# ECE 590-07 Final Project Report Template

**Bhrij Patel**
Computer Science
Duke University
bhrij.patel@duke.edu

**Jose Balcazar Caldas**
Computer Science
Duke University
jose.balcazar.caldas@duke.edu

## Abstract

NST is very commonly used in practice of generating new artworks that are deeply influenced by the style of famous artists as we demonstrate in this project. NST is found incredibly useful in the process of generating non-photorealistic renderings [3], and applications of NST can be seen in video processing. NST can also be used in the data augmentation process by randomizing an image's style so to better predict content related classifications. In this project we replicate the Neural Style Transfer (NST) from the two papers by Gatys et. al [1] and Ulyanov and et. al [2] and compare their performances. In order to create a new image we consider two minimizing two separable features of two different images: one containing the content and the other containing the style.

## 1 Background

Neural Style Transfer is a specific application of transfer learning where we generate a new image using two images, one for its content and the other for its style. The new image will then look like the content image with the artistic style of the stylized image. This algorithm is important in understanding how humans create art and also can quantify similarities of styling between different art periods.

## 2 Methods

The goal of this experiment is to generate a Deep Neural Network that is able to create a new image that will contain the content of one image and the style of another. We have implemented two algorithms that generate the neural style transfer, one of the original paper by Gatys et al [1] and the other by Ulyanova et. al[2]. Both attempt to minimize the loss of between the original image and the content image, and then update the new image style features by minimizing the loss between the new image and style image's style layers'.

### 2.1 Neural Algorithm of Artistic Style

Our implementation of Gatys et. al [1]'s paper includes using a pretrained VGG - 19 network that will be pre-trained using the weights of those found in imageNet. ImageNet weights are appropriate here since they have been trained to classify features of an image. In order to acquire the content and the style, we consider each of them as a separable features.

A layer with n distinct filters has n feature maps each of size m , where m is the height multiplied by the width of the feature map. So the responses in a layer l can be stored in a matrix $F_l \in R^{nxm}$ where $F_{j,j}$ is the activation of the ith filter at position j in layer l.

The pre-trained VGG model will take three inputs: a content image, a white noise image, and a style image. As discussed the content image will contain the detail of new image while the style image will

be the reference for how to stylize the new image. The white noise image, is a randomly generated white noise image that will be used during training as the changing image. For each epoch of training we will minimize the what we later discuss as the content loss and style loss between the changing white noise image and the respective reference images.

During training, we conduct gradient descent on the generated image of the network to match the original image's features to that of the newly generated image. First let us discuss how we will be calculating loss.

### 2.2 Loss Functions

The Gatys et. al [1] and Ulyanova et. al [2] both calculate loss style loss and content loss similarly. Let $F^l(x)$ be the $N$ feature maps outputted by layer $l$ of a network given input image $x$ with $F_i^l(x)$ representing the $i$th feature map.

Content loss is calculating very similarly in the way that gradient descent typically considers loss. Let content loss, $L_c$, be between two images $x$ and $\hat{x}$ and let $T$ be the set of layers from the network selected. Thus:

$$L_c = \sum_{l \in C} \sum_{i}^{N_l} ||F_i^l(x) - F_i^l(\hat{x})||_2^2$$

Thus for a single layer $l$ we try to minimize the set of features using Mean Square Loss. Selecting the layer is up to the experiment designer but we will use the fourth convolution layer as the layer to conduct content loss.

Calculating the style loss is a two step procedure, calculating the Gram matrix and the minimizing the mean square loss of the feature gram matrices of the style image and the new image. First, let us introduce the concept of the Gram Matrix. The gram matrix is the Hermitian matrix of a inner products whose entries are given by matrix $X$. Put simply a gram matrix, $G$ is the following: $G = XX^T$.

The Gram Matrix is computed as such: Defining $G^l(x)$ as the Gram Matrix at layer $l$ for input image $x$, element $G_{ij}^l$ is the inner product between the $i$th and $j$th feature map:

$$G_{ij}^l(x) = \langle F_i^l(x), F_j^l(x) \rangle$$

.

