PHOTOREALISTIC NEURAL STYLE TRANSFER

Brandon Finley
Applied Mathematics
University of Colorado, Boulder
brfi3983@colorado.edu

December 10, 2020

ABSTRACT

In this report, done for the Applied Mathematics 5720 class at University of Colorado, Boulder, we will be applying similar techniques from Neural Style Transfer (NST) to more realistic photos, hoping to keep their structure intact while also changing its style. Here, I mainly altered the architecture of the original network, that used a pre-trained VGG-19 model as the backbone. Namely, I tested different activation layers, different feature extraction layers and weights, and changing the loss function. The overall result was, sadly, a bit underwhelming as it was hard to keep the geometry the same but altering the style with the already given architecture as the framework. I suspect that more complicated methods should be used in order to gain further insight and more convincing results for this problem.

1 Introduction

In the world of deep learning, it is often forgotten that using predictions and classifications are not the only applications in the field. Additionally, it is known that the encoding of an image's characteristics is quite simple, that is it has spatial pixels with a number of channels. However, with the current advancements of Convolutional Neural Networks, it is more clear to see how deep learning can be applied to not just make predictions and classifications but also to alter the very characteristics of an image, whether that is using feature maps, attention mechanisms, generative adversarial networks, etc. In short, we increasingly see new ways to understand imagery and how computers might process that data. Among those techniques is style transfer. Style transfer utilizes deep learning as it optimizes the target photo to resemble both the original image as well as the style attributes of the given style photo. Although NST started as more of abstract paintings applied to realistic photos, I hope to utilize the framework to solve more photorealisic problems, such as color correction. All code will be recorded on GitHub.

2 Literature Review

Since 2016, Neural Style Transfer has been a part of the deep learning community [5]. There have been various models that have utilized its architecture. From my personal research, it appears that some of the breakthroughs have been fast style transfer [2], implementing perceptual loss [6], and even style transfer for videos that do not use any deep learning techniques. Within this report, I used an article found online for the baseline of the neural style transfer network [8], and then made changes to it with personal ideas. I also used slack exchange for various debugging and implementation. Although among these, style transfer applied to photo-realistic settings have been down, I decided to try to implement this with my own ideas.

3 Data

For this particular model, there is no training, validation, and test set. Using a pre-trained model, we froze the model's parameters. However, we cloned our input image and changed its content and style based off our loss functions. And

so, we ultimately over fitted a single image since that image encompassed training, validation, and testing. Nonetheless, some rudimentary data exploration will be used to look at how the original image compares to the style, its resolution, and at a high level, some aspects of the image that might pose problems when using our model.

Take for example the given input image and its corresponding style image. The input image was taken from the web [1] while the style image was taken from a personal favorite photographer, Garret King [7].

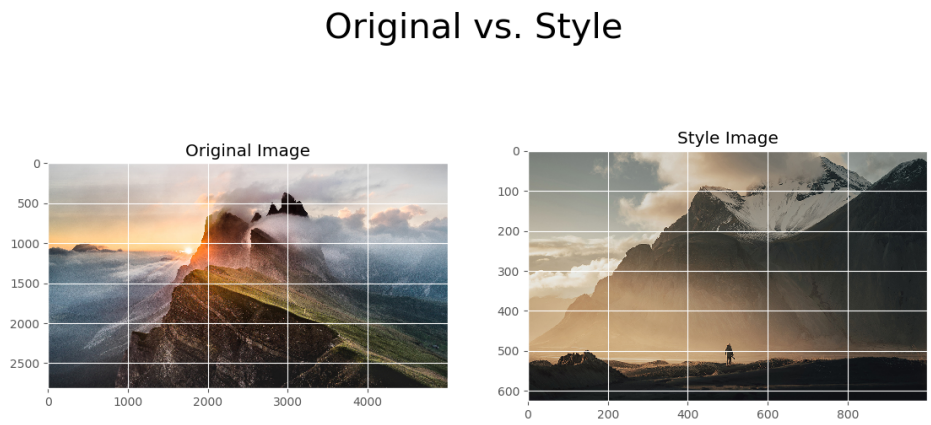

Figure 1: Input and Style Image

Here you can see that the style image is fairly similar to the original image as both feature mountains, some clouds, as well as some sun rays. However, clearly the styled photo is more muted and has a monotone color scheme to it. To gain further insight into these images, it is necessary to look at their composition from a photographers perspective, that is from their histograms, both luminance (light) and color. To do, we can unwrap each of our images and plot the histogram for luminance (Figure 2) and color (Figure 3).

