Stone Mele
04/23/2019
Deep Learning for Image Analysis
Tolga Tasdizen

Artistic Style Transfer Paper Recreation

Introduction:

Artistic style transfer is an incredibly interesting, engaging, and unique field within deep learning. On the surface level it may seem like a trivial pursuit with the goal of generating pretty pictures (see Figure 1), but it is far from that. Artistic Style Transfer has allowed us to gain new insights into what specific information is being extracted in certain regions of a network by the various convolutional filters. The interpretation of neural networks and the results they provide is a still a very open field of study within deep learning, with artistic style transfer only helping spark interest in interpretability and peeling back the layers behind the mystery of how neural networks actually represent these concepts behind the scenes. The main results from the Gatys et al paper is that deep learning models (VGG19 specifically) do have representations for content and style that are separable. In this write up I will discuss my attempt at understanding, recreating, and implementing the work done in Gatys et al paper.

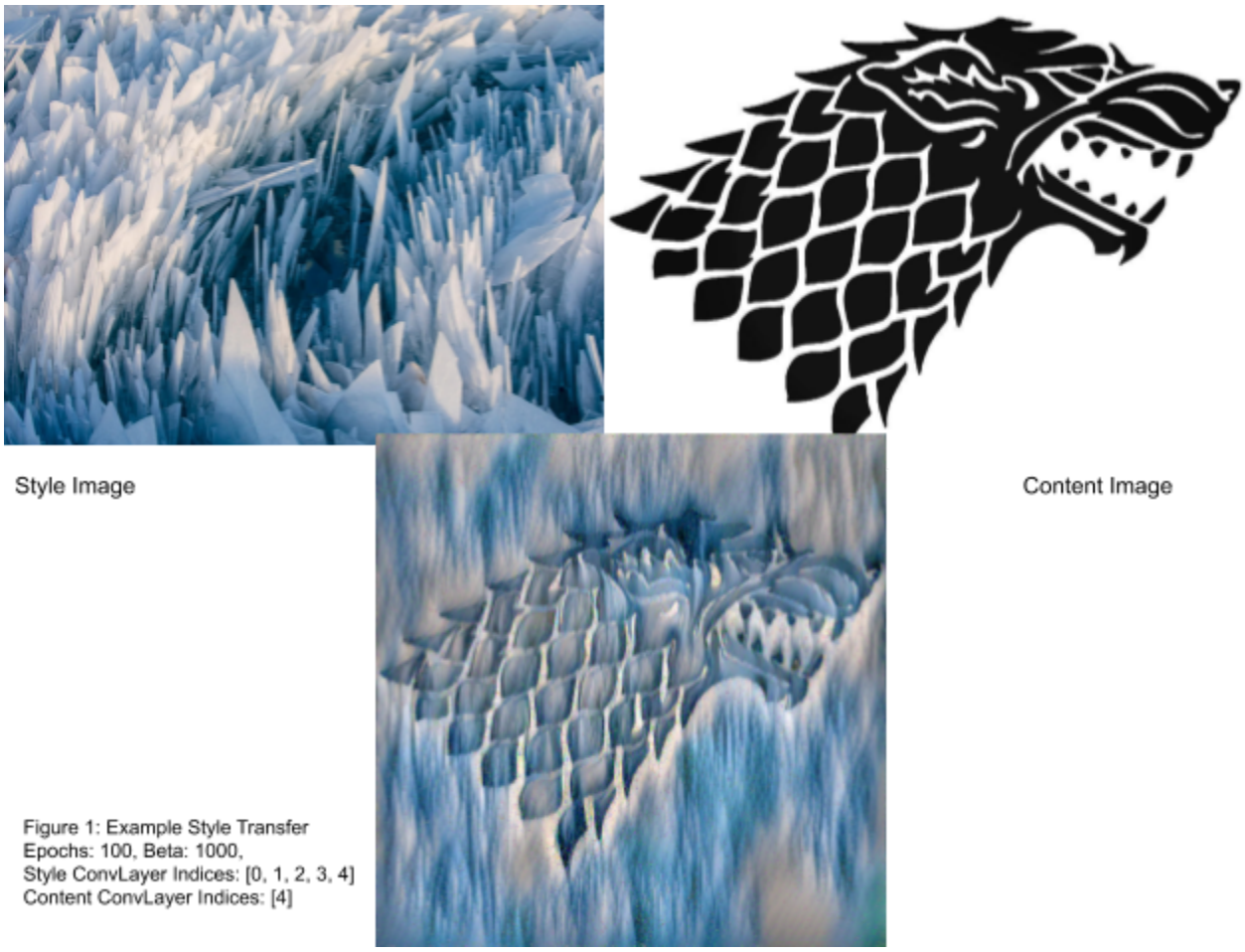


Figure 1: Example Style Transfer
Epochs: 100, Beta: 1000,
Style ConvLayer Indices: [0, 1, 2, 3, 4]
Content ConvLayer Indices: [4]

