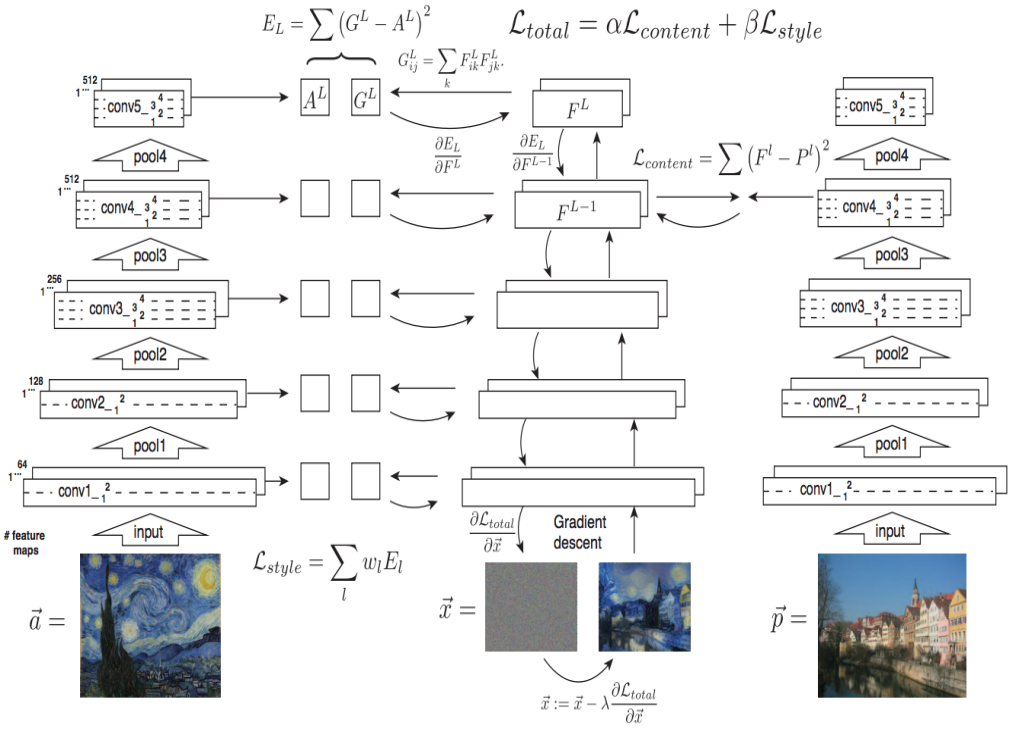

Figure 1: Style transfer algorithm

2.2 Gram Matrix

The style of an image can be described by the means and correlations across the different feature maps. Calculate a Gram matrix that includes this information by taking the outer product of the feature vector with itself at each location and averaging that outer product over all locations. This Gram matrix can be calculated for a particular layer, l, as:

$$G^l_{cd} = \frac{\sum_{ij} F^l_{ijc}(x) F^l_{ijd}(x)}{IJ}$$

3 Loss Function and Gradient Descent

To get an output which has content features and a style features that are most close to the input image features, we need to calculate *content loss* and *style loss*.

3.1 Content Loss

Content loss is simple to calculate. I passed the network both the output image and the content image. This will return the intermediate layer outputs from our model. Then we simply take the euclidean distance between the two intermediate representations of those images. Therefore, Content loss is a function that describes the distance of content from our output image (x) and our content image (p). Let C_{nn} be a pretrained deep convolutional neural network (in this project I used VGG19). Let X be any image, then $C_{nn}(X)$ is the network fed by X. Let $F^l_{ij}(x) \in C_{nn}(x)$ and $P^l_{ij}(p) \in C_{nn}(p)$ describe the respective intermediate feature representation of the network with inputs x and p at layer l. Then we describe the content distance (loss) as

$$L_{content}(p, x) = \sum_{i,j}(F^l_{ij}(x) - P^l_{ij}(p))^2$$

3.2 Style Loss

Computing style loss is a bit more complicated, but follows the same principle. However, instead of comparing the raw intermediate outputs, I compared the Gram Matrices of the two outputs. Mathematically, we describe the style loss of the output image (x) and the style image (a) as the distance between the style representation (Gram Matrix) of these images. We describe the style representation of an image as a correlation between different filter responses given by the Gram matrix G^l, where G^l_{ij} is the inner product between the vectorized feature map i and j in layer l. We can see that G^l_{ij} generated over the feature map for a given image represents the correlation between feature maps i and j. The contribution of each layer to the style loss is described by

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

where G^l_{ij} and A^l_{ij} are the respective style representation in layer l of x and a. N_l describes the number of feature maps at layer l and M_l describes $height \times width$ of feature maps at layer l. Therefore, the total style loss across each layers is

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

where w_l is weighting factors of the layer to the total style loss.