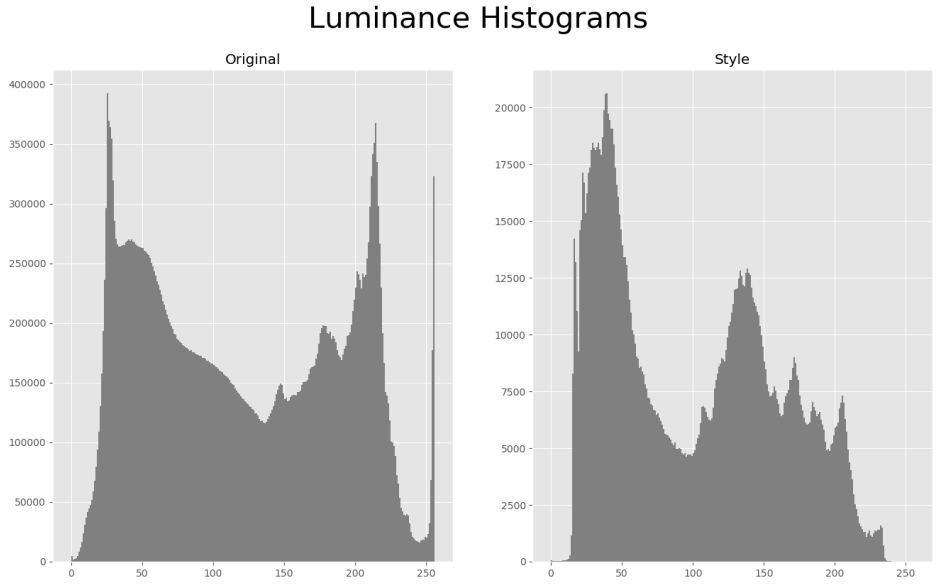

Figure 2: Luminance Histogram

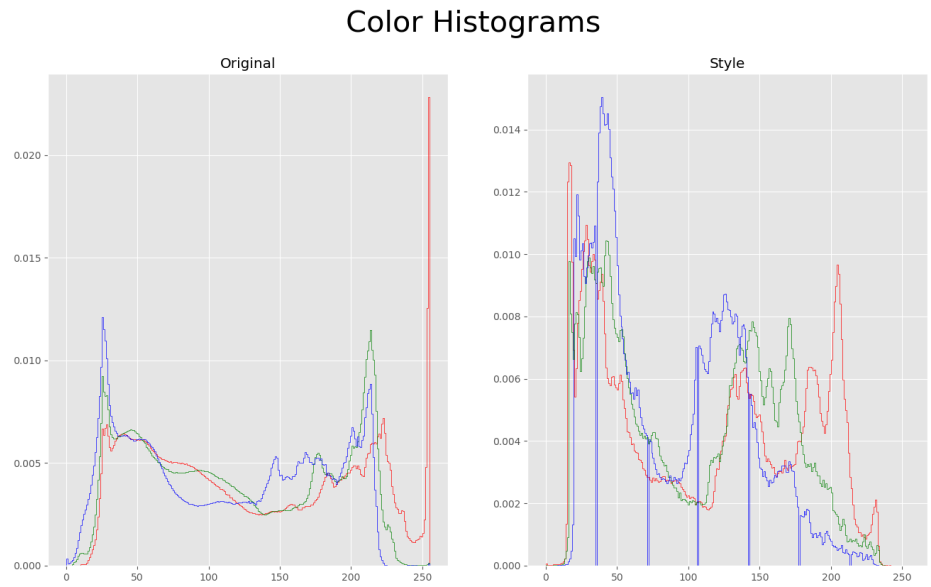

Figure 3: Color Histogram

From above, you will notice that the luminance histogram shows similar activities in the light spectrum as the shadows are heavily weighted and the midtones are less intense. However, it is clear that the stylized photo has its highlights (bright parts) dimmed as the original shows a peak near bin number 215 and 255, while the stylized photo has no such peaks and dies out completely near bin number 235.

