Style Transfer

Zesheng Liu, Mateo Parrado

2022/08/05

Abstract

In this work we experiment with multiple different strategies for neural based style transfer on both videos and images. We follow previously published methodology proposed by Gatys et.al while attempting to optimize the output, trying a variety of different networks for feature extraction, different optimizers different loss functions and different regularization terms to produce the ideal output. We find that style transfer also need tine tuning for different inputs. The choice of layer representation for style and content, the choice of loss function, the choice of optimizer can somehow affect the final results while the choice of pretrained model can slightly affect the generated image. Besides, adding several regularization term can make us able to get better results. In the last, we try to extend our video on style transfer for video.

1 Introduction

With the advancement of internet technology the demand to produce not only high quality content, but to produce it rapidly is drastically increasing. Feature films that took years to edit and produce are being replaced with short Tik Toks that can be churned out rapidly and repeatedly. Users expect to be able to edit photos on their phones immediately and easily. Both of these applications place a strong emphasis on filters to improve the perceived quality of the image without spending more time on lighting and editing. Most of these filters are fairly basic, increasing exposure or decreasing contrast, but our project asks what if we took this a step further? What if instead of simple image manipulations we could transfer the style from one image to another? This would allow us to change the mood, the time period, and the even the content of an image to more accurately fit the goals of our end user.

Our project uses two images (or an image and a video). The first image is the style image. Our neural network will transfer the features and colors from this image into the other image. The second image is the content image. This image will be manipulated to have a more similar appearance to the style image while attempting to ensure that the original content of the image is preserved.

In addition to simple style transfer we will be implementing a few additional capabilities. The first is handling large images, allowing high resolution style transfer. The second is style transfer for short videos, which could be applied to things such as live photos.

2 Related Works

The Neural Style Transfer is proposed by Gatys et.al.[1] In the paper, the author take the output feature maps in the pretrained VGG-19 model, and use the Gram matrix to represent the style of a image.

The author thinks that the loss of the generated image is a combination of the content loss and the style loss,i.e. $Loss = \alpha Loss_{content} + \beta Loss_{style}$. The choice of α, β refer to preferences that whether we want it be more similar to inputs in content or style. They use the L-BFGS method to generate the final output that minimize the total loss. This method will typically need several minutes to be done.

After that, in order to make the style transfer faster, Justin Johnson et.al proposed a GAN-like model which can make the time for generalization much faster[2]. They introduced a image transform net that can be trained to generated images and a loss network which is a pretrained VGG-16 network used to calculate the style and content loss. This is like how we are training a GAN model as the image transform net can be seen as a Generator and the loss network can be seen as a Discriminator. After training the model, the step of generating output will only be one step of generation which is pretty fast.

Fujun Luan et.al proposed another kind of modification to the Neural style transfer to keep the output to be photo-realistic.[3] They introduce a relugarization term to the original loss function and modified the style loss by introuduce the use of semantic segmentation method and index matching to make sure that the network will learn the color of 2 related objects in the input, i.e. matching the sky in the style input with the sky in the content input and learn there color relations.

Huang et. al propose a solution to maintain more consistency between video frames and ensure continuity [4]. They propose modifying the model outlined above by adding a third kind of loss called temporal loss. This loss function will try

to minimize the difference between consecutive frames. Because most videos are fairly high frame rate we can expect there to be little change between the content of each frame, and therefore we will want to keep the style as consistent as we can.

3 Methods

Our architecture starts by using a pretrained model. We found the 19 layer model created by Visual Geometry Group (VGG19) worked the best for our purposes, and therefore used it for most of our testing. All of the pretrained models we experimented with were primarily designed for image classification.

