Neural Style Transfer with a VGG16 Convolutional Network

Abdiqani Ainab
Red ID: 819415799
San Diego State University
abdiqaniainab@gmail.com

Introduction

If you've ever used the popular mobile application Snapchat, then you've probably used their famous dog filter or the whimsical eyes popping out of your head one. If you're and avid user of snapchat then you've probably used the paintbrush tool. The paintbrush tool allows the user to take a photo or video and apply an artistic style of their choosing from a pre-determined available style list.

That method in which snapchat seemingly allows the user to become their favorite art style is called a "Neural Style Transfer". The Neural Style Transfer is an optimization technique in which you take two images. The first image is called the "content image" and that's usually whatever photo you'd like to turn into art. The second is referred to as the "style image" which is the desired art style that you'd like to apply to the content image. Then you mix both images together so that your final output image is a blend of your original content image done in the artistic style of your style reference image. In this paper I'll explain to you the process in which I created and implemented my own neural style transfer using an already pre trained convolutional network as well as my results. The rest of the paper is as follows. In section one I'll provide a brief overview on Gatys et Al findings in their report "*A Neural Algorithm of Artistic Style"*. Then in section 2. I'll go briefly go over the convolutional neural network I employed. In Section 3 I'll explain how I manipulated the loss functions with VGG16 model while following both Gatys et Al paper and Johnson et Al. paper. Section 4 is dedicated to the actual optimization problem for neural style transfer. Section 5

1. The Gatys Paper

The Gatys paper came about after people were looking for better ways of extracting content and texture of images in order to get a better result. Gatys et Al. came up with the

idea of using a deep neural network to achieve the desired result and after testing their hypothesis they introduced their article on it in 2015. Their paper boiled down to 3 key parts: extracting the content, extracting the style and then blending the image. Gatys and his team used a pre-trained VGG19 neural net and made some changes to the weights so it would adapt to their task. Then used the output of one of the hidden layers as its content extractor. Then the used the same tactic in order to extract the style from their image however they added one more step that was based on the gram matrix of the filters of the given hidden layer. Lastly they used an optimization problem in order to blend the content of content image with artistic style of the style image.

1. Model

Gatys et al. introduced the idea that convolutional neural networks that are pre trained for image classification are already equipped with the knowledge on how to encode semantic and perceptual information about images. In their paper they used the VGG19 neural network model but after some research I found it best to use the VGG16 model instead. The VGG16 model offers slightly more speed. Since I wasn't using the model for image classification as it was intended; I ended up not needing the FC layers or the final soft-max classifier.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

figure. parts of the model that is valid for use.

```
{'block1_conv1': <tf.Tensor 'Relu:0' shape=(3, 512, 512, 6
4) dtype=float32>,
 'block1_conv2': <tf.Tensor 'Relu_1:0' shape=(3, 512, 512,
64) dtype=float32>,
 'block1_pool': <tf.Tensor 'MaxPool:0' shape=(3, 256, 256,
64) dtype=float32>,
 'block2_conv1': <tf.Tensor 'Relu_2:0' shape=(3, 256, 256,
128) dtype=float32>,
 'block2_conv2': <tf.Tensor 'Relu_3:0' shape=(3, 256, 256,
128) dtype=float32>,
 'block2_pool': <tf.Tensor 'MaxPool_1:0' shape=(3, 128, 12
8, 128) dtype=float32>,
 'block3_conv1': <tf.Tensor 'Relu_4:0' shape=(3, 128, 128,
256) dtype=float32>,
 'block3_conv2': <tf.Tensor 'Relu_5:0' shape=(3, 128, 128,
256) dtype=float32>,
 'block3_conv3': <tf.Tensor 'Relu_6:0' shape=(3, 128, 128,
256) dtype=float32>,
 'block3_pool': <tf.Tensor 'MaxPool_2:0' shape=(3, 64, 64,
256) dtype=float32>,
 'block4_conv1': <tf.Tensor 'Relu_7:0' shape=(3, 64, 64, 51
2) dtype=float32>,
 'block4_conv2': <tf.Tensor 'Relu_8:0' shape=(3, 64, 64, 51
2) dtype=float32>,
 'block4_conv3': <tf.Tensor 'Relu_9:0' shape=(3, 64, 64, 51
2) dtype=float32>,
 'block4_pool': <tf.Tensor 'MaxPool_3:0' shape=(3, 32, 32,
512) dtype=float32>,
 'block5_conv1': <tf.Tensor 'Relu_10:0' shape=(3, 32, 32, 5
12) dtype=float32>,
 'block5_conv2': <tf.Tensor 'Relu_11:0' shape=(3, 32, 32, 5
12) dtype=float32>,
 'block5_conv3': <tf.Tensor 'Relu_12:0' shape=(3, 32, 32, 5
12) dtype=float32>,
 'block5_pool': <tf.Tensor 'MaxPool_4:0' shape=(3, 16, 16,
512) dtype=float32>,
 'input_1': <tf.Tensor 'concat:0' shape=(3, 512, 512, 3) dt
ype=float32>}
```