From the color histogram, we see fairly different spectrum. Namely, the original photo has a balanced set of red, green, blue pixels, where none are overtaking the other in intensity. Contrary to this, the stylized photo has overwhelming blues in the lower end of the spectrum and overwhelming reds in the upper end. This provides the stylistic look, however, as the shadows are more blue and the highlights are more golden.

4 Model

The model presented in this paper will use the standard Neural Style Transfer (NST) as a backbone; however, it will contain alterations so that it can be specifically applied towards photos, namely matching the style but preserving the quality and geometry of the photo. Before we begin with the alterations, we will briefly go over the backbone and its techniques.

The backbone for NST, as noted in the original paper, is a pre-trained VGG-19. This gives us access to use many different types of convolution layers while generally keeping the dimension of the image in tact. Using this backbone, NST first works by taking intermediate feature maps from various layers, both for content and style. Now, using those feature maps, you must compute what is called the *content loss* as well as the *style loss*. For the content loss, we simply want our image in geometry to be close to the original one; thus, it is natural we compute how close they are in pixel values across the whole image. We also note that deeper feature maps maintain more complex structures and so are better feature maps for content. In our model, we choose the last convolution layer as our content layer, "conv5_4". To do this, we use the mean squared error (MSE):

$$L_{content} = \frac{1}{n}\sum_{i,j}^{n}(F_{i,j} - Y_{i,j})^2 \tag{1}$$

where F is the feature map and Y is the target image. The above formulation makes sense for the content loss; however does not make sense for the style loss as we wish to transfer similar components but for them not be exact. Thus, similar to a heat map one creates for correlation, we will create a gram matrix out of the features from the style layer. Doing this for particular style layers (as contrary to content layer, multiple style layers offer different aspects), and then summing them together, we get total style loss. This allows one to see a big picture of how certain features are

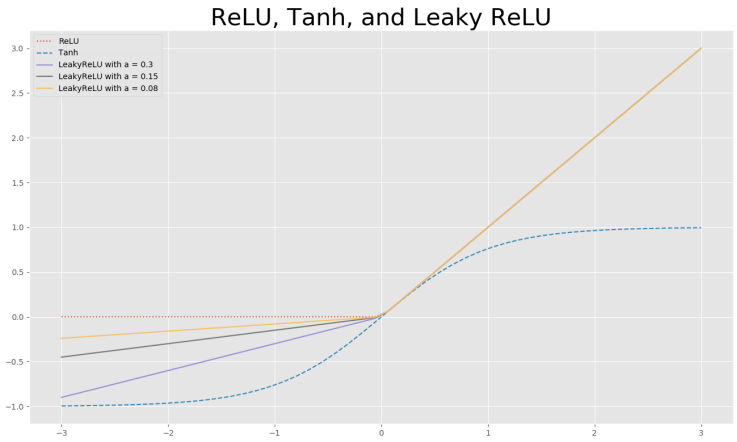

Figure 4: Activation Functions

related to each other, and so, can transfer the main sense of style onto our chosen picture. Rigorously, a gram matrix is defined as the a matrix that encompasses all possible inner products from a set of vectors $V = a_1, a_2, \cdots, a_n$.

$$G = \langle a_i, a_j \rangle \quad \forall a_i, a_j \in V \tag{2}$$

In order to preserve the original photo, one idea is to just increase the weight on the content loss, and thereby decreasing the importance of the style. However, what this entails is that the model then does not transfer all elements from a style perspective. A way around this is to maintain the relative importance of style in comparison to content but to alter the architecture or optimization of the original model. And so, we have used the following techniques to transfer the original model to a more realistic photo domain application.

Activation Functions: The most dramatic effect I encountered was the change of the activation functions. Recall the report I did that covered the topology of deep neural nets [3]. In this report, we learned that by using ReLU as an activation function, is not a homeomorphic function and thus does not preserve topological properties. As we wish to maintain as much information from the picture, this poses a problem. Also, notice the specific activation function on a graph in figure 4. You will notice that any information in the negative domain is completely lost. Now, this is good for fast convergence, and sometimes generalization; however, when we are using only a single image, we wish to maintain quality. Thus, a natural alternative to using ReLU is to use a homeomorphic function and one that preserves information. We will test both of these using *Tanh* as well as *LeakyReLU*. For the LeakyRuLU, I used a grid search over the slope values, a, being $[0.3, 0.15, 0.8, 0.0]$, where the last one is the same as ReLU.

