A Neural Algorithm of Artistic Style

E4040.2016Fall.GOGH.report
Daniel Kost-Stephenson dpk2124, Joshua Safyan jds2258, Robert Minnich rcm2164
Columbia University

Abstract

We present an implementation of "A Neural Algorithm of Artistic Style" by Leon A. Gatys et al. (2015). A comparison of results is given. Hardware and software limitations inhibited exploration of the hyperparameter and pre-processing space. We also discuss difficulties with L-BFGS CPU performance on larger images and an alternative Theano implementation of Adam on the GPU. Lastly, we consider the follow-on work "Combining Markov Random Fields and Convolutional Neural Networks for Image Synthesis" (Li and Wand, 2016). Using the methods described in Gatys et al., we achieve qualitatively similar results in style transfer but fail to achieve similar results for Li and Wand.

1. Introduction

Recently, artificial neural networks have been used to accomplish a multitude of tasks in computer vision. One such task is artistic style transfer from an image or painting (style image) to a second image (content image), that is usually a photograph. This concept was first demonstrated recently in "Image Style Transfer Using Convolutional Neural Networks" (Gatys et al. [1]) and has since inspired further style and content manipulations of images. The primary goal of this project is to review and reproduce the contents of the original paper, while comparing any differences in results.

By following the methodology of the original paper closely, we managed to obtain results that are visually appealing. Several challenges and difficulties were encountered, mainly with hyperparameter tuning and computational limitations. We overcame difficulties with hyperparameter tuning by simply using trial and error, but the computational requirements of our algorithm proved to be more of a hurdle. There were few differences overall, but we found that our selection of model hyperparameters varied significantly.

2. Summary of the Original Paper

2.1 Methodology of the Original Paper

Below, we outline the methodology found in the original paper in order to provide a basis of comparison.

The process of generating an image with content and style representations of the original two images is relatively straightforward. Both the content and style images are forward propagated through the pre-trained VGG-19 network to extract their activations at specific layers ('conv4_2' for content and 'conv1_1', 'conv2_1', 'conv3_1', 'conv4_1', 'conv5_1' for style). A third image, usually consisting of white noise, is then run through the same network in order to extract activations at the same layers. The loss between the synthesized image and the content/style image is then calculated as the sum of squared error between activations:

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

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

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

where alpha and beta are hyperparameters which dictate the contribution to the loss from the style and the content in the generated image. In addition,

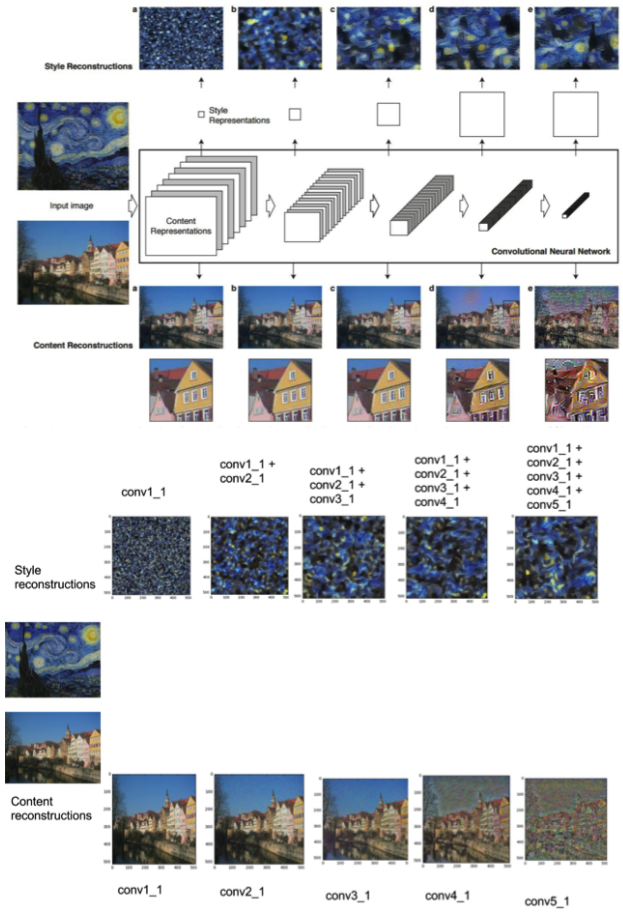
$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^{l} - A_{ij}^{l}\right)^2$$