Methods:

General Architecture

In order to achieve the results from the paper one needs to shift their mind from how they normally think about a neural network and how to train one. This task is quite weird in that the network does not change throughout the whole process while a single image is the piece of data that is being manipulated and optimized over. Instead of searching for weights that result in a satisfactory classifier or regressor, we are searching for an image that results in a satisfactory level of activation within the network itself. In our case a satisfactory level of activation is one that matches the activations of the content and style images at their respective convolutional layers of interest. This whole process was quite weird to think about especially in comparison to traditional training frameworks, however once the whole network architecture is laid out, and the loss functions are put in place it becomes clear. A simplified example of a network architecture that can be created with my implementation and like those discussed in Gatys et al can be seen in Figure 2.

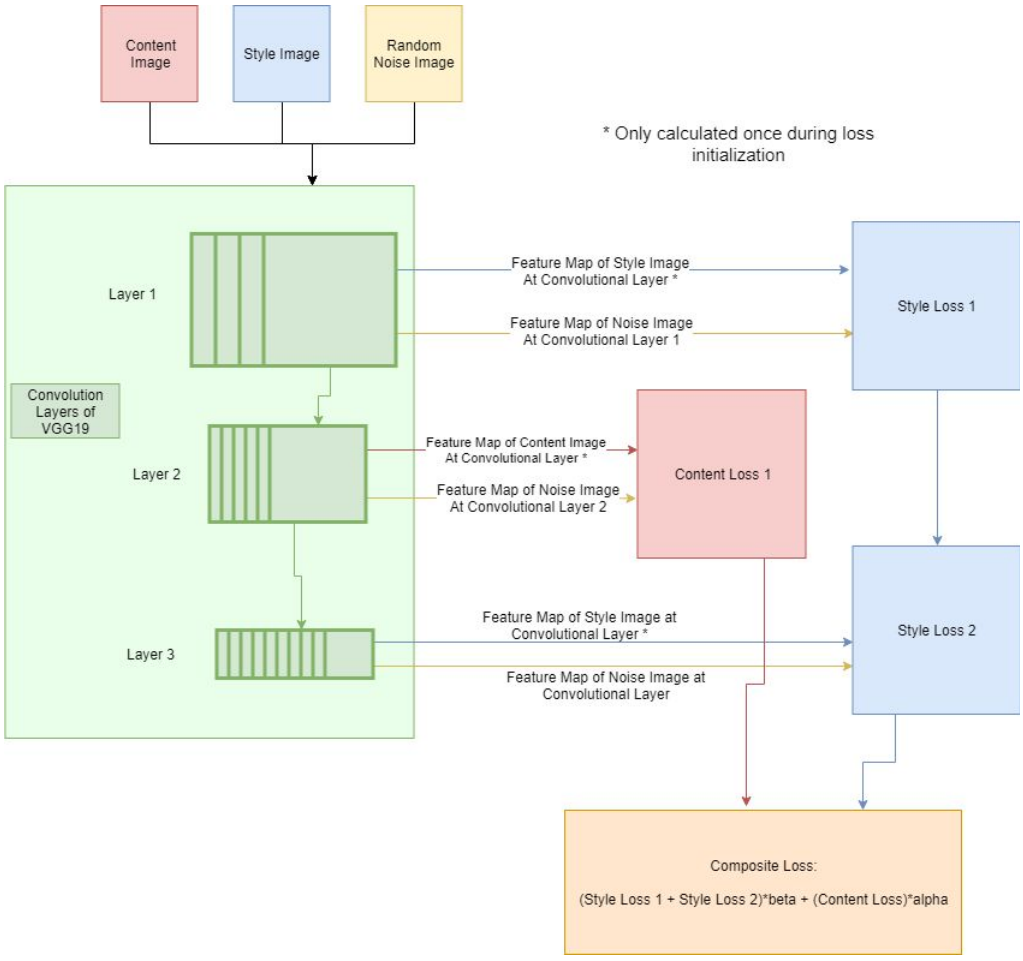


Figure 2: Example network architecture

A brief overview of how this model works is that both style and content losses are attached to various convolutional layers within the network. These losses monitor the amount of content and style the generated image is gathering. Like with most neural networks we take advantage of gradient descent and try to minimize the loss between the content and style losses simultaneous hopefully reaching an image that embodies both respectively.