Content and Style Layers: In addition to using select activation functions, it is noted from the original paper that the specific layers were chosen in an effort to maintain the geometry but also transfer the style from *paint-like* photos. In my model, we will utilize that idea but attempt to mainly preserve the photo. Doing so, recall that the deeper layers in a Convolutional Neural Network (CNN) go, the more complicated the feature maps. This is evident from a previous report I did, where I visualized the feature maps from a pre-trained CNN [4]. Thus, as the content layer was chosen to prioritize more complex features rather than edges, it seems sensible to apply the same methodology to the style layers. Now, one can *change the style layers to be deeper* and weight them evenly as before, or take the already chosen layers and weight them to prioritize the deeper style layers. We will focus on the latter and if time permits, do the former. In that case, I changed the following weights that correspond to [“conv1_1”, “conv2_1”, “conv3_1”, “conv4_1”, “conv5_1”] from $[1.5, 0.80, 0.25, 0.25, 0.25]$ to $[0.1, 0.3, 2.0, 7.0, 10.0]$.

Additionally, I converted the max pooling layers to be average pooling to preserve information.

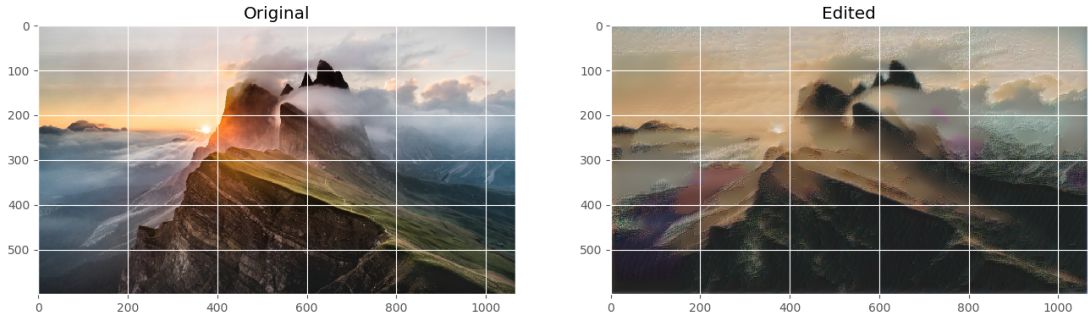

Figure 5: Standard Tangent

5 Results

In this results section, we will be mainly using a grid search to look at the different combinations between the different activation functions as well as the different layers and weights. This was done by simply fixing one variable and altering the other. I am sure I missed some combinations as each iteration took a while to compute; however, will still got fairly comprehensive coverage of the different alterations.

5.1 Activation Functions

We first start with altering the type of activation functions. As noted above, I hypothesize that LeakyReLU or Tanh would work well as it does not negative the negative activated neurons. We first start with Tanh and its various combinations. Figure 5 was generated with the content weight being $1 \cdot 10^3$ and the style weight being $1 \cdot 10^6$. Additionally, the standard convolution layer, "conv5_4" was chosen as the content layer and the standard style layers were chosen with their original weights (see section 4). We can see that this output is far from what we intended to create as its geometry is still fairly lost. Also note that this is with Tanh, not ReLU or LeakyReLU and that the number of iterations used was 1000.

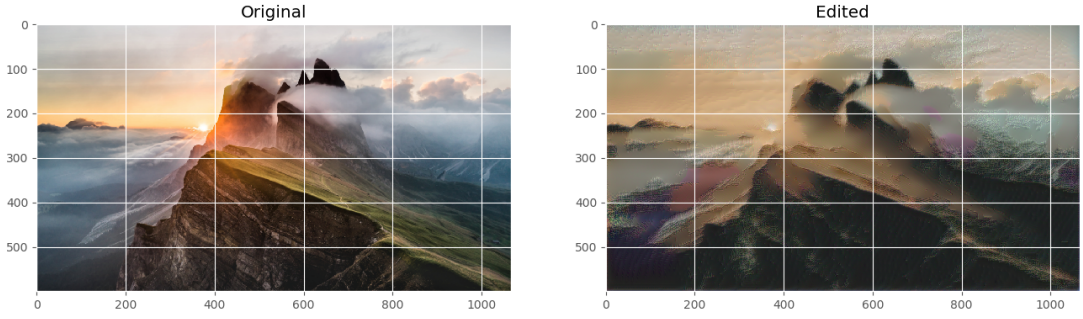

Figure 6: Tanh with Weighted Content

As seen from the results for this particular example, it appears the weighting the content layer does not appear to make much of a difference, that is with the other parameters fixed. Also note that while I hypothesized that using Tanh would help in preserving information, it appears that it does just as bad as ReLU. After looking at the graph for Tanh, however, it does make sense. Just like the vanishing gradients, with Tanh you will saturate some of your weights and thus it will not progress much. While we are not changing the weights of our feature maps, the saturation from Tanh may be causing loss of information from our original image's geometry, thereby causing its final outcome to resemble that of ReLU.