and

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

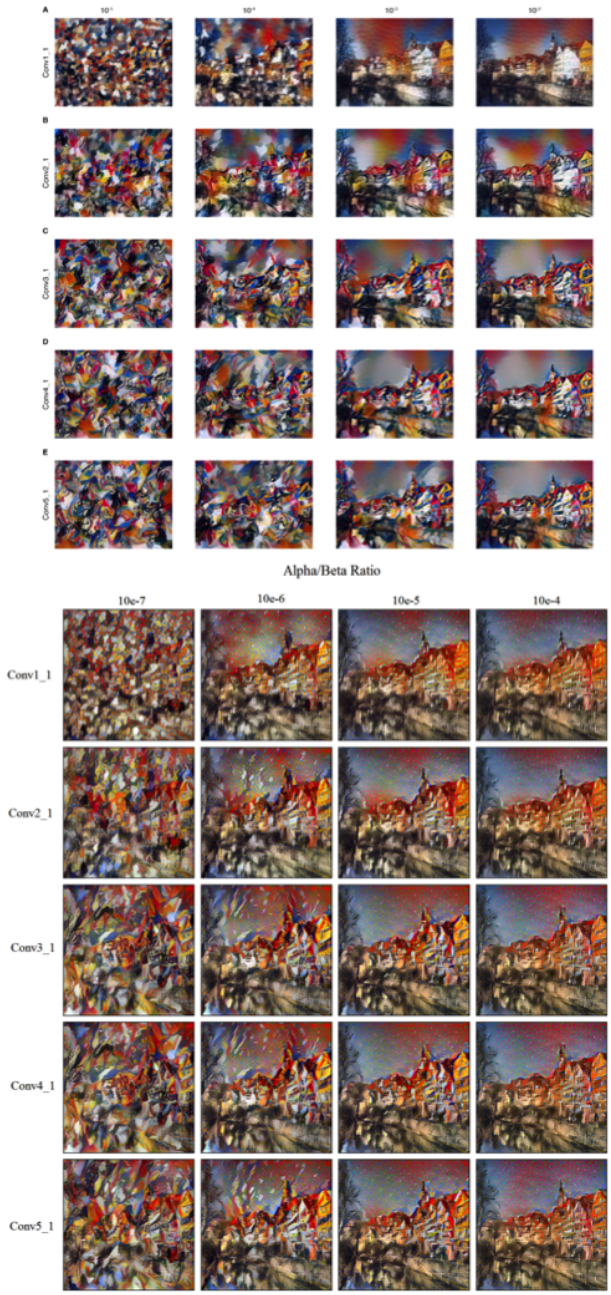
is the Gram matrix which measures correlation between features in a layer. The gradient of the total loss with respect to the synthesized image is calculated and subtracted from the image itself as an update step to minimize the total loss. The optimizer of choice is second order L-BFGS method which works well for images [2].

2.2 Key Results of the Original Paper



One of the key findings of the original paper is that style and content representations are independent and can therefore be used in conjunction to produce new images. Most of the other results are in the form of images synthesized by their methodology. Here, we present two relevant images presented in the original paper. Figure 1 shows the representation of a style and a content image at different convolutional layers in the VGG19 network. The content image is a picture of a waterfront in Tubingen, Germany and the style image is "The Starry Night" by Vincent Van Gogh.

Another key finding of the original paper is the ratio of the alpha and beta hyperparameters. Figure 2 shows a grid of content/style blends and the sensitivity of the model to its different hyperparameter ratios using "Composition VII" by Wassily Kandinsky as the style. The paper suggests an optimal ratio of 10^{-3} or 10^{-4}.



We can see from the figure above that we have very similar effects on the change in variables. There are differences within the grids but we can attribute this to the difference in the number of pixels between images. In later experiments we found that more pixels lead to better feature transfer.

3. Methodology

This section outlines the methodology used in our approach to generating images that blend content and style.

The methodology we employed was very similar to the original paper with the notable exception of the hyperparameter ratio. The paper recommends a ratio of 10^{-3} or 10^{-4}, but we found that ratios anywhere from 10^{-8} to 10^{-6} gave the most visually appealing results. Other than tuning the hyperparameters, our architecture is identical to the architecture outlined in the original paper.

