EECS 445 Final Report: Improvements to *A Neural Algorithm of Artistic Style*

Frost, Bradley
bfrost@umich.edu

Kovacinski, Stephen
kova@umich.edu

Pitt, Kevin
kpitt@umich.edu

Sawicki, Nathan
nsawicki@umich.edu

Simonson, Luke
lukesimo@umich.edu

December 23rd, 2015

1 Abstract

The machine learning and artificial intelligence community has recently directed more effort and attention towards understanding whether the concept of 'style 'can be learned through state of the art machine learning methods.[5] This novel idea presents numerous challenges, with the core challenge being what the definition of style truly is from a machine learning perspective. Various techniques already exist that can be leveraged and modified to make progress in this problem.[4] Our team has implemented a new algorithm that can learn and generalize the style of an iconic artist through processing many of their paintings or the paintings of many different artists. Our technique makes use of the computational power of convolutional neural networks (CNN), along with foundational machine learning algorithms such as K-Nearest Neighbors and nuanced loss functions. In this paper, we outline the changes made to an initially published algorithm that broke ground on this problem, as well as compare the results of our updated algorithm with what was previously achieved.

2 Problem

After experimenting with many combinations of machine learning techniques including CNN's,[2] K-Nearest Neighbors, and Gram Matrices, we have constructed an algorithm in order to generalize the 'style'representation of *multiple images*. The algorithm proceeds to take that learned style and apply it to an arbitrary image in such a way that it will augment the image's appearance to match the learned style, while also maintaining the basic content properties of the original image.

We find motivation in this problem for a number of reasons. First and foremost, understanding 'style 'from a machine learning perspective is an absolute novel challenge to the computer science community. As there does not exist one clear and agreed upon definition of style, through a machine learning lens, our team sees this as an opportunity to make relevant contributions to this constantly-evolving topic. Furthermore, if this was achieved at a high enough level we believe this solution could be extended into multiple realms of real world applications with a positive impact on numerous industries and consumers. We outline a couple of those ideas below:

Our algorithm can be reasonably applied to image generation. With this method's ability to store important information about an image (content and style), one could modify this algorithm to take hundreds of different images of the same type of object and generate a completely new image or augment an old image of a similar object. For example, one could take hundreds of pictures of a car and generate the basis for what the "average" of these cars looks like. Another possible implementation would be taking a pre-generated image of a car, and using our algorithm to change the style of the car. Ideally, changing the style would enhance the fine details of our pre-generated image to look more like the style images of the cars we gathered. This is just one of many opportunities to extend the potential of image generation in order to automate tasks that were previously reserved for "artists".

This technique could also be extended into a variety of other relevant industries. Social media sites and apps could use it as an image filter, allowing the user to alter their images to look like their favorite artist painted it. The technique could be extended to different mediums, such as the potential to analyze and generate new music, allowing the ability to create a song with Mozart's and Beethoven's signature tones.

3 Background

The naive approach of representing 'style'(or texture) was first based on using features learned from a multi-layer network[1] trained from pre-formed style labels[3] that attempted to recognize objects from the image. Such labels include photography techniques such as HDR and Macro, artistic periods such as Baroque and Renaissance, and so on. The learned features would then be used to represent the style as a single feature space. However, this approach was limited because it only attempted to *classify* different style patterns, and then group images by labels rather than determining a tangible representation of style. Consequently, this approach may not have been ideal when trying to compare several images with no distinction of what style is. However, these methods inspired a modernized approach that attempted to better capture a new representation of style in the context of a Convolutional Neural Network (CNN).

The modernized approach, which recently came from a publication known as *A Neural Algorithm to Artistic Style*,[4] served as a focal component to the premise and motivation of our proposed method, which from a high level, expanded on their findings in a multitude of ways. This publication reported a key finding in that a CNN can be used to output a distinct representation for the style in an image, apart from the content that defines that same image. It was stated that style is simply computed as correlations between the various neurons in the network. Because of this finding, it is then possible to overlay the individual style learned from Image A onto another arbitrary Image B while maintaining the content that initially defined image B. This finding served as a core foundation to our proposed method to be described in later sections. The caveat to this approach though is that it only takes one style image and one content image into account, limiting the use of this otherwise groundbreaking discovery.

The results of this most recent approach are striking upon first sight. An example displays the successful overlaying of the style depicted in the *The Starry Night* painting from Vincent van Gogh onto a photo of the "Neckarfront" in Tubingen, Germany. The content of the Neckarfront image can certainly be identified in the augmented image, but further analysis shows a fairly substantial loss in content, overtaken by elements of the painting, with clear distortion in the center buildings and trees on the left side of the image. A significant goal of our project was to diminish any potential

for content loss as we generalized to a larger set of style images.