Finally, we have the results from implementing LeakyReLU. I ran four different test with 1000 iterations each, with the parameters for the slope value being $[0.3, 0.15, 0.08, 0.0]$. This allowed us to see the progression from the different values as well as compare them to ReLU.

First, let us consider that the parameters for this are 1000 interactions, and default style layers and weights. However, we will be changing the value of the slope in LeakyReLU. We will begin with a = 0.3.

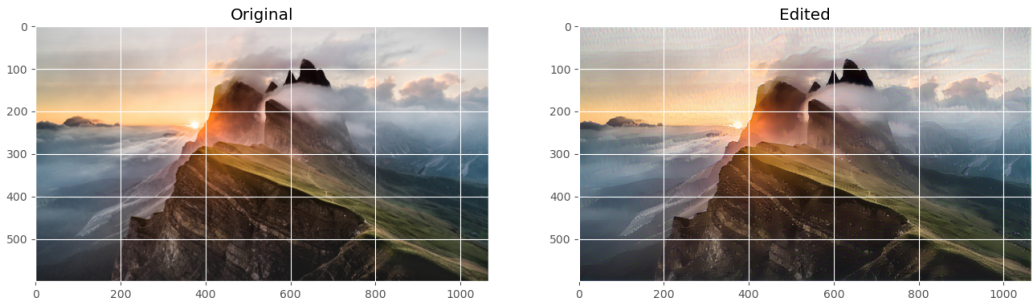

Figure 7: LeakyReLU with a = 0.3

Notice that with a higher slope value, that, generally, more information is preserved. This can be seen as all other parameters were fixed but the change of this slope value caused the stylized image to be fairly intact structurally. We will see the effects of slowly decreasing this slope value.

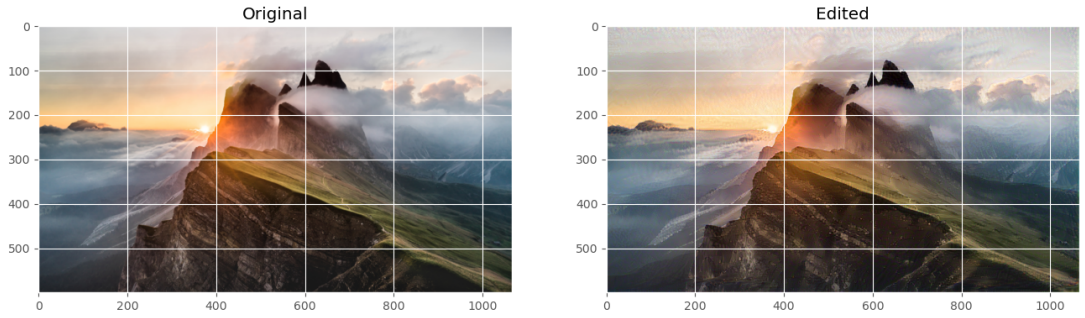

Figure 8: LeakyReLU with a = 0.15

Next, with the same fixed parameters, we have $a = 0.15$. You will notice that it does not appear to change much in terms of the output stylized photo. It appears to be the same resolution as the image above with maybe some hazier parts of the image; however, it is fairly comparable to $a = 0.3$. I do not expect this to hold up, however, as ReLU is dramatically different from the rest.