List of all the valid VGG layers

2. The Loss Function

As explained earlier in section 1. Gatys et al loss function minimization cam be broken up into three parts content loss, style loss, and Denoising otherwise known as the total variation loss.

3.1 Content Loss

In the Gatys et al paper, the team drew the content feature from block4-conv2 in the VGG19 model. However, after reading the Johnson et al article "*Perceptual Losses for Real-Time Style Transfer and Super-Resolution"*. I decided to use block2_conv2 because the block4_conv2 lost too much detail. The content loss is scaled squared Euclidean distance between feature representations of the content and combined

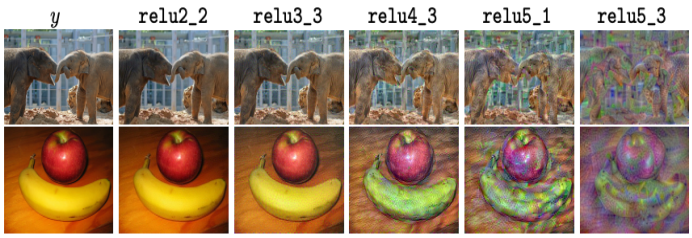


figure on variation between different layers in the model.

3.2 Style Loss

The first part of the style loss problem is defining the Gram Matrix. The gram matrix captures information about which features tend to activate together. By solely gathering those aggregate statistics across the style image its blind to specific arrangement of objects inside the image. This means that it allows me to take the artistic style of the image without any of the content in the image.

The style loss becomes the scaled squared Euclidean norm of the difference between the gram matrices of the style and mixed image.

3.3 Denoising

The Denoising function is exactly what it seems. It allowed me to smooth the output

image and by reducing the noise inside the image.

4. Neural Style

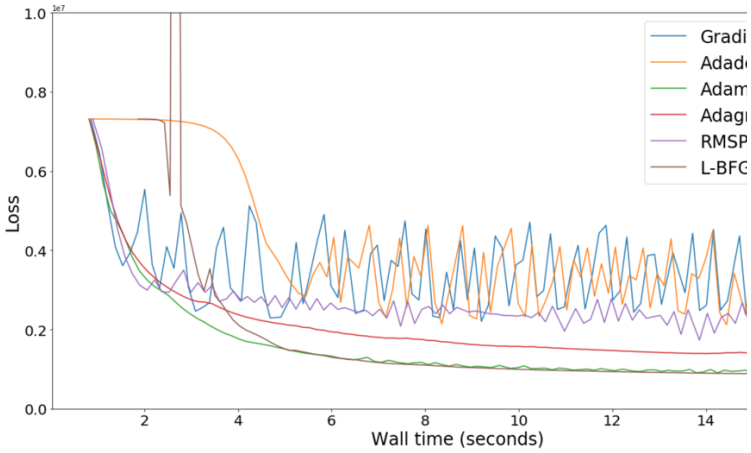
After computing the artistic style of the style image I now needed to employ an optimization problem to solve for the mixed image. The first step now was to define the gradients of the total loss relative to the mixed image. I then used said gradients to improve on our mixed image iteratively to minimize the loss.

After I defined our gradients it was time to compute the loss and gradients on a single pass while receiving it from two different functions due to the fact that scipy.optimize requires separate functions for loss and gradients.