The network shown in Figure 2 is not the only permutation of the network possible as any number of style or content losses (more information on the different types of losses in methods section) can be added to the network. The only limit is on the number of convolutional layers in the network itself. In Figure 1 we have one content loss, and two style losses, however not shown in Figure 1 the content and style losses can also be added onto the same Convolutional layer they do not always need to be exclusive. A majority of the images generated within this write up in fact contain content and style losses on the same convolutional layer. Another interesting thing about the proposed methods for this paper, and my implementation of it, is that it can be applied to any convolutional neural network not just VGG19 like that used here.

To initialize the loss modules within the network we need to run the content and style images through the network and store their feature maps at the desired convolutional layers for use later when calculating the loss. Once this is done we can then iteratively run the random noise image through the model and update the image based on the loss value calculated from the composite loss. This composite loss is comprised of two main components. Those being the content losses, and style losses. I will go over each of these components in following section.

In order to implement this network as a whole I made a Pytorch Module, this can be seen in the transfer.py file under the StyleTransfer class. This class requires your provide an existing model with convolutional layers, an image with the desired content, and an image with the desired style, both the indices of the convolutional layers you want to incorporate into your content and style losses, and various hyper parameters used for weighting content and style losses. What this class does is it takes the existing network and recreates it while attaching content and style losses to the convolutional layers specified by the indices given on the class initialization. During the initialization is when the feature maps for the content and style images are ran through the network and are saved for later use. Using this class will give you a model you can use to optimize your noise images with so that the desired content and style are transferred over.

An important nuance to generating these images that caused me some trouble is the variant of Gradient Descent that you use. For the Gatys at el paper the author uses the L-BFGS [1] algorithm for their image generation. This is a small implementation detail I ran into after trying to get both Adam, and Standard Gradient Descent to

converge well, and reliably. Fortunately Pytorch has an implementation, and after switching over to the L-BFGS algorithm the hyper parameter search became much easier as I could use the parameters presented in the paper as a good starting point. Before this paper I had no experience with L-BFGS algorithm for gradient descent, but after some research it has been shown to outperform SGD in training time for many scenarios [4]. This is very beneficial advantage as the more images I can generate the more information I can gather and learn from.

Content Loss:

The content loss refers to equations 1, and 2 within Gatys et al paper also shown below as Equation 1, and Equation 2.

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

Equation 1

$$\frac{\partial \mathcal{L}_{content}}{\partial F_{ij}^{l}} = \begin{cases} \left(F^{l} - P^{l}\right)_{ij} & \text{if } F_{ij}^{l} > 0 \\ 0 & \text{if } F_{ij}^{l} < 0 \end{cases}$$

Equation 2

For these equations p, x, and l respectively correspond to the content image, the generated noise image, and convolutional layer to take the feature maps from. Gatys et al also defines P^l and F^l as the corresponding feature maps for p and x after they have reached convolutional layer l in the network. This leaves the final equation as a fairly simple one as it's just the element wise squared error between the two feature maps of p and x at convolutional layer l. I did diverge from the original implementation for this section a little in that I added a normalization term where I divide by the number of total elements being square and summed. I did this as the loss values for the content loss layers were getting fairly large, and this helped bring them back down to a reasonable size. This means the final content loss function is practically the mean squared error between the two feature maps.

I choose to implement this loss functions as a Module within the Pytorch eco system. I choose to do it this way as it made it easy to integrate these loss layers into already built in networks and Pytorch Modules provide a simple interface to work with the forward and backward pass through a network. My code for this can be seen in transfer.py under the CLoss class. The code simply saves the feature map of the content image through the network, and compares it to the generated noise image after each pass. This leaves us with a loss value with respect to the content image every time an image is ran through the network. Like discussed above there are no limitations to how many content losses are present in the network just so long as they appear after a convolutional layer.

I think it's important to discuss the interpretation of this content loss function and what is actually minimizing. As discussed above we are simply taking the mean squared

error between the two feature maps. Trivially the optimal image to minimize the loss is the original content image itself so that is what the model will tend to recreate. However as you traverse down the network the convolutional filters don't get to work with as much low level information about the image as earlier layers. Knowing this the question is now where is the best place for the content loss in the network? If placed on the higher up convolutional layers the optimizer will have an easier time recreating the whole image especially down to the minor details. If you place the network on lower level convolutional layers the loss will be less informative as the features it gets to work with are more high level, thus resulting in a reconstruction that mostly retains the more abstract features of the content image. It may seem counter productive at first but it is more beneficial to place the content loss on layers at the lower levels so the optimizer only has the capability of generating a higher level abstraction of the content. This will deter overfitting the minute details of the content image allowing the style to also get its chance to manifest itself within the generated image. An experiment is detailed in the experiments section under Content Reconstruction in which I explore this question and show how the higher level, and lower levels of the convolutions relate to the reconstruction of the content image.