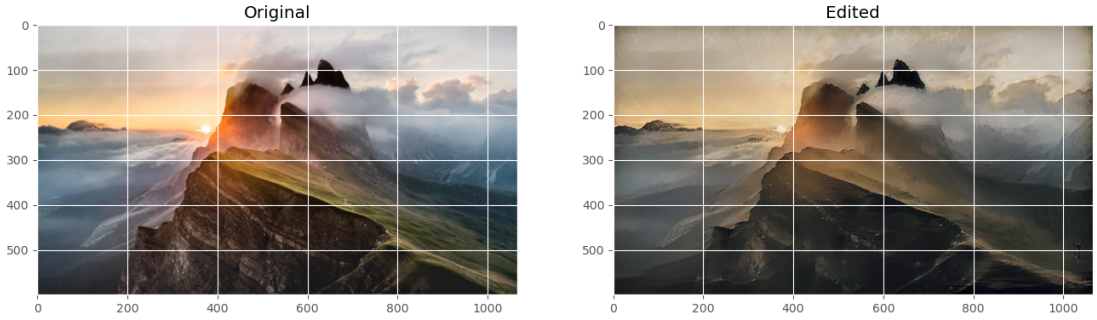

Figure 9: LeakyReLU with a = 0.08

Now with $a = 0.08$, we notice the largest change in our image. Namely, it appears to have not only its texture but also its color altered. It is interesting to me that rather than seeing a shift of one before the other (since we implemented the color loss functionality), we see both change at the same time. Ideally, I would want to separate the two so that the user has some control over texture versus color. This would allow one to choose more of a painting rather than a photorealistic result.

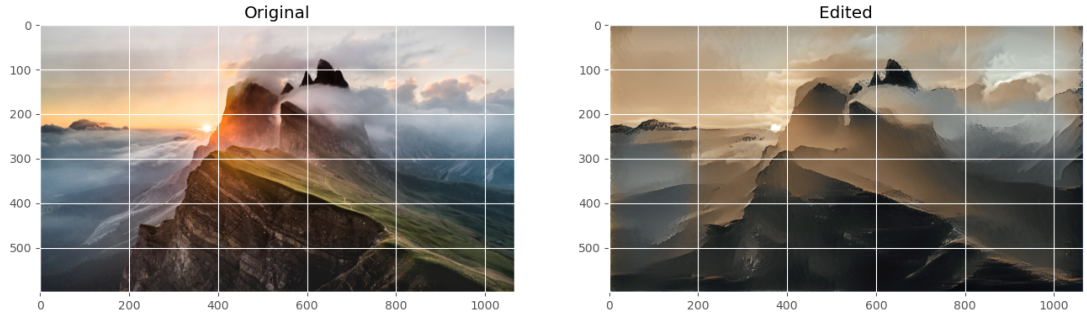

Figure 10: ReLU

Lastly, we have ReLU, or LeakyReLU with $a = 0.0$. As one can see, the activation function is very effective at changing the texture but not with preserving the geometry of the original photo. This was exacted as RELU is not a homeomophic function as it "folds" space with multiple mappings to the line y=0.

In general, there does appear to be a trend with the slope and the effect of the stylizing; however, it also appears to be a bit less gradual that I was expecting. Therefore, it is hard to determine if one can control the extent of this method, and so, I might recommend an alternative method where the results are a bit more predictable. Granted, this was for this particular image; however, it appears evident that using LeakyReLU is a bit trickier than previously though in style transfer.

5.2 Altering Style Layers

Now, we attempt to determine if changing the style layer weights will be advantageous. To do this, we use the same setup as our second architecture but change the layer weights to focus more on the deeper style feature maps, thereby causing less of a pastel look onto the image and hopefully more of a color or light transformation instead of texture. We can see the results in figure 11.

Next, we have the same architecture, layers, and activation function; however, we simply switched the content and style weight. That is, now the style weight is $1 \cdot 10^3$ and the content weight is $1 \cdot 10^6$.

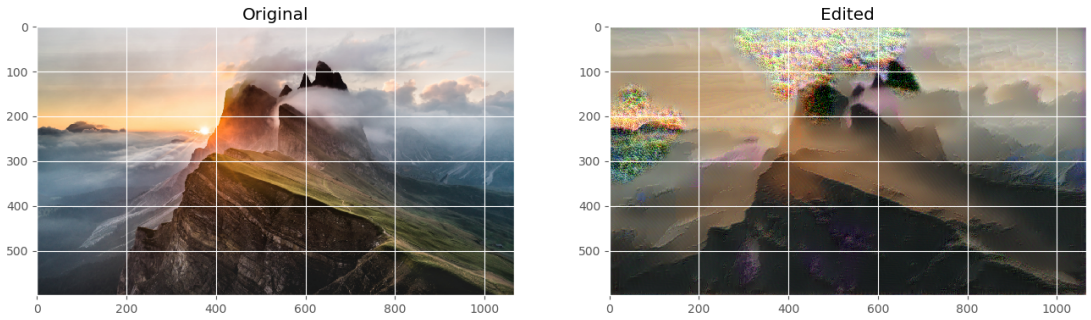

Figure 11: Tanh with Deeper Style Layer Weights

Notice that the results are actually opposite of what we want and that there is clearly some side effects from doing it this way. While this is only for a particular sample, this tells me that this is not a proper method and should be avoided.