3.3 Total Loss

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

where α and β are the weights of each loss functions.

3.4 Gradient Descent

To see which output minimizes the total loss, I used gradient descent as an optimizer. We iteratively update our output image such that minimizes our loss.

4 Total Variation Loss

One downside to the above implementation is that it produces a lot of high frequency artifacts (noise). We can decrease these artifacts using an explicit regularization term on the high frequency components of the image. This technique is called the *total variation loss*. High frequency component is basically an edge-detector. As more high frequency artifacts are produced, more edges(lines) will appear in the image. Below figure shows how the high frequency components have increased after Neural Image Transfer.

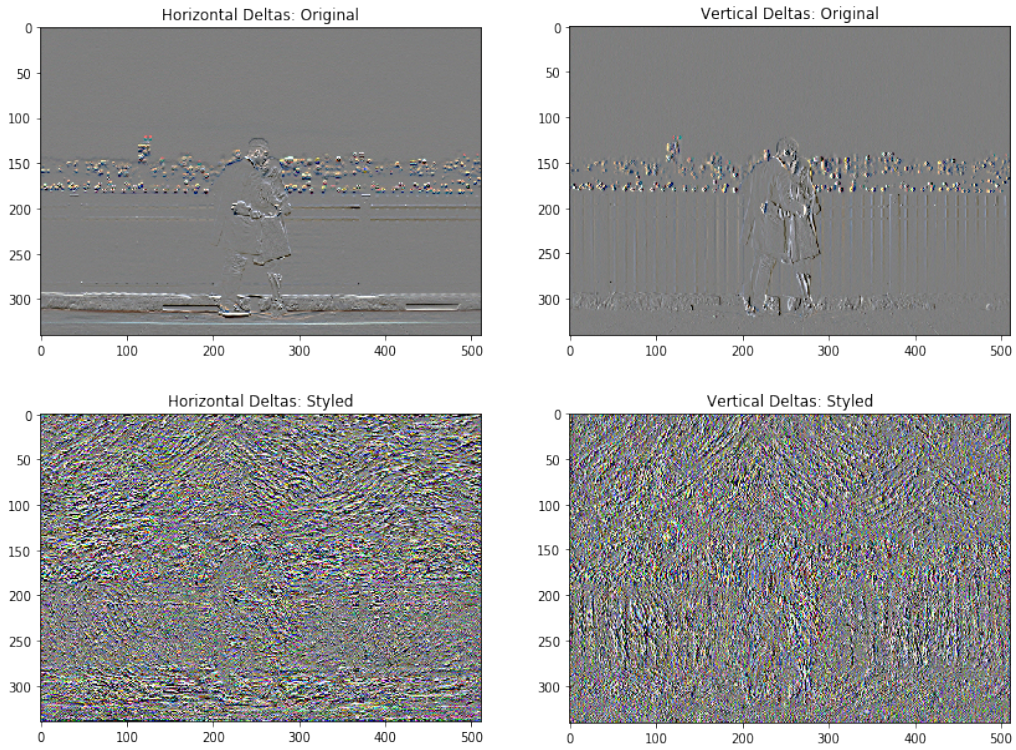

Figure 2: Increase in high frequency artifacts

5 Results

Below figures show style images, content images, and generated images. The generated images are blended well. They possess semantic information of content images and have features of style images.

(a) The Starry Night (Van Gogh) (b) The Scream (Edvard Munch)