Letting $R$ be the set of layers from the network selected for the style loss, $L_s$, between two images $x$ and $\hat{x}$:

$$L_s = \sum_{l \in R} ||G^l(x) - G^l(\hat{x})||_2^2$$

For both of our architectures, the network in question is the descriptor network, and in our case as mentioned before, the pretrain VGG-19 model.

Thus, during the training of our VGG-19, when training $Q$, given an style image $s$ and content image $y$ with random noise $z$, let $\hat{x} = Q(y, z)$. Thus:

$$L = L_c(\hat{x}, y) + \alpha L_s(\hat{x}, s)$$

where $\alpha$ is a scalar to control the weighting between the two losses. To be clear, the layers used to calculate loss are from $D$ but it will be $Q$ that will be updated with back propagation.

### 2.3 VGG Model Training

Now to replicate Gatys et. al [1], we generated a random white noise image, then chose the content and style images for model to reference. Moving forward we will refer the white noise image as the updating image as it is the image we are conducting training on. Using a 300 epoch time period we inputted these three images. These images were read in, resized to be the size of 256 x 256 pixels and then transformed to be tensor objects. Like Gatys et. al [1] we used a L-BFGS optimizer in the forward pass. For each epoch we extracted the feature map of the content image's fourth convolutional layer and the feature map of the updating image's fourth convolutional layer. We then

find the content loss between these two feature maps. Once that is complete we then calculate the gram matrices for five convolutional layers of the updating image, convolutional layers 1,2,3,4,5 and that of the style images. Let a pair of gram matrix refer to the gram matrix of one layer for one image and that of the same layer of the other image. For each respective pair of gram matrices, we calculate the style loss. Thus total style loss is the sume of the five pair of gram matrices style losses.

Once content loss and style loss have been calculated, we weight the style loss constant, appropriately to enhance styling at $1x10^10$. Now having your total loss you back propagate and proceed with training.

Note: Pytorch has a tutorial that replicates the Gatys et. al [1] paper for additional reference. We referenced this tutorial ourselves to find how to calculate the gram matrix as well as compare our implementation to theirs. We also referenced a github repository that did a adoption of the Gatys et al[1] paper but used a VGG-16 model with a Adam optimizer [4].

### 2.4 Image Generation then Training

Here is our implementation of Ulya:

Architecture consists of a generator, $Q$, and a descriptor $D$. The descriptor $D$ is a pretrained network, and in our case, VGG-19 trained on ImageNet. For implementing the architecture of $Q$, we closely followed [**?**].

Image $y$ is of size $3 \times H \times W$, where 3 is the number of channels and $H$ and $W$ represent the dimensions of each channel. An image pyramid with 6 layers is created from $y$. The first layer, or $L_1$ is the original image and $L_2$ is a downsampled version of $y$ to half the size. Generalizing, the size of $L_i$ is $3 \times \frac{H}{2^{i-1}} \times \frac{W}{2^{i-1}}$. We concatenate another 3 channels of random noise to each layer. Let $z_i$ be the random noise concatenated to layer $L_i$. The size of of $z_i$ is matches the size of $L_i$ and each element in $z_i$ is i.i.d of a uniform distribution between 0 and 1.

We used average pooling layers to create the image pyramid of $y$. The size of the filters and stride used on $y$ to create $L_i$ are both $2^{i-1}$. Each layer of the image pyramid is passed into a convolution block that outputs 8 feature maps. A convolutional block consists of 3 blocks of convolutons. The first two blocks each consist of a convolution of a $3 \times 3$ kernel and the input is padded to preserve shape. It is then followed by batch normalization layer and Leaky Relu of 0.1. The third block is similar except there is no padding and the kernel size is 1.