Style Loss:

The Style Loss refers to equations 3,4,5,6 from the Gatys et al paper also shown below in equation 3,4,5,6 . The same notation as in the Content Loss section is used, and additionally added a, G and A respectively refers to the original style image used to initialize the network, the Gram matrix of the feature map for the generated image, and the Gram matrix of the feature map for the style image.

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

Equation 3

$$\frac{\partial E_l}{\partial F_{ij}^l} = \begin{cases} \frac{1}{N_l^2 M_l^2} \left((F^l)^{\mathrm{T}} \left(G^l - A^l\right)\right)_{ji} & \text{if } F_{ij}^l > 0 \\ 0 & \text{if } F_{ij}^l < 0 \end{cases}$$

Equation 4

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

Equation 5

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

Equation 6

A new concept known as the Gram matrix (equation 6) is also introduced within this section. The Gram matrix is defined as follows say your given a matrix that is $y \in R^{(N,M)}$ then the gram matrix is simply the result of performing a dot product between

every N vectors. This is the same operation as performing $y * y^T$ resulting in a (N,N) sized matrix. In our case a normal input size for layers within a convolutional neural network is (Batch Size, Number of Filters, Width, Height) in order to perform the Gram matrix calculations a reshaping of the matrices is required so we end up with (Batch Size * Number of Filters, Width*Height). Since the batch for our case is always 1 the transformation is pretty trivial.

What does the gram matrix actually represent though? When the original Gatys et al paper first came out the answer to this was not clearly presented within the paper. After researching and reading through the Li et al [3] paper, in which the Gram matrix can be shown to be reformulation of minimizing a maximum mean discrepancy (MMD) problem, I have gained some intuition. The Gram matrix gives us a way to reason about the distribution of the relationships between the features in the features maps of the convolutional output. Comparing the two gram matrices, of the style and generated image together, allows one to compare how the distribution of the relationships between the two seperate feature maps compare. For style transfer we do not really care that the features are activated in the same location we only really care that the features share the same relationship distribution between them. This is different than how the content loss is used as we are simply taking the mean squared error between the raw features maps. This means the location of the activations within the feature maps is critical in achieving a good loss, conversely this should not be an issue when transferring style as its a global attribute of the image invariant to location. Overall the style losses main goal is to achieve a similar relationship distribution between the features at specific convolutional layers between the original style image and the generated noise image.

The same question as before appears where does one place the style losses in the network? Unlike the content loss in the paper and seen within the experiments section under Style Reconstruction we can see that each convolutional layer holds its own piece of the overall style. Knowing this it's critical to play around with the indices you choose when generating images as different combinations will result in very different images.

The Style Loss and Content Loss are fairly similar constructs mathematically as the end result are practically the mean squared error between two different features maps, however even though these two losses seem quite similar the simple transformation into the Gram space really changes a lot in how the generated image is created. My code for this loss function can be seen in transfer.py in the SLoss class. It follows a very similar structure to the CLoss class except logic for the Gram matrix calculation is included.

Composite Loss:
The composite loss refers to the combination of the content Loss and style Losses throughout the network. The mathematical formulation for this is given in equation 7 of the Gatys et al paper also shown in equation 7 of this paper.

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

Equation 7

The compositie loss is a weighted summation of all the losses within the network. The weights α, and β play a critical role as will be see in the experiments section under Composite Loss Parameter Exploration. A bad selection of these will result in unsatisfactory results in the final image generated.

Experiments:
For this project I tried to recreate all of the experiments/figures within the paper Gatys et al paper. This consisted of exploring both content and style reconstructions as well as exploring the parameter space that relates the two together. Following are details of the experiments I ran and the images that they generated.
Content Reconstruction**:**
In this experiment I attempt to perform content reconstructions at various layers throughout the network. The paper describes a guideline for doing so, and fortunately it was quite easy to set this experiment up with the use of my StyleTransfer module. The code for this experiment can be seen in ContentReconstruction.ipynb. In this experiment I try and reconstruct the original content image from a randomly generated noise image. I do this for convolutional layers with indices at [0, 1, 2, 3, 4, 5] each running at 100 epochs and an α value of 100, no style losses were included when making these images. Figure 3 shows the results of this experiment.

Figure 3: Content Reconstructions

As discussed above, in the methods section for the Content loss, as the higher convolutional layer indices are reached the image quality degrades, but the overall content of the image remains. Conversely the initial layers of the network can almost recreate the image perfectly. This is a good indication that we want to stick with layers not at the beginning of the network to avoid overfitting the content to the generated image hindering the ability for the style to present itself.

Style Reconstruction**:**

This experiment is very similar to the content reconstruction experiment except it is trying to reconstruct the style of the original style image in comparison to the content image. The code for this can be seen in StyleReconstruction.ipynb in which I try and reconstruct the style from randomly generated noise image using convolutional layer indices of [0, 1, 2, 3, 4, 5] each running for 100 epochs with a β parameter of 10000, with no content losses included. Figure 4 shows the results.