3.1. Objectives and Technical Challenges

As outlined in the introduction, our objective was to reproduce the results of the original paper. Several technical challenges were encountered along the way, namely with hyperparameter tuning and the computational requirements associated with the task.

The adjustment of the alpha and beta hyperparameters were one of our biggest challenges. We noticed that an acceptable blend of content and style required a different alpha/beta ratio than the one outlined in the original paper. In addition, the only way to determine if a blend is appropriate is to generate an image and examine the results. On the hardware we had available, this took approximately 10-15 minutes, meaning we would have to generate a sample 224×224 to examine the results and determine if a particular blend was acceptable. This was made especially difficult because the hyperparameter ratio of content and style is dependent on the actual images themselves, meaning that different choices of content and style images require different ratios to obtain good results.

Additionally, the software used required a hybrid CPU/GPU approach. As recommended by the original paper, the L-BFGS algorithm was used to optimize the synthesized image but it was only available in Scipy, not Theano. This essentially neutralized the computational advantage of a GPU and led to a significant increase in running time but led to appealing results. We experimented with using first-order optimization methods such as Adam and RMSprop but found that these generally yielded poorer results.

3.2. Problem Formulation and Design

Our design to solve the given problem can be broken into several phases:

1) Build VGG-19 model using pre-trained weights
2) Image preprocessing
3) Relevant content/style layer extraction
4) Building theano functions to evaluate the loss and gradient of the loss with respect to the generated image
5) Using the created theano functions in the Scipy L-BFGS optimization
6) post-processing, plotting and evaluating results

We start off by building a VGG-19 class using pre-trained weights we obtained online [4]. The class is composed of convolutional and fully connected layers implemented in Theano.

During the image preprocessing phase, the loaded images for content and style are resized to a user-specified height and width, the means for each RGB layer are subtracted and a new axis is added to make the image 4 dimensions (for easy integration with Theano).

After the images are processed, a white noise image of the same dimensions is generated and the activations are extracted for the style and content images in the relevant layers using Theano functions.

Once the layers are extracted, the computational graph is built using the equations listed in section 2.1. Once the style loss and content loss are computed, we sum them to obtain the total loss. The gradient of the loss with respect to the generated image is then taken in order to apply a gradient descent technique. The gradients of the content and style respectively are given as

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

and

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

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

respectively.

After the graph is built, we create two theano functions that output the loss and gradient, then pass them into the L-BFGS function, along with our generated flattened image. The user selects the number of iterations, and we found that approximately 150 iterations is sufficient.

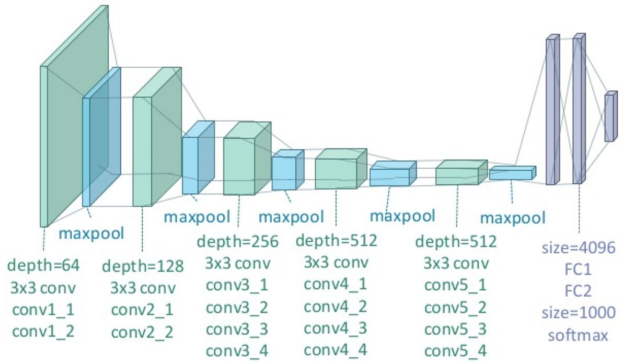
Once the optimization is complete, the image is post-processed by adding the means for each RGB layer and reshaping the image back to its original format of (224, 224, 3). Finally, the output is plotted and saved.

4. Implementation

This section will go into some detail on the VGG-19 network we used to extract style and content representations, as well as give more specific details on the pseudocode we used.

4.1. Deep Learning Network

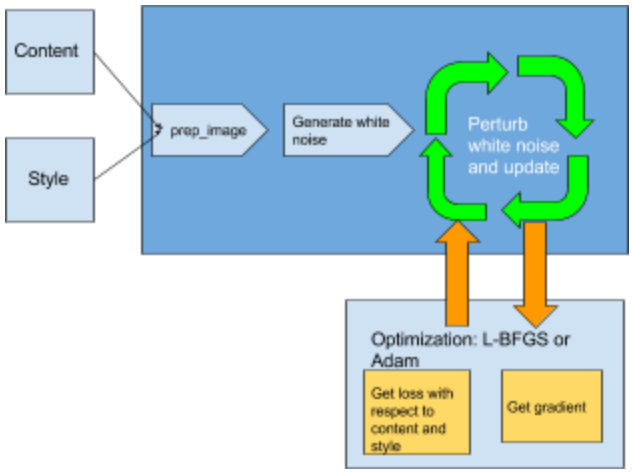
As mentioned earlier, the deep learning network used was the pre-trained VGG-19 network used to classify images. This network was trained on millions of images, and excels at image recognition, winning the 2014 ImageNet competition. This is useful for our application because we need the network to learn the content and style representations for the content and style images. The architectural block diagram is presented below.