We follow the style transfer proposed by Gatys et.al[1] and modified these pretrained models by adding a few loss functions to them. The first loss function is the content loss. This loss checks the difference between the generated image and the input image and attempts to minimize it. The main goal of this process is to ensure we output an image that is still recognizable as the input image. We want to ensure that most objects remain and that we are not simply generating a completely new image. The second form of loss we are adding is called style loss. This loss measures the similarity between the features of the generated image and the style image. Note that this function uses a feature based comparison, as opposed to the pixel based comparison we used in the content loss. In order to achieve this we use a so called "Gram Matrix." The simplest way to compare two feature vectors is to take the dot product between them. The Gram essentially does just that. It takes the dot product between every feature in the source and generated image. We then attempt to make these features as similar as possible.

In our project, we do several experiments based on this model. We use different loss function, which in the original paper the author uses MSE loss. We use different pretrained model, and different optimization method to get the best result. Also, follow the idea mentioned in Johnson et.al[2], we introduce a total variation regularizer to increase the spatial smoothness in the output image, which is defined as the sum of absolute value of the difference in nearby pixels, i.e. $\sum_{i,j} |x_{i,j} - x_{i+1,j}| + |x_{i,j} - x_{i,j+1}|$. We also introduced the photorealistic regularizer methoded by Fujun Luan et.al[3] to preserve the structure and make it photorealistic. It uses the Matting Laplacian of the original content image times the vectorized output image at every epoch, which defined as $L_m = \sum_{i=1}^{3} V_c[O]^T M_I V_c[O]$, where M_I is the Matting Laplacian of input image and $V_c[O]$ is the vectorized version of output image. In the last, in order to generate a large output, we use the idea of homography.

For video transfer we only make slight modifications to the model. The first modification is that we add a new loss called "temporal loss" as suggested by Huang et al. [4]. Temporal loss is the difference between two consecutive frames. The idea behind this is that we want to try to ensure as much consistency as possible between frames to make sure that no stylistic artefacts are jumping around as we watch our video. We apply this temporal loss on all the same layers that we apply our content loss because the two are closely related. We made a few modifications to the training process as well. Instead of training on the entire video at once we train the model repeatedly in small chunks. We then average the results of these chunks to use as our video output.

4 Experiment and Results

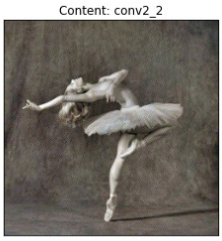
4.1 Image Style Transfer

4.1.1 Layer Representation of VGG-19

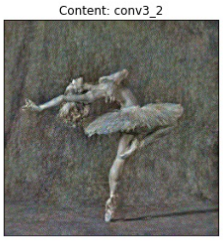
At the beginning, we initial the output image by some gaussian noise and set either the style weight or the content weight be 0, to see how the layer output of a VGG-19 model represents the style and content.

In the original paper, Gatys et.al choose the layer 'Conv4_2' to be the content layers. We compared its representation with other layers.

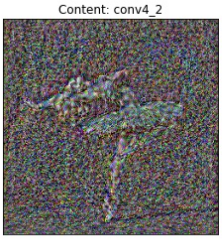
(a) Content Represents: Conv1



(b) Content Represents: Conv2



(c) Content Represents: Conv3



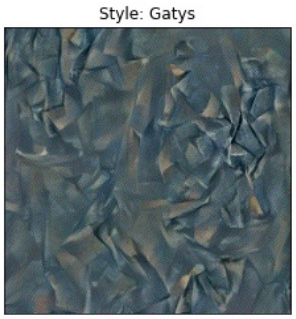
(d) Content Represents: Conv4

(e) Content Represents: Conv5

FIGURE 1: Content Representation of different choice

From Fig.1, we can see that if we choose a deeper layer to be the content layer, our model will tend to learn what the content is rather than learn exactly what the pixel values is. 'Conv4_2' will either make our model be able to learn the content and make our model not focus on learning the exact pixel values.

(a) Style Representation of Pytorch Tutorial Choice



(b) Style Representation of Gaty et.al

FIGURE 2: Style Representation of different choice

We find that in the Pytorch Tutorial, they choose different style layers. Then we set the content weight equals to 0 and see what style output our model will generate based on the choice of style layers by Gatys et.al and the Pytorch Tutorial. In Fig.2, we can see that the style representation of the pytorch tutorial choice is only some repeated features, as they choose the first 5 layers to be the style layers. While in the original paper, Gatys et.al choose several deep layer which ensures the model can learn the style batter and therefore generated style output will more close to the style input.