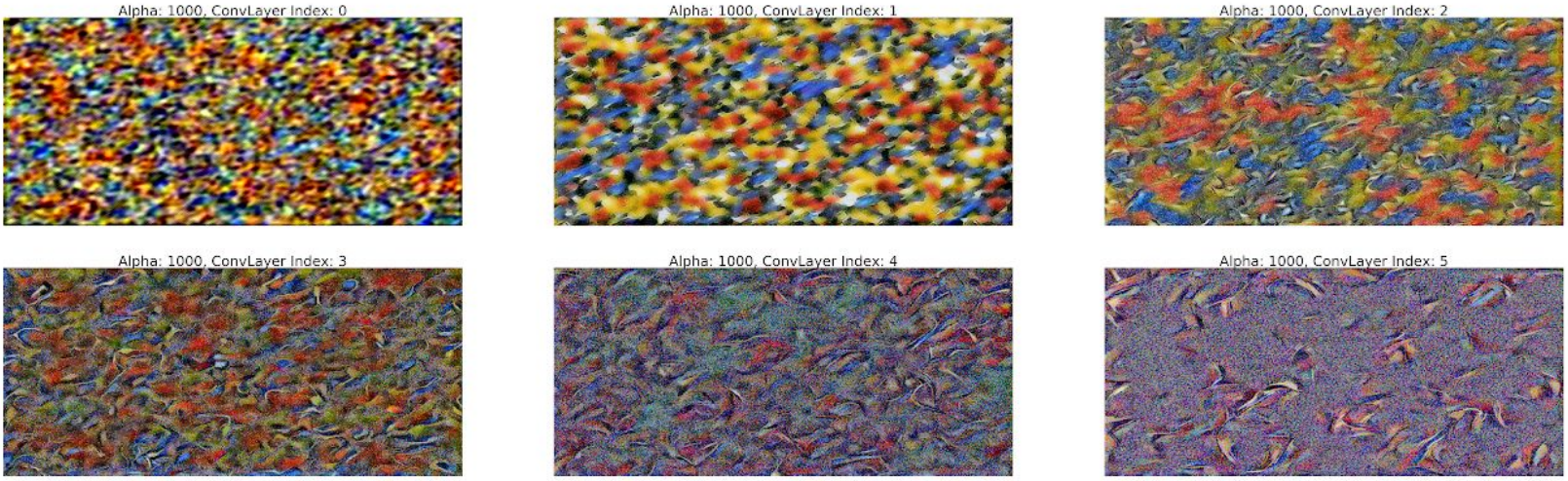


Figure 4: Style Reconstructions

As described in the methods section for Style loss we see that each individual convolutional index has its own unique style to contribute to the overall image. This fact leads us to not only apply the style loss to just one convolutional layer but a combination of them in order to achieve the best looking results. This is also interesting on an interpretability perspective as this shows different features at various gradients of intensity are stored along the convolutional layers.