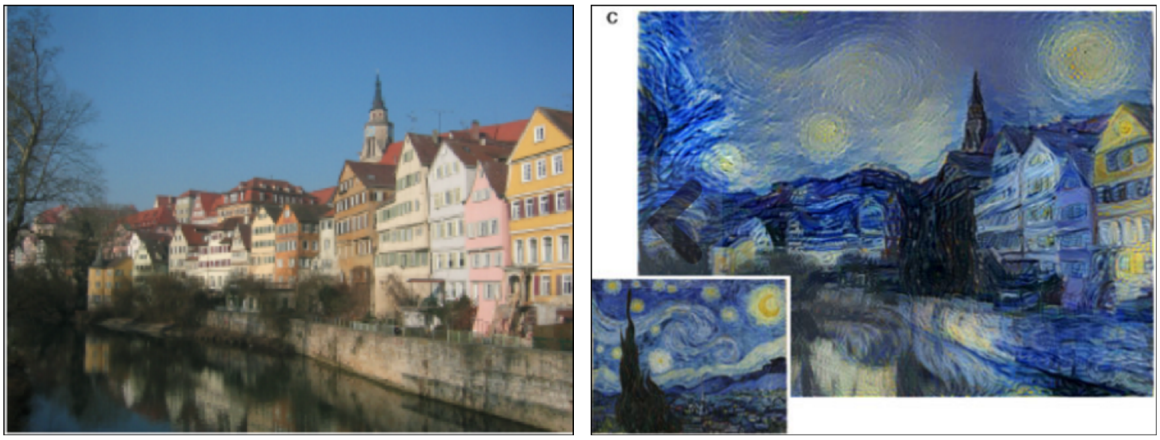

Figure 1: Comparison of content image (left) with applied style (right)

The modernized method generated these results by using a basis VGG-Network, which is known to rival human performance in visual object recognition. The feature space is generated by this VGG-Network, which uses 16 convolutional layers and 5 pooling layers. Upon initialization, the method begins with white noise for the generated image, and performs gradient descent through each layer, which will be described by their determined loss function.

The content loss function is defined below where F^l is the feature representation of the generated image and P^l is the feature representation of the input image at a particular layer l.

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} (F_{ij}^l - P_{ij}^l)^2 \tag{1}$$

For the style representation loss function, they implement a gram matrix equation which calculates the inner product of the feature maps and uses that output to calculate the style component of the total loss at a particular layer as follows:

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

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

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l \tag{4}$$

Finally, they compute the total loss at layer l with respect to the content and style,

$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x}) \tag{5}$$

where β and α are constant weighting factors on the computation. This previous method was key in breaking ground on this novel challenge, and enabled a path for generation of even more promising results.

4 Approach

To improve upon the original paper's results, we discussed several potential approaches and tried each of them individually or in combination with each other. Initially, we attempted to a primitive method using Fourier analysis. This a method previously described by Oppenheim[6] for the signal processing community. However, this technique yields low quality output. This demonstrates the need for a more elegant machine learning approach. Our first task was to mimic the code presented in *A Neural Algorithm of Artistic Style*.[4] Once we had this code running, a very important change was made very quickly. We found that changing the initial generated image from white noise to the content image provided a result that appeared to just "paint" the content with the style image. In other words, our output image begins closer in the solution space to the content image, so our output is more likely to converge to a solution that resembles our content image.

The algorithm was generating reasonable outputs, however, we still needed to implement the handling of multiple inputs. First, we attempted to concatenate several style images together to see if we could get a more generalized style of the artist not specific to one painting. Simply appending all style paintings into one image yielded decent output, but we were unsatisfied with this mundane approach. We then tried computing the average Gram matrix of every style image at each layer of the network. However, we found that this approach did not provide acceptable outputs. In the end, modifying the layer-wise loss function yielded the best results. Our initial loss function penalized over a single Gram matrix. One of the key findings of this paper is that the loss function can be summed over multiple Graham matrices to produce high quality outputs. We consequently implemented a k-Nearest Neighbors algorithm to find the closest style images to our content image. This made for an intuitive method to apply many style images to a content image and repeatedly get reasonable results. Lastly, we attempted to weight the nearest neighbors different in the loss function to see how this would affect the result. We will now go into more detail about each step of the approach.

4.1 Fourier Method

Our exploration of style and content began with an image processing technique invoking the Fourier Transformation. We start with the fourier transform of two images. We created hybrid images using the magnitude of the Fourier Transform of one image, and the phase of another image. The hybrid image below is the inverse Fourier Transform of $X_4(w)$, with the magnitude of the $X_2(w)$ (goat picture), but the phase $X_1(w)$ (Monet painting).