The purpose of the original VGG19 model is classification, which is why the fully connected layers are present at the end. We are only interested in the convolutional layers so we can extract relevant activations at each layer.

4.2. Software Design

A high-level flow diagram of the software:



Algorithm:

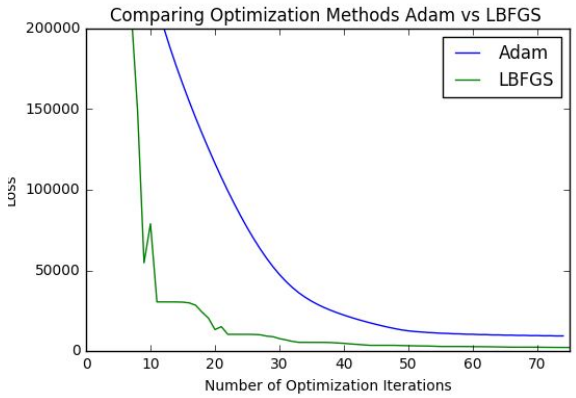
Style_activation := style image activations of style layers
Content_activation := content activation of content layer
generated_image := generate white noise image of same size as content image
Until some stopping condition do:

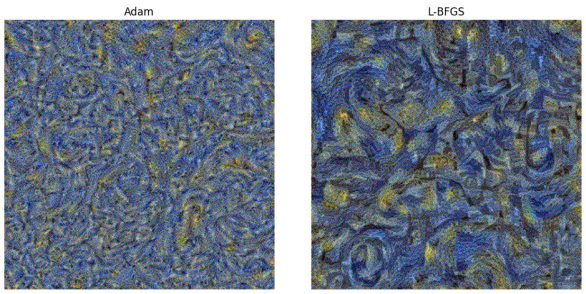
1. Run the generated_image through VGG-19 and extract the activations of the style layers and content layer.
2. Find the sum of squared difference between the activation of the content layer and the content image's activation of that layer
3. Find the gram matrix for the activations of the style layers, and find the squared error between the generated_image matrix and that of the style image.
4. Compute the total loss as a weighted average of the content and style losses
5. Find the gradient of the generated_image with respect to the loss
6. Perturb the generated_image using L-BFGS or Adam to minimize the loss

5. Results

5.1. Project Results

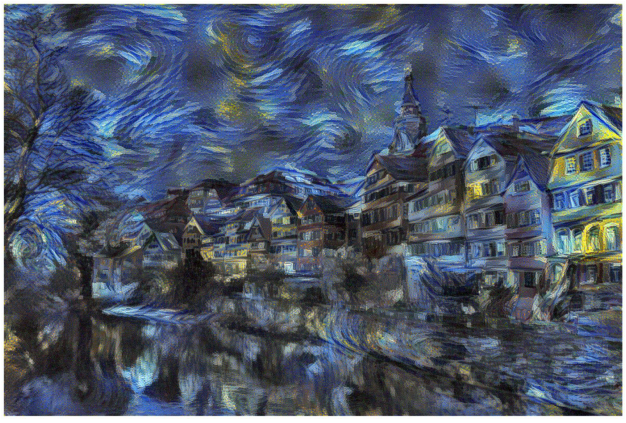
Experiments were also done using different optimization methods such as Adam. This was attempted in order to fully address the hardware/software challenges we were facing using L-BFGS.





While we were able to run Adam on the GPU, the L-BFGS seemed to give better results. Adam has the capability of achieving the same amount of loss, but this adds the additional challenge of finding the correct hyper parameters for Adam. The hyper parameters added additional complexity on top of the content/style ratio. Therefore we chose to use L-BFGS for our final results.

Using the implementation described above, we were able to obtain similar results to those displayed in the original paper. An example of the waterfront in Tubingen, Germany styled as Van Gogh's "The Starry Night" is shown below, using a hyperparameter ratio of 10^{-8}.