The output of the convolution block is then concantenated to back of the $L_{i-1}$'s feature map output as additional feature maps. During concatenation of features maps of layer $L_{i-1}$ and $Li$ for $1 \leq i \leq 5$, we upsampled the features maps from $L_i$ using nearest neighbor interpolation and added them as additional feature channels to the output features of $L_{i-1}$. So for example, the $8$ $8 \times 8$ feature maps of $L_6$ outputted from the convolution block were upsampled to $16 \times 16$ and then concatenated behind the $8$ $16 \times 16$ map of $L_5$ outputted from $L_5$'s convolution block, creating $16$ $16 \times 16$ feature maps. This concatentation is then passed through another convolution block that preserves the number of total channels (e.g. 16 channels in, 16 feature maps out) and then upsampled again and attached to the output of $L_4$. In the end, the last concatenation created $48$ $256 \times 256$ channels. This is then passed into one final block with a kernel size of 1 and 3 feature maps as the output to be then inputted into the pretrained descriptor network.

Please refer to the Implementation Details for a visualization provided by [**?**].

## 3 Experiments and Results

We trained our generator $Q$ using 18 images from the MSCOCO 2014 dataset. As previously mentioned, we used a VGG network previously trained on the ImageNet dataset. All input images were resized to $256 \times 256$.

### 3.1 Reproducing Good and Bad Cases

In this section, we simply recreate some examples of good and bad cases from both papers. We found that both papers had a varying degree of success dependent on the styling. Much like the practice of art, we found that there is no one hyper-parameter that works best for every style. In figure 1, we see an example of a good and bad case of using the convolution layer 2 as the reference content later and every one of the 5 layers as the style references. It is clear that Van Gogh's starry night is embedded stylistically in the Yosemite park image without removing any artifacts crucial in identifying the content as belonging to Yosemite. Composition VII by Wassily Kandinsky, on the right, when used as the style image, causes the network to output an image that is very dissimilar to the Yosemite park image.

Figure 1: First Row (Style Images: Starry Night (Middle), Composition VII (Right). Second row: Original Image (Left), Starry Night Stylized Output(Middle), Composition VII Stylized Output(Right)

Similarly to the reproduction of the Gatys et. al [1] model, the Ulyanova et. al [2] model also had good and bad cases of adopting artistic styles to the content. In figure 2, using Van Gogh's Starry night on Che Guevara's profile we see his profile embedded in the starry night background. When using am image of the actual night sky as the style image, we noticed that Che Guevara's face disappears into the night. We concluded that preserving the content, using a mostly dark blue image proved to be challenging for the model.

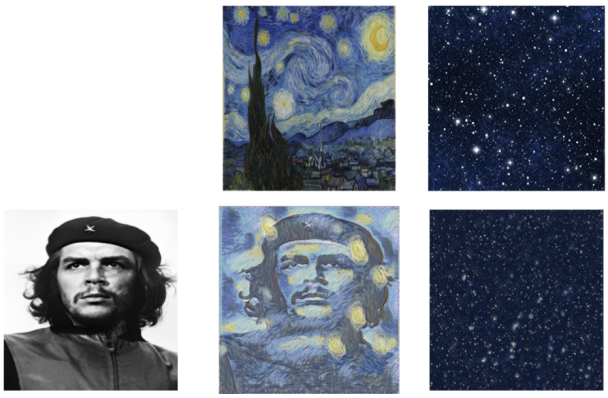

Figure 2: First Row (Style Images: Starry Night (Middle), Night Sky With Stars (Right). Second row: Original Image (Left), Starry Night Stylized Output(Middle), Night Sky With Star Stylized Output(Right)

### 3.2 Pairs of Style/Content Images of Differing Complexities

In this experiment, we attempted to the affect of complexity of the style and content images on the generated image. We found that for both papers, the networks were able to reproduce a good image

with the complex content complex style images. However, with the simple content, simple style combination both networks produced images that were vaguely similar. The Gatys et. al [1] model provided the images in figure 3. Here we see how complexity case study performed well but the the styling of the yellow grey image was to simple to adopt.