Figure 3: Style images

Figure 4: Content image 1

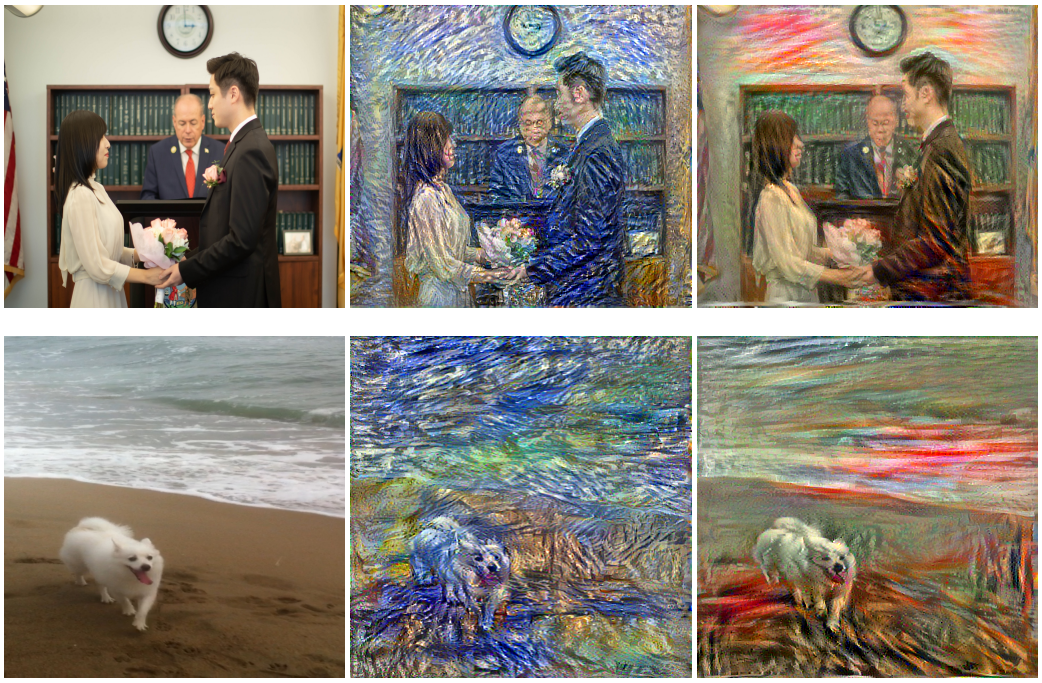

Figure 5: Content image 2, Content image 3

6 Factors that may affect the output image

6.1 Different Random Seed

In this section, we will see how the image changes if we generate the images with different random seeds. Since the change is more likely to be dramatic as the training EPOCHES decrease, we will see how random seed affects the output image with different EPOCHES.

(a) EPOCHES=10, random seed(1) (b) EPOCHES=5, random seed(1) (c) EPOCHES=1, random seed(1)

(d) EPOCHES=10, random seed(2) (e) EPOCHES=5, random seed(2) (f) EPOCHES=1, random seed(2)

Figure 6: Output images for different random seeds

Generated images having higher EPOCHES for the training procedure are very similar to each other regardless of random seed. However, it is much easier to find the differences between the generated images with different random seeds when we put lower EPOCHES for the training procedure.

6.2 Training Iterations

We will now see how number of training iteration affects the output images.

(a) 100 Training Iteration (b) 75 Training Iteration

(c) 50 Training Iteration (d) 25 Training Iteration

Figure 7: Output images for different training iterations

As we increase the training iteration, the image becomes more like a real painting of Van Gogh. The content image and style reference image get blended better as we increase the training iteration. The image with lower training iteration looks more disordered.

6.3 Relative Weight of Content and Style Loss in the Loss function

Image content and style cannot be completely disentangled. When synthesising an image that combines the content of one image with the style of another, there usually does not exist an image that perfectly matches both constraints at the same time. However, since the loss function we minimize during image synthesis is a linear combination between the loss functions for content and style respectively, we can smoothly regulate the emphasis on either reconstructing the content or the style. Below figure shows how the weight of content and style loss regulates the output.

(a) Style:10^{-3}/ Content:10^{5} (b) Style:10^{-2}/ Content:10^{4} (c) Style:10^{-1}/ Content:10^{3}

Figure 8: Output images for different weight of content and style loss in loss function

A stronger emphasis on style will result in images that match the appearance of the artwork more but show less of the photograph's content. Therefore, for a specific pair of content and style images one can simply adjust the trade-off between content and style to create visually appealing images.

Appendix: Unpaired Image-to-Image Translation

A-1 Introduction

The style transfer can also be done without any paired image, with just the target image and the source of the style. For example, transferring a style of painting of Monet like a real photograph. For this advanced project, I referred to the paper, [2]`Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks (Zhu et al., 2017).`

This paper proposes a method that can capture the characteristics of one image domain and transfer these characteristics into another image domain without any paired training examples. It is using conditional GAN (CycleGAN) to demonstrate unpaired image to image translation as it uses a cycle consistency loss to enable training without the need for paired data.

Style Transfer using CycleGAN can be done without GPU. However, it takes a tremendous amount of time to train and obtain the final result image without a help of GPU. Since my computer doesn't have GPU, I will only follow a basic few steps of the paper, which includes training a model and trying it on a single image.