4.1.2 Style Content Tradeoff

By control the style weight and content weight in our loss function, we can control whether the generated image is more closer to the content image or more closer to the style image.

(a) Content Weight:1 (b) Content Weight:10 (c) Content Weight:100

(d) Content Weight:1000 (e) Content Weight:10000 (f) Content Weight:100000

FIGURE 3: Style Content Tradeoff: Fixed style weight

(a) Style Weight: 1 (b) Style Weight: 10 (c) Style Weight: 100

(d) Style Weight: 1000 (e) Style Weight: 10000 (f) Style Weight: 100000

(g) Style Weight: 10^7 (h) Style Weight: 10^8 (i) Style Weight: 10^9

FIGURE 4: Style Content TradeOff: Fixed content weight

From Fig.3 and Fig.4, we can see how the output are chaning by controlling different style weight and content weight.

As our task is doing style transfer, we want the output to be both closer to the content image and closer to the style image. Therefore, we find that if we set style weight be $10^5/10^6$ and content weight to be 1, the generated image is the best one.

4.1.3 Different Optimization Method

We compared the effect of different optimization method. In the paper, they use the L-BFGS method. Here due to the fact that JAX framework don't have a good L-BFGS optimizer, we will compare the results between AdaGrad, Adam, SGD and RmsProp with learning rate to be 10^{-2} and 10^{-3}.

(a) RmsProp, lr=10^{-2} (b) SGD, lr=10^{-2} (c) Adam, lr=10^{-2} (d) AdaGrad, lr=10^{-2}

(e) RmsProp, lr=10^{-3} (f) SGD, lr=10^{-3} (g) Adam, lr=10^{-3} (h) AdaGrad, lr=10^{-3}

FIGURE 5: Output Image With Different Optimization Methods

From Fig.5 we can find that RmsProp with lr=10^{-2} will create some strange noise to the output. Due to this reason, it is better not to use RmsProp as the optimizer for style transfer. For the rest three methods, AdaGrad converge very slow even with lr=10^{-2}. Therefore, if we want to use it as the optimization method, we need to train a longer time, and in that case, it is not the best choice. For Adam and SGD, it will both generate pretty good output. We should notice that SGD with lr=10^{-3} also converge slower. In conclusion, we find that the L-BFGS mentioned in Gatys et.al work[1] is not the only optimization method that can do style transfer. Both Adam and SGD will also works well if you fine tuning the learning rate and the training epochs. Probably one advantage of using L-BFGS is that we don't need a learning rate for it, which can reduce our work when training the style transfer.

4.1.4 Different Loss Function

We In the original paper, Gatys et.al uses the MSE loss[1], which is L2 loss. Here we compared it with different loss functions like L1 loss, smooth L1 Loss and Huber Loss.

(a) MSE Loss (b) L1 Loss (c) Smooth L1 Loss (d) Huber Loss

FIGURE 6: Output Image With Different Loss Functions

From Fig.6, we can see that for this certain style input and content input, all the four kinds of loss functions seems to working properly and can give out good outputs. For this certain input, I will prefer the L1 Loss as it can add some style

color on the dancer's white dress while the other will keep it almost white. For other style input and content inputs, we may need to do some fine tuning with other parameters like the choice of optimizer and the choices of layers to find out the best loss function.

4.1.5 Different Pretrained Models

Gatys et.al uses the pretrained VGG-19 in their paper[1]. We compared it with VGG-16 and ResNet to see if the output will getting better.