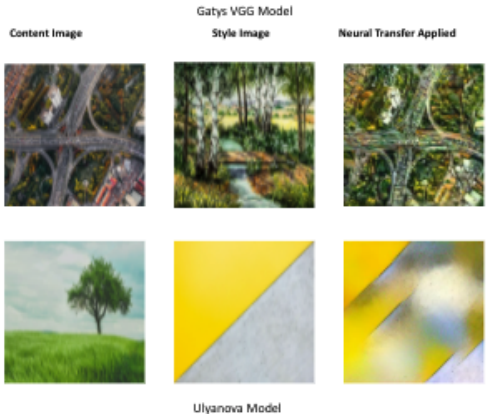


Figure 3: For the Gatys et. al paper: Complex content photo(Top Left), Complex style (Top Middle), Complex NST Output (Top Right). Simple content(Bottom Left), Simple style (Bottom Middle), NST Transfer output (Bottom Right)

In the Ulyanova et.al [2] model we see a very similar behavior. The complexity case provided an output that is clearly the original content stylized. The simple case study, however, resulted in odd outputs, where content attributes like the trees were maintained but the style was not properly transferred.


Figure 4: For the Ulyanova et. al paper: Complex content photo (Top Left), Complex style (Top Middle), Complex NST Output (Top Right). Simple content (Bottom Left), Simple style (Bottom Middle), NST Transfer output (Bottom Right)

When comparing the visual outputs of both model, we see that the Gatys et. al[1] model was able to keep some finer details of the content then the Ulyanova model.

### 3.3 Reproducing Analysis from Each Paper

In Gatys et. al [1]'s we attempted to replicate the first figure which is the collection of style transformations to an image. We will now demonstrate for style weight $1X10^{10}$ and content weight equivalent to 1 our images. In figure 4 we have three images. The orginal image (left) and two stylized images( middle and right). We wanted to reproduce the Gatys et. al [1] analysis that established the choice of the convolutional layer for content. The middle image has the second convolutional layer as the content layer and the right image has the fourth convolutional image. As is clear in the

images, the middle images contains a greater detail of the original Yosemite park image than the right image. This is expected as earlier convolutional layer contains the finer detail of an image while later convolutional layers contain more abstract features. Using the fourth layer as the convolutional layer for content really allows the starry night style to be very clear in the image.

Figure 5: Original Image(Left). Convolutional Layer 2(Middle). Convolutional Layer 4(Right).

For Ulyanova et. al [2], the authors wanted to experiment with the balance between style and content during testing. During training there is the hyperparameter $\alpha$ that controls the weighting of the texture loss w.r.t the content loss. However, for testing that is not an option as the model is already trained on a specific $\alpha$. Thus scaled the noise vectors by a factor of $k$ and thus all the elements in the noise vectors effectively became i.i.d from 0 to $k$. Below is the result of their experiment from their paper:

Here is our attempt:

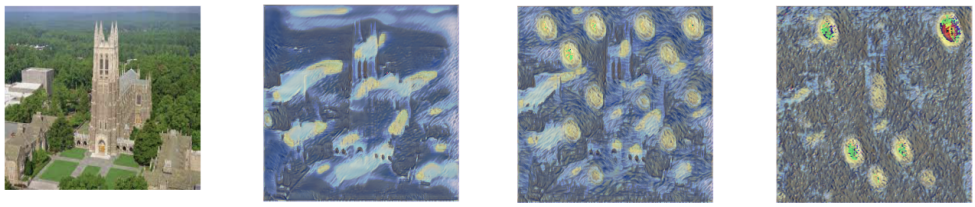


Figure 6: Training a generator on Starry Night, we inputted the image of the Duke Chapel (left) during test time while scaling the noise vectors by the different factors of k

## 4 Conclusions

Neural Transfer Style, as is posited in the Gatys et. al[1] and Uylanova et. al[2], can be successfully replicated to create a new image with the content from one reference image and the style of another. We were able to replicate select analysis conducted in both of the papers, specifically analyzing the effect of the specification of a the content convolutional layer and the effects of the scale $k$ on the noise vectors on the models previously discussed. This analysis supports the robustness of these algorithms in Neural Style Transfer. This has major impacts on the larger computing community, creating new ways to do adversarial training, conduct data augmentation, and also provide an deeper understanding of the ways that human perceive artwork in terms of separability.