A-2 Preprocessing

Unlike style transfer using VGG19 network, this process requires a training data set as this method does not require any paired image to train a source of the style.

Before training, I will apply random jittering and mirroring to a training data set. These are image augmentation techniques that avoids overfitting. This is similar to what was done in pix2pix. In random jittering, the image is resized to 286×286 and then randomly cropped to 256×256. In random mirroring, the image is randomly flipped horizontally, i.e left to right.

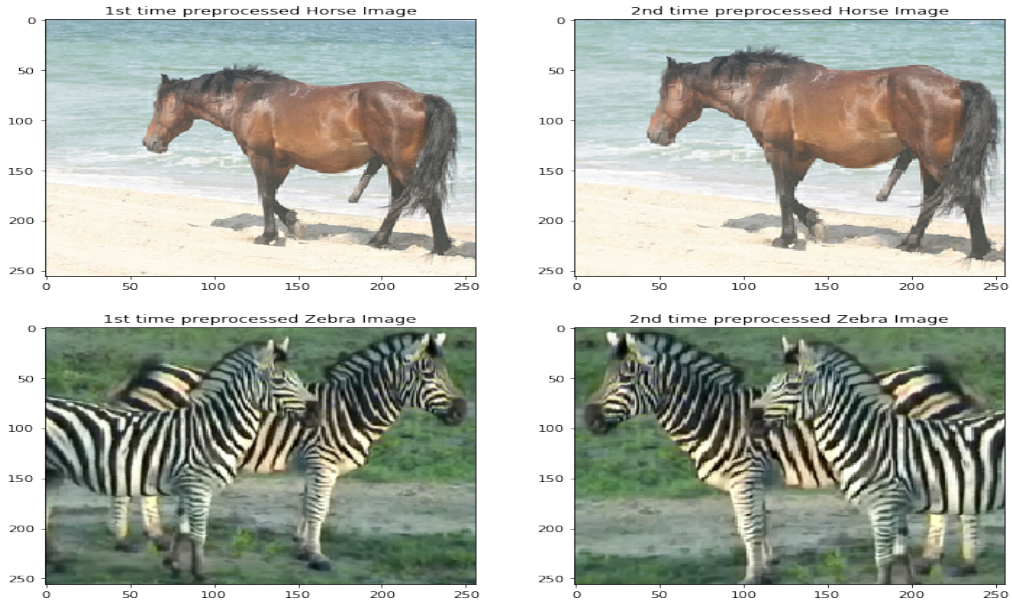

Figure 9: Changes in images after preprocessing

A-2 Generator and Discriminator

Two generators (G and F) and two discriminators (D_X and D_Y) are trained for CycleGAN.

- Generator G learns to transform image X to image Y. $(G : X \rightarrow Y)$
- Generator F learns to transform image Y to image X. $(F : Y \rightarrow X)$
- Discriminator D_X learns to differentiate between image X and generated image $X = F(Y)$.
- Discriminator D_Y learns to differentiate between image Y and generated image $Y = G(X)$.

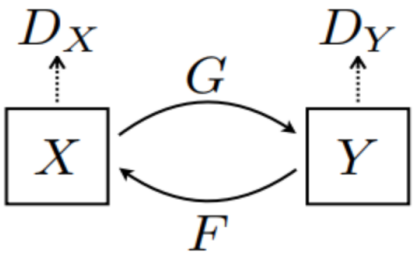

Figure 10: Two mapping functions (generators) and two discriminators

A-3 Loss Function

Our objective loss function contains three types of terms: 1) *adversarial losses* for matching the distribution of generated images to the data distribution in the target domain, 2) *cycle consistency losses* to prevent the learned mappings G and F from contradicting each other, and 3) *identity loss* to measure how close $G(Y)$ and $F(X)$ are to Y and X.

1) Adversarial Loss

We apply adversarial losses to both mapping functions, G and F.

$$L_{GAN}(G, D_Y, X, Y) = \mathbb{E}_y[\log D_Y(y)] + \mathbb{E}_x[\log(1 - D_Y(G(x)))]$$

where G tries to generate images $G(x)$ that look similar to images from domain Y , while D_Y aims to distinguish between translated samples $G(x)$ and real samples y. G aims to minimize this objective against an adversary D that tries to maximize it.

2) Cycle Consistency Loss

Cycle consistency means how much the result is close to the original input. For instance, if we translate a sentence from English to French and then translate back from French to English, the final translated sentence should be the same as the original sentence.