First we take the fourier transform of you two images.

$$X_1(\vec{w}) = |X_1(\vec{w})|\, e^{i(phase(X_1))} \tag{6}$$

$$X_2(\vec{w}) = |X_2(\vec{w})|\, e^{i(phase(X_2))} \tag{7}$$

Then we take the inverse Fourier Transform of our hybrid images X_3 and X_4:

$$X_3(\vec{w}) = |X_1(\vec{w})|\, e^{i(phase(X_2))} \tag{8}$$

$$X_4(\vec{w}) = |X_2(\vec{w})|\, e^{i(phase(X_1))} \tag{9}$$

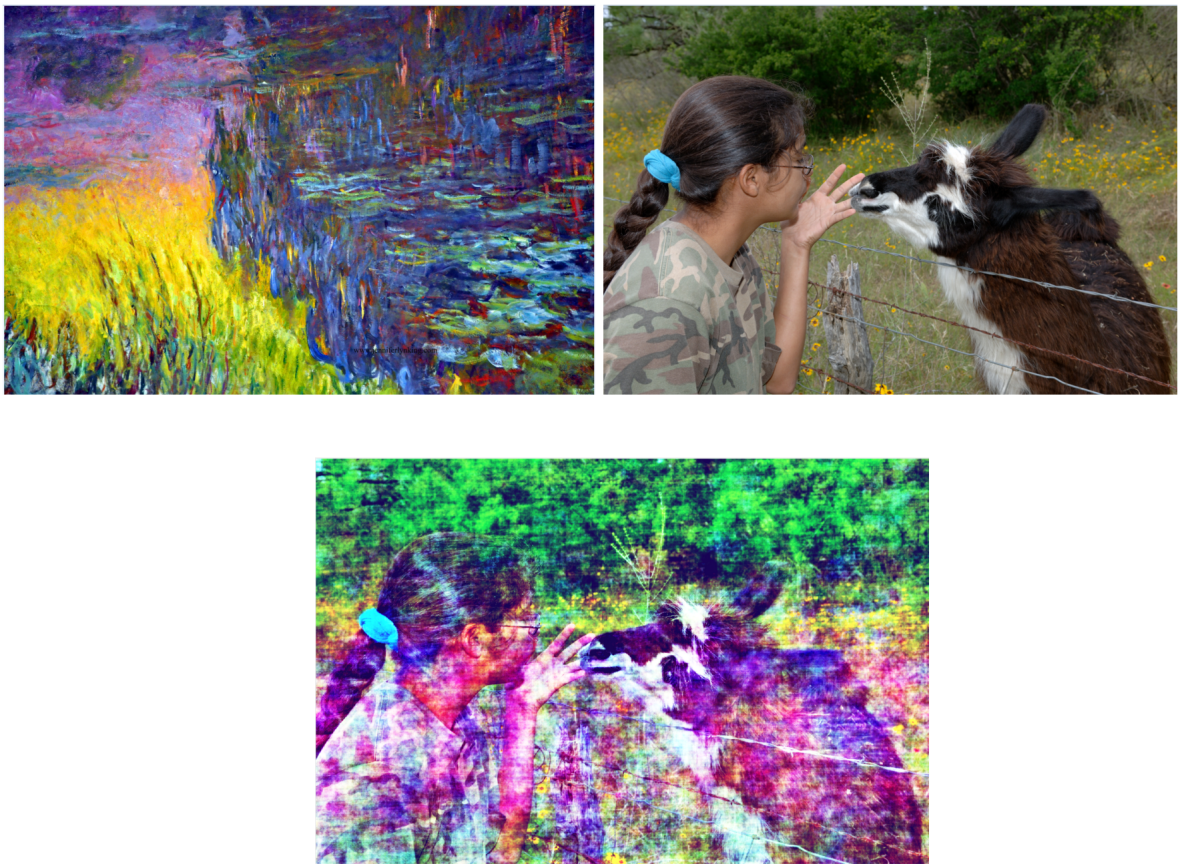

Figure 2: Using the phase of the Monet painting (top left) and the magnitude of the goat picture (top right) to proudce the output (bottom)

4.2 Change Initialization Image

In the original paper, image generation is started from only white noise. In our implementation we start with the input image with added noise, instead of only white noise. We found that by starting with the input image, more distinguishable content is preserved in the output image. We believe that this creates a better output because our goal is to keep as much content from the input image as possible. Starting strictly from the input image causes strange artifacts. So we apply a very small amount of gaussian noise on top of the image, to help create randomness in the final image.