Composite Loss Parameter Exploration**:**

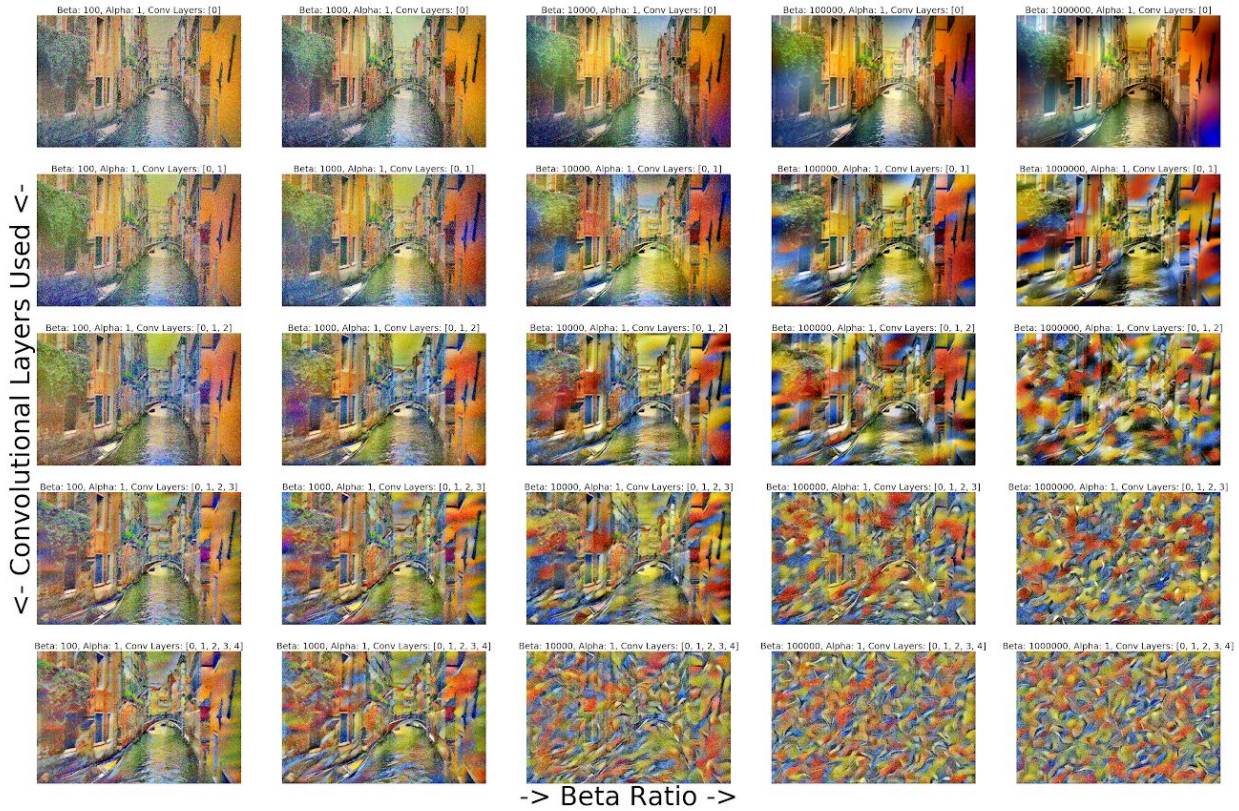
I really wanted to recreate Figure 3 from the Gatys et al paper as I thought it displayed the key components of the parameters space and how it can be used to form different images. To create it I set up a simple script that ran through a grid of parameters creating a new image for each different pairing. I used Figure 5 and Figure 6 as the

content and style images. The parameter space I tried had the content convolutional layer index of 4, β's at intervals of [100, 1,000, 10,000, 100,000, 1,000,000], and style convolutional indices using values [[0], [0,1], [0,1,2], [0,1,2,3], [0,1,2,3,4]] this resulted in a grid of 5x5 images seen in Figure 7. In Figure 7 the use of the different convolutional indices for style are along the Y axis while the different use of β's are along the X axis.

Figure 5: Content

FIgure 6: Style

Figure 7: Parameter Exploration



As you can see as expected while the β parameter and the number of style content indices increase so does the level of style that is transferred to the image. It's quite cool to see how the content of the image is slowly swallowed by style as the parameter space is expanded. It's also interesting to just examine a single column so you can see just how much the style is dispersed through the different layers, showing not a single convolutional layer holds the key to transferring the style over further supporting the results from style reconstruction experiment.

Literature Survey:

With the initial release of this paper and its newly found technique a lot of interest was direct towards this area of research. A very interesting and insightful paper that came as a direct result of the Gatys et al paper is Demystifying Neural Style Transfer [3]. This paper takes the Gram matrix and shows it can be reformulated to minimizing the maximum mean discrepancy with a second order polynomial kernel. This helped me understand the true nature of the style loss in that it is just a way to compare the two feature relationship distributions between the style and generated images. This reformulation also opened up the possibility of making use of the kernel trick in which the paper explored. I would highly suggested reading through this paper to gain a better understanding for the style loss and seeing the interesting ways using different kernels transform the overall style that is transferred.

After reading through current papers on the subject the technique presented in this paper is no longer the only way to transfer style using neural networks. The new method [5] uses an autoencoder network in which both the content and style images are ran through the encoder and then combined within the latent space in a specific way so that when ran through the decoder a stylized image is generated. The main benefit of this new process is that it can achieve a stylized image in a feed forward pass of the network, so no optimization methods like that seen in Gatys et al paper are required. This provides a huge speed up and allows for playing around with the parameters affecting the stylized image to happen much faster. Even though new techniques for transferring style have arisen the concepts presented in the Gatys et al paper still prove to be novel and valuable advances within the deep learning community.

Conclusion:

This project was a very insightful journey into some of the inner workings of convolution layers of a neural network. I gained a lot of insight into how neural networks represent and propagate features throughout. Trying to interpret the difference between the results of the content, and style loss was also a very valuable exercise in seeing what information can be extracted from the convolutional layers of the network, and how simply changing the perspective of the feature maps unlock a whole plethora of

valuable information. A key insight I experienced that I did not expect to discover was the shifting of my mind from viewing neural networks from being a complete black box feature extractor to something I could explore, and discover interesting things about. This project has only sparked my interest into this branch of the deep learning family and I hope to continue exploring this space.