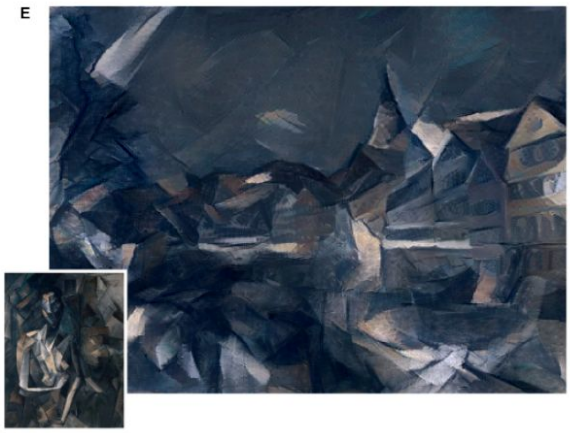
5.2. Comparison of Results

Using the same content/style combination as a basis of comparison, we can see that the only major difference between our synthesis and the synthesis in the original paper is the presence of more consistent texture in the sky and on the buildings.

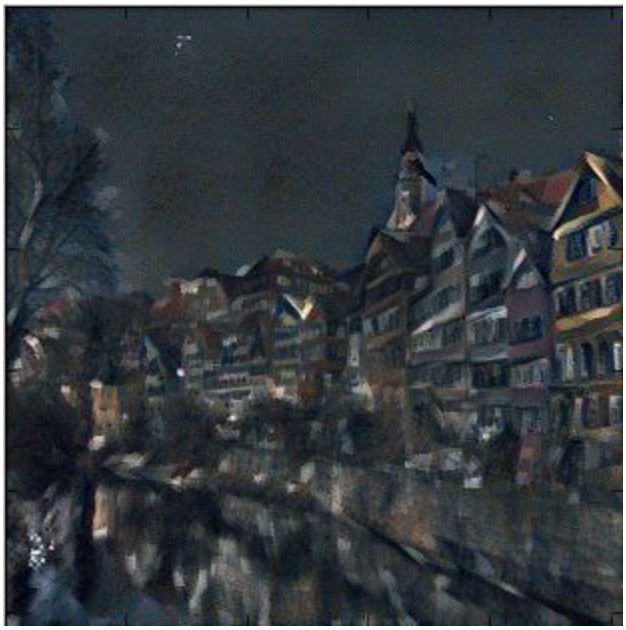
Gatys et al.

Ours

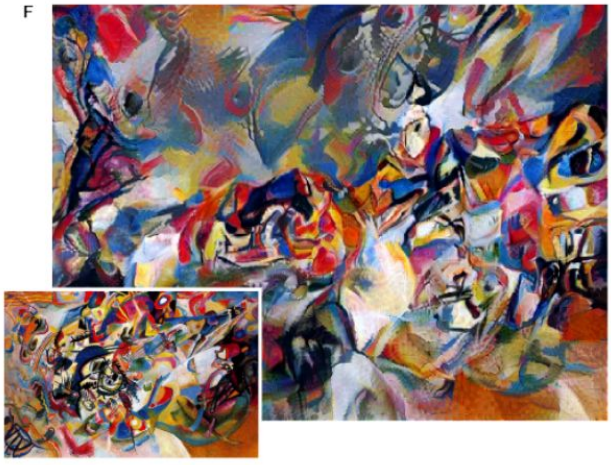
Gatys el al.



Ours



Gatys et al.



Ours

We find qualitatively similar results. Achieving the same blend of style vs. content proved difficult in some cases, as the content to style ratio changed depending on input images.

5.3. Discussion of Insights Gained

We believe that further investigation of some aspects of our model could lead to slightly more appealing results going forward. One major insight was that the higher the number of pixels lead to better feature transfer. The convolution by nature will suppress information within the image. By having a small number of pixels, less feature information is then transferred to the generated image and key features that distinctly define the style image can begin to disappear.

Additional fine tuning of the hyperparameters would certainly result in the optimal mixture of style and content, in particular for Adam. Perhaps adding a form of regularization in the cost function would help smooth the result and make the synthesized image less granular. One other important factor we didn't investigate was initialization of the generated image. Since this is an optimization problem, we expect the initialization to have a big impact on whether our optimizer will actually find a global minimum. Lastly, we believe zooming in on a specific area of the style image might improve the overall consistency of the generated image and lead to less dark patches where style seems to be less present.