6 Ablation Studies

For our ablation section, I wanted to try to change the loss function so that our model focused more on color rather than texture. While I have done other tests in the model that could be included in this section, I unfortunately did not include some of the results to show. For instance, I tested what happens when you dramatically increase the instances, maxing the number to 7000 (seems to increase style effects linearly). I also tested what happens when you pass your image through all subsequent layers rather than individual layers (appears to actually cause less effects as the feature maps include all subsequent layers). Nevertheless, for this section, we decided to focus on our color loss implementation.

6.1 Color Loss Function

Seeing that our results prioritized texture in its style transfer rather than color, I sought out an alternative method to try to introduce color into the loss function. We can use the same method for style except use it for the input images that uses their RBG channels as our features.

Loss Function: Specifically, I decided that many of the techniques above did alter the photo and its style in the direction I wanted, but it appeared that color by itself (rather than texture or other style elements) was not altered as dramatically. As the main focus of this report is to apply a type of style transfer (such as color correction) to realistic photos, I wanted to somehow prioritize colors. Although this idea surfaced near the end, I was able to implement some form of it. Namely, using the same techniques described above for capturing the correlation between features in a gram matrix, I applied the same technique but for the original image's color. This was done by passing in the original image (as well as the image being optimized) and creating a gram matrix by flattening the channel dimension, resizing to match the smaller shape between the original and optimized image, and then taking the MSE between the two gram matrices. Although fully implemented, this seemed to lack functionality from the tests I ran, and so I wonder what a better way to implement this would be. Formally, this was done by changing the loss function

$$L_{total} = L_{content} \cdot w_{content} + L_{style} \cdot w_{style} \quad (old) \tag{3}$$

$$L_{total} = L_{content} \cdot w_{content} + L_{style} \cdot w_{style} + L_{color} \cdot w_{color} \quad (new) \tag{4}$$

Using this method, I personally found that it generated minimal results. However, I will note that during this implementation that I had to push my comfort level with PyTorch as I was resizing the tensor during the forward pass to do calculation on it. While I would rather register a forward hook, I needed to preserve the computational graph so that back-propagation could occur. Thus, I am not sure if my implementation was completely correct.

The reason why I doubt this is that I set my weights for (content, style, and color) to be (1.0, 1.0, $1 \cdot 10^{10}$), respectively. Thus, if I witness minimal results, this tells me that this method does not work. I plan on looking further into this as intuitively I believe the method makes sense.

7 Conclusion

In summary, I wish I could say that this report accomplished what to do for photorealistic results; however, this paper serves more of what *not to do* to get the desired results. In general, it appears that LeakyReLU helps the most with combining content and style in a realistic manner as well as changing the weights on your overall content and style loss. A finer mesh with different slope values might pose as the best result. Additionally, it appears changing the style layers and weights do not make a noticeable different for photorealsitic results. Lastly, although a implementation of color in the loss function seems to be reasonable, either my implemention or method is incorrect. Unfortunately, the results were not particularly convincing; however, I can say that apart from the base Neural Style Transfer backbone, all ideas here were in fact original. In the future, I recommend a more exhaustive grid search be done to cover all bases including multiple test input and style photos. I also would urge a second look at the color loss implementation. Nonetheless, I learned a lot going through this report and look forward to seeing how others generated photorealistic results.

References

[1] dolomites mountains mountain landscape nature, hd wallpaper.

[2] Logan Engstrom. Fast style transfer. https://github.com/lengstrom/fast-style-transfer/, 2016.

[3] Brandon Finley. Topology of deep neural networks (paper review), 2020.

[4] Brandon Finley. Visualizing convolutional neural networks, 2020.

[5] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks, 2016.

[6] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution, 2016.

[7] Garrett King. Iceland 2019.

[8] Gaurav Singhal. Artistic neural style transfer with pytorch, 2020.