## References

[1] Gatys et. al. *Image Style Transfer Using Convolutional Neural Networks*. 2016.

[2] Dmitry Ulyanov, et al. *Texture Networks: Feed-forward Synthesis of Textures and Stylized Images*. 2016. In 2016 IEEE Conference on Computer Vision and Pattern

[3] SwordHolder *https://github.com/SwordHolderSH/neural-style-pytorch*

[4] Bhrij Jose GitHub repository *https://github.com/Bridge00/ece590-neural-style-transfer*

# A Appendix

# B Timeline and task allocation

Please provide your working timeline of the project in this section. If you are working in a group of two students, please also specify how the task is allocated among the two members.

Most of the implementation was done on the schedule of the student, either Jose or Bhrij, for their respective paper. The algorithms took about two weeks to implement.

Jose worked primarily on replicating the Gatys et al paper and Bhrij worked on the Ulyanova et al paper. Each of us implemented the algorithms respectively and then discussed how they differed.

# C Implementation details

For the second pipeline implementation, we used Adam optimizer with learning rate of 0.1 and trained on 600 epochs with batch size 1. Also below is a visualization of the generator architecture described in Section 2.4.

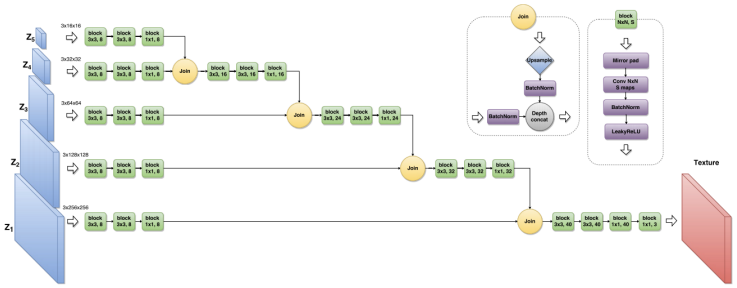


Figure 7: Visualization of the architecture for the generator described in [**?**]. Please note that in this visualization, they refer to their texture synthesis application so they use only 5 layers for the image pyramid and all the layers only consist random noise so the input is 3 channels rather than our 6 channels as described in Section 2.4

# D (Optional) Additional experiment results

## D.1 Obtaining Original Content Image

As a fun experiment, we decided to see if we could somehow output the content image. Using the second approach, we trained the network on some style image $s$ and collection of content images. During test time we use pass in a test image $c$ to the trained network to get a generated image $g$. Next we train another model using the $c$ as the style image and the same collection of content images for the training images. During test time for this second model, we pass in $g$ to get $g'$. Theoretically since some of the content of $c$ is loss to produce $g$, we don't expect that $g'$ to look exactly like $c$ either.

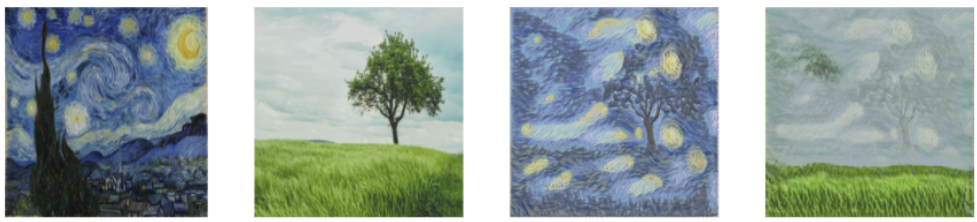

Figure 8: We had one generator trained on style image (left) and then inputted the content image (2nd from left) to generate they stylized image (3rd from left). Finally we trained another network using the content image as the style and then inputted the stylized image to create the image on the far right.

We see that the from using Ulyanov paper, that $g'$ was able to get some of the content back, i.e. the grass. The tree is slightly visible but covered by the lingering texture of the Starry Night.