6. Conclusion

Overall, we managed to accurately reproduce the contents of the original image and drew several key conclusions, based on our results. Firstly, we observed that the choice of hyperparameters greatly impacts the quality of the result. We also noticed that the choice of solver has a big impact on the result and that L-BFGS seemed to converge more quickly than Adam. Going forward, several modifications could be made to potentially yield better results, including regularization and zooming in on the style image. This fine tuning of the model should be looked into so that the visual appeal of this model approaches that of the original.

6. Acknowledgement

Provide acknowledgement if needed, such as support, help, or assistance from someone. These support, help, assistance are crucial.

7. References

Include all references - papers, code, links, books.

[1] https://bitbucket.org/e_4040_ta/e4040_project_gogh

[2] A Neural Algorithm of Artistic Style Leon A. Gatys, Alexander S. Ecker, Matthias Bethge https://arxiv.org/abs/1508.06576

[3]Karen Simonyan, Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition: http://arxiv.org/pdf/1409.1556

[4]Lorenzo Baraldi Keras Model to extract VGG19 LayerWeights:https://gist.github.com/baraldilorenzo/8d096f48a1be4a2d660d

[5] Applied Deep Learning 11/03 Convolutional Neural Networks: http://www.slideshare.net/ckmarkohchang/applied-deep-learning-1103-convolutional-neural-networks

8. Appendix

8.1 Individual student contributions - table

	dpk2124	jds2258	rcm2164
Last Name	Kost-Stephenson	Safyan	Minnich
Fraction of (useful) total contribution	1/3	1/3	1/3
All group members contributed equally to all aspects of the project.			

Combining Markov Random Fields and Convolutional Neural Networks for Image Synthesis

1. Introduction

This paper is identical in the architecture, optimization and content loss as the paper mentioned above, with the difference of a modified style loss function and the addition of a smoothing prior. The goal of this paper is to focus on utilizing local features within a style image and the generated image, which will allow for better blending of content and style. This is especially useful when trying to blend multiple images.

In contrast to the previous method, which used features based on image location, this method has the ability to detect similar features between images and generate more fluid images. This is done using the style loss below:

$$E_s(\Phi(\mathbf{x}), \Phi(\mathbf{x}_s)) = \sum_{i=1}^{m} ||\Psi_i(\Phi(\mathbf{x})) - \Psi_{NN(i)}(\Phi(\mathbf{x}_s))||^2$$

$$NN(i) := \underset{j=1,\ldots,m_s}{\arg\min} \frac{\Psi_i(\Phi(\mathbf{x})) \cdot \Psi_j(\Phi(\mathbf{x}_s))}{|\Psi_i(\Phi(\mathbf{x}))| \cdot |\Psi_j(\Phi(\mathbf{x}_s))|}$$

For a given style layer, the authors split the activations of the generated image and the activations of the style image into "neural patches." They find the sum of squared norms between each of the activation patches from the generated image, and that patch's nearest neighbor among the activations of the style image. The nearest neighbor equation is given above. In the implementation section of the paper the authors describe their method for computing the nearest neighbor. They use a convolution of the generated image patches with the style image patches as filters, and select the style patch that yields the maximum activation.

The authors demonstrate style transfer similar to that presented by Gatys et al. in addition to image synthesis. For the former, they present qualitatively superior results for images with analogous features (e.g., a painting of a face for style and a photo of a face for content). They note that without analogous features their method produces inferior results for style transfer compared to Gatys et al.

2. Implementation

The challenge with this method is that it requires a brute force method of determining nearest neighbor neural patches. For large images this becomes an extremely difficult task in terms of running time. We initially attempted to run this method with similar parameters as the paper, but this proved to take too long with the amount of time left to turn in the report.

We then adjusted our approach, by limiting the size of the image, increasing the strides of the convolution, and doing a single dot product to achieve a single grid, from which we could determine the Nearest Neighbors. This method consumes more memory, and ultimately did not allow us the ability to capture meaningful feature information.

3. Conclusion

Overall the method of this paper was to allow more local features using Nearest Neighbor feature selection before calculating the style loss. This method is not ideal however because of the $O(n^2)$ run time that is required in order to determine the Nearest Neighbor. This combined with an L-BFGS method that does not utilize a GPU, did not give us enough time to implement the full method. After attempting to work around this by utilizing smaller images and features, we were not able to get results similar to the paper.