After the loss and gradient functions were computed. My mixed image starts off as a random collection of valid pixels and then I used the L-BFGS(Limited- memory Broyden Fletcher Goldfarb Shannon) algorithm to iteratively improve on it because its quicker to converge than gradient descent. I ended up stopping after 20 iterations because the output looks descent and also because my laptop ends up crashing if I leave it unattended. I'm sure if I set it at 100 iterations it'd look much smoother.

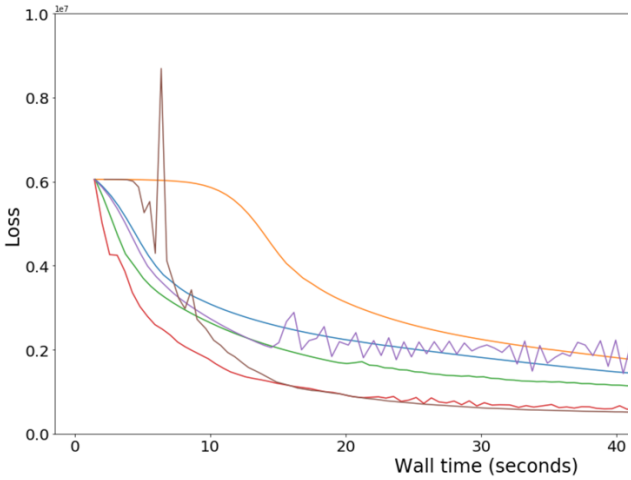
5. Algorithm Evaluation

Although I didn't increase my number of iterations past 20 I found that even at 20 iterations the L-BFS implementation was still pretty good at producing a mixed image with minimal loss at lower iterations than other optimizing algorithms. I heavily relied on Slav Ivanovs findings on how to choose your optimizer for neural style transfer. Here are some of his findings below.



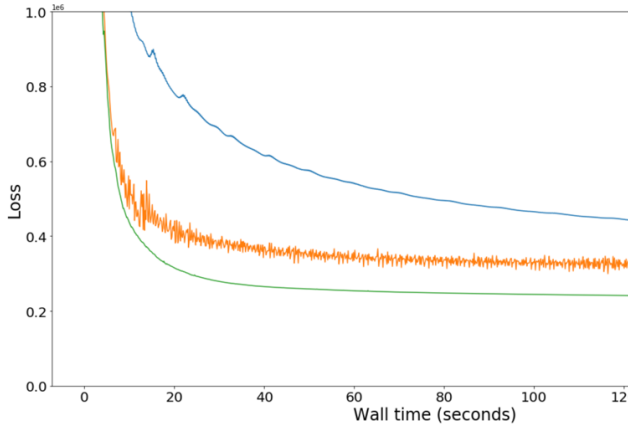
Here is an example of L-BFGS running for 100 iterations at 300 x 300 pixel size as you can see it converges fast with the Adam algorithm

For this second figure the optimization algorithms are running at 100 iterations for a larger number of parameters

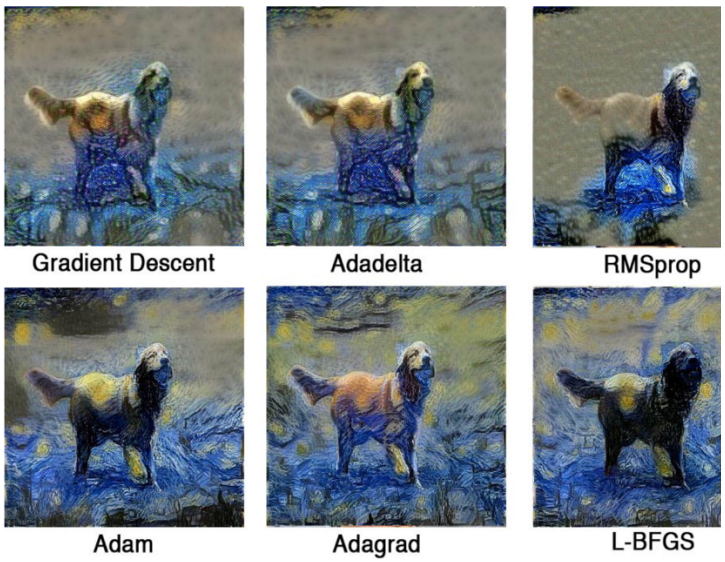


This time the Adam algorithm converges faster but the L-BFGS optimization achieves less loss over time