4.3 Concatenation of Training Images

Our first naive attempt with the use of multiple images was to concatenate all of the style images into one single style image. With a small number of input style images this approach worked surprisingly well, outputting very reasonable images the contain style from all the images. This method does not scale very well though. Concatenating images together makes the style image

Figure 3: Comparing starting the generated image from white noise (left) and content + gaussian noise (right)

larger. This then results in loss of detail, because such a large image needs to be scaled down for input into the CNN.

4.4 Average Gram Matrix

Style representation is achieved using the inner product of feature maps generate in the VGG CNN. These feature correlations are given by the Gram matrix, $G^l \in \mathbb{R}_{N^l \times N^l}$ where G^l_{ij} is the inner product between the vectorized feature map i and j in layer l:

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk} \tag{10}$$

So for each style image and layer within the CNN a gram matrix can be generated to represent the style of this painting. Using very basic intuition, we can create a gram matrix that represents the average style of multiple images by just taking the average over all these gram matrices. So our updated equation for gram matrix at layer l looks like:

$$G^l_{ij} = \frac{\sum_a (\sum_k F^{al}_{ik} F^{al}_{jk})}{A} \tag{11}$$

where a represents an image at layer l and A represents the total number of images.

4.5 Modified Loss Function

Our most successful attempt to represent style of multiple images is through the summation of loss for each style painting. So the loss function for style is represented as a difference between the gram matrix of the style painting and the generated output image:

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G^l_{ij} - A^l_{ij})^2 \tag{12}$$

Our implementation creates of vector of A_{ij}^{l} to represent multiple input images A_{ij}^{kl}. So our loss function is now:

$$E_l = \frac{\frac{1}{4N_l^2 M_l^2} \sum_k \sum_{i,j} \left(G_{ij}^{kl} - A_{ij}^{kl} \right)^2}{K} \tag{13}$$

4.6 K-Nearest Neighbors

Through experimentation we have found that the best output images are created when the style image or images have similar content to that of the content image. This similarity can help insure that the style is a reasonable choice for the content image. So when we want to style a picture as a certain artist, we can find the best paintings to use and increase the likelihood of a pleasing output. The way we pick these style images is using a K-Nearest Neighbors technique. The loss function for this is:

$$E = \frac{1}{2} \sum_{i,j} (C_{ij} - S_{ij})^2 \tag{14}$$

where C_{ij} and S_{ij} represents an RGB pixel value for content and style image respectively at the ijth position. In our testing we found the a K value around 4 to produce the best output. When K is increased much more than this the output image to trying to balance too many styles together.

4.7 Spectrum Weighting

After implementing k-Nearest Neighbors to find the best k style images, it made sense to attempt to weight the nearest neighbors unevenly such that we would get more of a certain style into the generated image. This was done by choosing weights such as 0.6 for the first neighbor, 0.3 for the second, and 0.1 for the third, such that the weights add to 1. Ideally, with more computing power, these weights would be treated as hyperparameters that could be cross validated to find the optimum image. Our implementation of this makes sure that whatever weights are chosen are indeed valid.

5 Evaluation

The goal of this project was to achieve meaningful outputs using the style of multiple input images. We will break up this goal in two objectives. First we will look at multiple style images from the same artist, and we hope to see some sort of generalized style for a single artist. Second, we will evaluate how well our algorithm represented a generalized style using images from different artists. Of course art is in the eye of the beholder, but we hope you will agree that this algorithm succeeds in both objectives.

5.1 Multiple Inputs, Same Artist

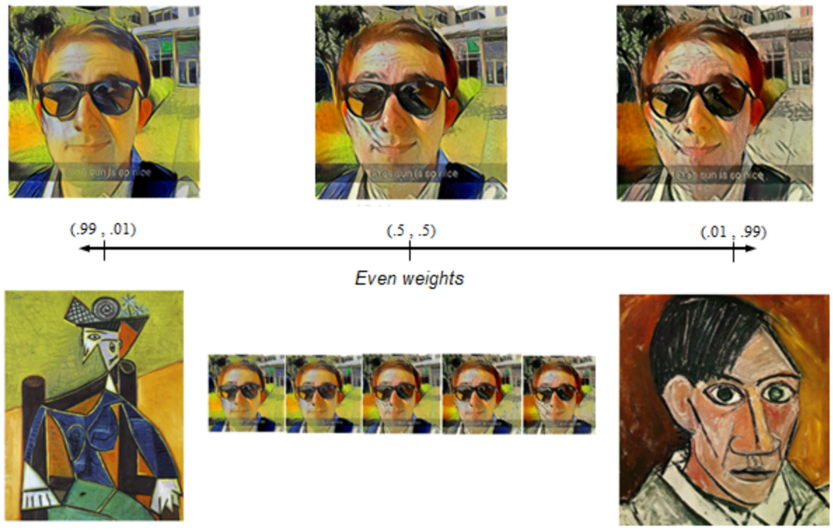

Figure 4: Spectral analysis of weighting furthest neighbor to nearest neighbor using different paintings of a single artist