• Calculating cycle consistency loss:

1. Transform image X to generated image $\hat{Y}$ using generator G.
2. Transform image $\hat{Y}$ to generated image $\hat{X}$ using generator F.
3. Calculate mean absolute error between X and $\hat{X}$.

• Forward cycle consistency loss:

$$X \rightarrow G(X) \rightarrow F(G(X)) \sim \hat{X}$$

• Backward cycle consistency loss:

$$Y \rightarrow F(Y) \rightarrow G(F(Y)) \sim \hat{Y}$$

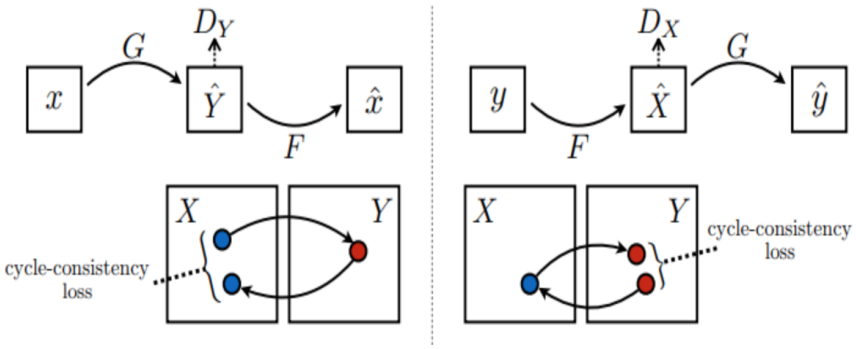

Figure 11: Cycle consistency loss

3) Identity Loss

Since generator G translates images to image Y, if we feed image Y to generator G, we would expect to get image something close to image Y (same for X and generator F).

• Calculating identity loss:

$$L_{Identity} = |G(Y) - Y| + |F(X) - X|$$

I used *Adam* optimizer for this project and applied gradient descent using the optimizer to minimize the total loss function.

A-4 Result

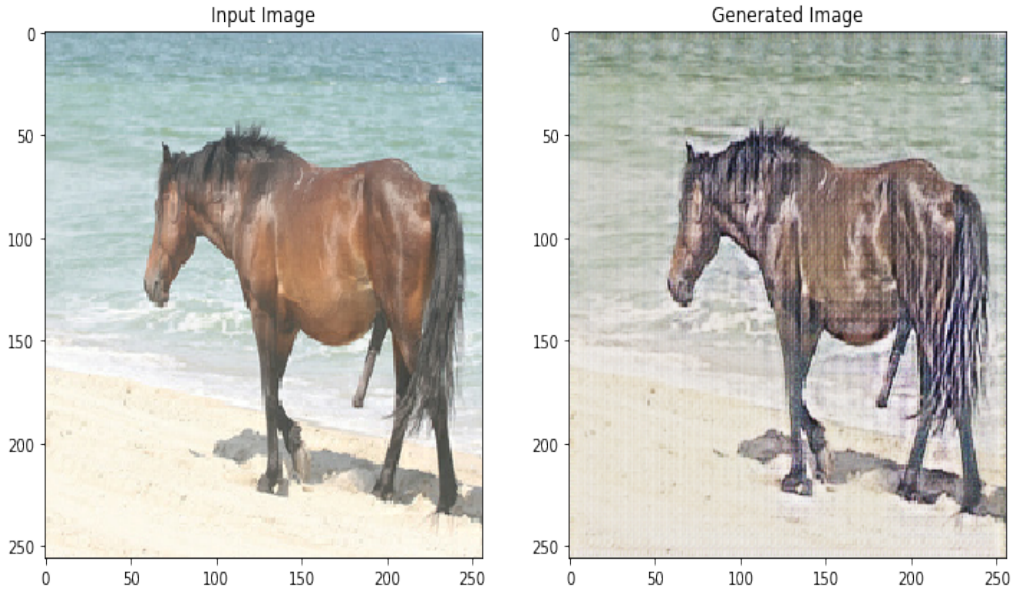

Figure 12: Training Result: Horse to Zebra

Next Step

In this project, due to the technical limit, I trained the model for a very small number of epochs. We can see that a form of a horse in the input image did not change perfectly into a form of a zebra in the generated image. We can improve the result, if we train the model for a larger number of epochs.

References

[1] Image Style Transfer Using Convolutional Neural Networks (Gatys et al., 2016)

[2] Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks (Zhu et al., 2017)