For the third comparison he only used the top three performing algorithms and ran them at 1000 iterations at a 300 x 300 size

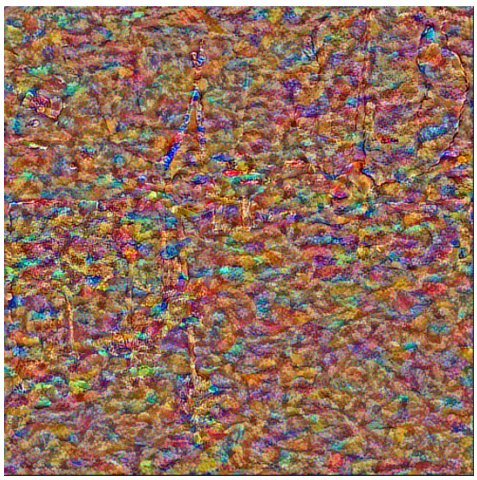


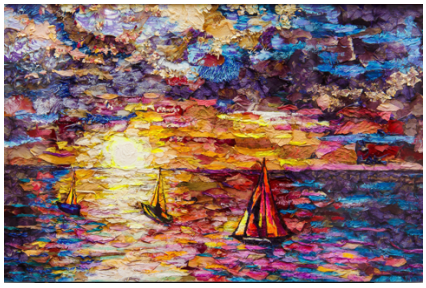
The L-BFGS algorithm actually performed faster than both algorithms and achieved 50% less loss than them.



Out of all the optimization algorithms I believed in the L-BFGS algorithm the most it seemed to truly blend both content and style images and that's what I hoped to achieve with my work. Although I couldn't run my Neural Style Transfer for 1000 iterations I ran it for 20 and was able to achieve a blended result.

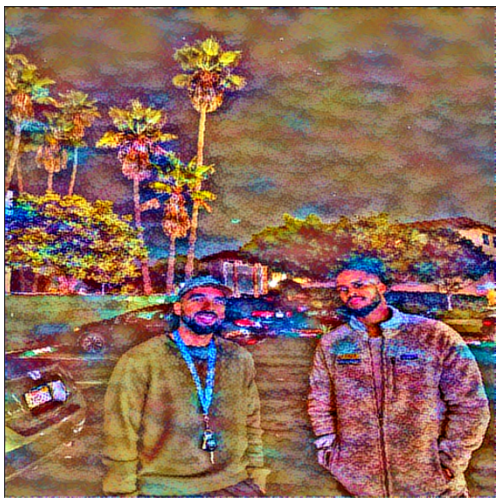
The content and style images





Blended Image after 1 iteration

Blended Image after 5 iterations:



Blended Image after 10 Iterations:

Blended image after 20 Iterations:

6. Conclusion

After 20 iterations took more than an hour I decided that it would be best to stop my testing there. It is possible that a faster optimization algorithm would be able to produce better results in less iterations but that would require a better laptop with stronger GPU which is something that o didn't have available to me.

References:

L.A. Gatys, A Neural Algorithm of Artistic Style, (2015).

Neural style transfer : TensorFlow Core. *TensorFlow*. https://www.tensorflow.org/tutorials/generative/style_transfer. Accessed 20 December 2019

Kausar, A. (2019, April 24). Neural Style Transfer-Remixing Visual Concepts. *Medium*. Red Buffer. https://medium.com/red-buffer/neural-style-transfer-remixing-visual-concepts-b2ff98f69ea. Accessed 20 December 2019

How Do Neural Style Transfers Work? *By Ayush Singh*. https://hackernoon.com/how-do-neural-style-transfers-work-7bedaee0559a. Accessed 20 December 2019

J. Johnson, A. Alahi, L. Fei-Fei, Perceptual Losses for Real-Time Style Transfer and Super-Resolution, (n.d.).

Ivanov, S. (2017, June 26). Picking an optimizer for Style Transfer. *Medium*. Slav. https://blog.slavv.com/picking-an-optimizer-for-style-transfer-86e7b8cba84b. Accessed 20 December 2019