In Figure 4, you can see the result of weighting two different style images, and the effect it has on the result. We run the algorithm with K = 2, 10 iterations in the optimization step, and using images chosen from a collection of Picasso paintings. We begin by weighting one image very heavily and then slowly iterate through the weightings until the other image is weighted heavily.

On either end of the spectrum in Figure 4, we see much of the texture taken from the respective style image. In the middle of the spectrum, we find a nice balance of both paintings that were chosen as nearest neighbors. One feature that was striking in this example are the diagonal black lines that result on the right side of the spectrum. We can see these very large features are being pulled from the Picasso self portrait, which was chosen as the nearest neighbor.

5.2 Multiple Inputs, Different Artists

Our results from the previous section were encouraging. However, one of the main goals of the project was to implement the algorithm effectively for multiple style images by different artists. Here you can see the result of weighting style images by different artists. We run the algorithm with K = 2, 10 iterations in the optimization step, and using images chosen from a collection of Picasso and Van Gogh paintings. We begin by weighting one image very heavily and then slowly iterate through the weightings until the other image is weighted heavily.

In figure 5, on either end of the spectrum, we see much of the texture taken from the respective artist. In the middle of the spectrum, both artists are represented. The features from both artists seem to blend with each other reasonably well. In our even weight output, final image takes thick dark lines from Picasso, but swirling blue brush strokes from Van Gogh. The result is rather convincing.

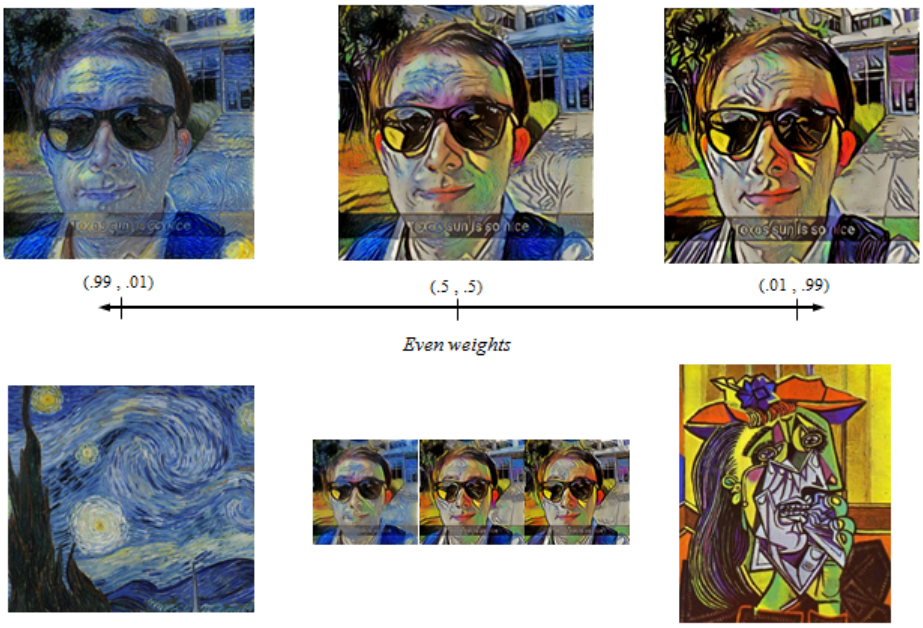

Figure 5: Spectrum of weights for style images by different artists

References

[1] Vgg-19 model description. `https://gist.github.com/ksimonyan/3785162f95cd2d5fee77`.

[2] Lasagne library github repository. `https://github.com/Lasagne/Lasagne`, 2015.

[3] BethgeLab. Vgg-19 normalized pre-trained weights. `https://bethgelab.org/media/uploads/deeptextures/vgg_normalised.caffemodel`.

[4] L. Gatys, A. Ecker, and M. Bethge. A neural algorithm of artistic style. `http://arxiv.org/pdf/1508.06576v2.pdf`, 2015.

[5] S. Karayev. Recognizing image style. `http://arxiv.org/abs/1311.3715`, 2013.

[6] A. Oppenheim, A. Willsky, and S. Hamid. Signals and systems, 1996. 2nd Edition.