There are a few directions in which I would like to continue exploring this project. The first being the types of models that I use as the base for my network. VGG19 is quite a large network in memory requirements so it would be nice to work with a smaller model which would hopefully let me generate higher resolutions images. I also think it would be interesting to see if the relationship between the content, and style encoded in the convolutional layers holds similar across different network architectures. I think this would be valuable as it would allow us to gain insights into how different architectures store relevant feature information throughout the network.

I also think it would interesting to explore different transformations to the feature maps within the loss function in comparison to just the gram matrix. As hinted at in the Li et al paper the gram matrix is not the only useful metric in extracting style information. Working with different transformations could possibly assist in rate of convergence for the images, or even help enhance the features that manifest themselves.

Finally I would also like to explore the new advances in this area of research, most specifically the architecture described in Universal Style Transfer via Feature Transforms as this seems to be a much more generalizable, speedy, and customizable way of performing style transfer.

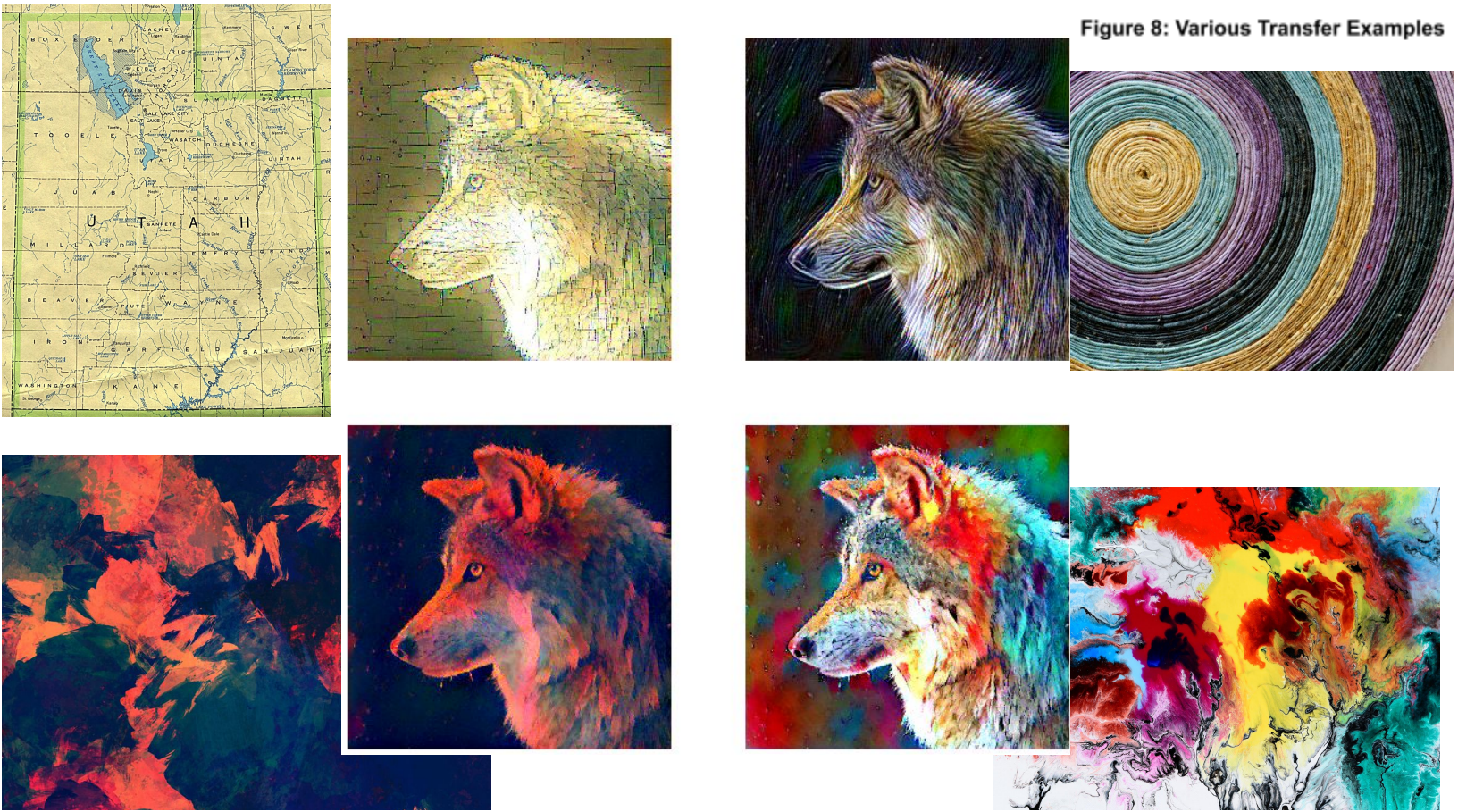


Figure 8: Various Transfer Examples

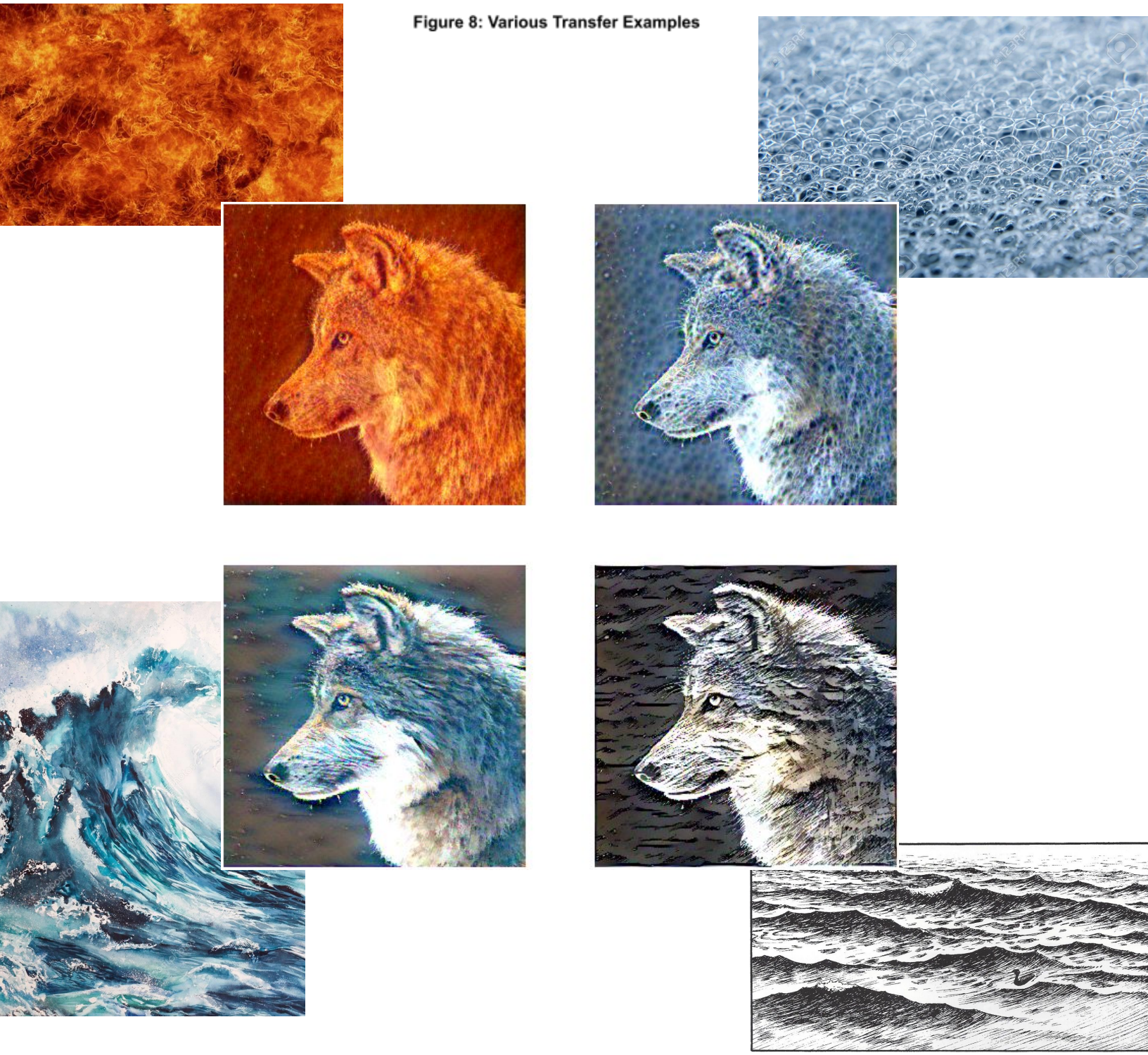


Figure 8: Various Transfer Examples

References

1. Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In CVPR, 2016
2. L. A. Gatys, A. S. Ecker, and M. Bethge. Texture synthesis using convolutional neural networks. In NIPS, 2015. arXiv:1508.06576v2
3. Y. Li, N. Wang, J. Liu, and X. Hou. Demystifying neural style transfer. arXiv:1701.01037, 2017
4. V. Le, Quoc & Ngiam, Jiquan & Coates, Adam & Lahiri, Ahbik & Prochnow, Bobby & Y. Ng, Andrew. (2011). On Optimization Methods for Deep Learning. Proceedings of the 28th International Conference on Machine Learning (ICML-11). 2011. 265-272.
5. Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, Ming-Hsuan Yang. Universal Style Transfer via Feature Transforms. arXiv:1705.08086v2