(a) VGG-19 (b) VGG-16

(c) ResNet18 (d) ResNet34 (e) ResNet50

FIGURE 7: Output Image With Different Pretrained Models

From Fig.7 we can see that if we use pretrained VGG models we can get better generated output compared with ResNet model. Therefore, we can say that for our task, VGG model can do better feature extraction compared with ResNet model. If we must use ResNet model, we may consider to use ResNet 18 or ResNet 34, which is slightly better than ResNet 50. For this input style and content, we don't have a obvious difference between the output generated by VGG-19 and VGG-16. We also notice that due to the skip connection feature in ResNet models, it will runs slightly faster than VGG models.

4.1.6 Apply Regularization Term

As mentioned before, we introduce the total variation regularization and the photorealistic realization. By adding these regularization terms, we can make our output image more photorealistic.

(a) Original Content Image (b) No Regularization Term

(c) PhotoRealistic Regularization (d) PhotoRealistic + Total Variation

FIGURE 8: Regularization Term

From Fig.8, we can see that our regularization term can improve the output. Compared with the output with no regularization term, PhotoRealistic + Total Variation can improve the quality of the output image especially on the trunk of the tree. It looks more photorealistic and more similar to the original content, rather than getting red like the style colors.

4.1.7 Larger Image

In the last, we work on improve the runtime of our code and try to work with large scale images.

As we are using JAX framework, it is not as fast as the Pytorch version as JAX is a pretty new framework and are not being fully optimized like Pytorch.

@jit is a command for just in time compile. It can compile and run your code in a faster mode like C/C++. If we need to call a function several times in our code, by using this command we can speed it up. Besides, writing the code in the format of class and avoid calculating same thing several times, we can also speed up our code and make it more readable. On Colab, if we don't use these tricks, it will take 1min8s to train the style transfer 500 epochs with Adam optimization. If we add these tricks, 2000 iterations will only need 1min10s.

In order to working with larger images and avoid out of memory error, we use the idea of homography and image stitching.

(a) Left Side (b) Right Side

(c) Stitched Image (d) Whole Image Output

FIGURE 9: Image Stitching For Larger Output

In Fig.9, we divide a larger content image into two patches, which is the left side and the right side. We split the style image by a similar way, and train the style transfer on these two patches. After that, we use the method of image stitching to connect these two patches together and get the output for the whole image. We can see it is almost the same as we directly train the style transfer on the whole input content image. Therefore, if the image become really large, we can consider to divide it into several patches and train them individually, then we can connect them back for the final output.

4.2 Video Style Transfer

For video transfer I recorded a short video of myself dancing in my webcam10.

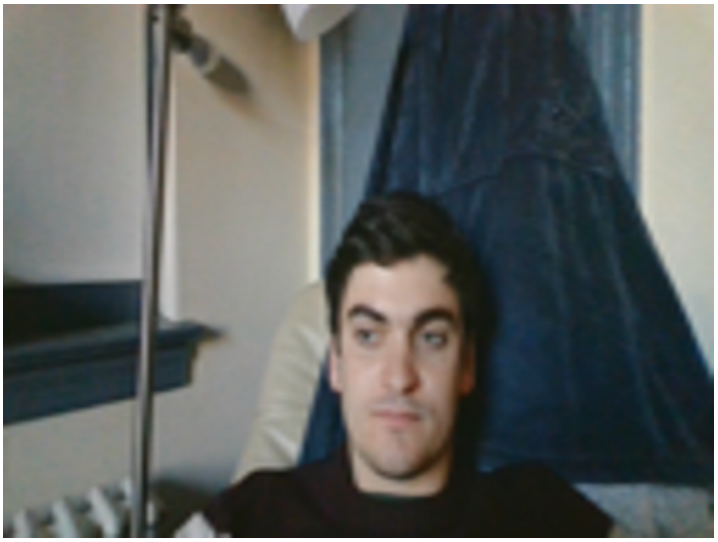
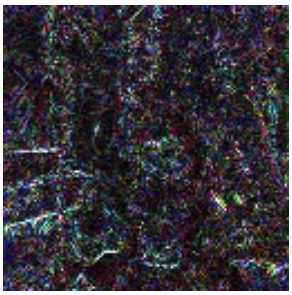
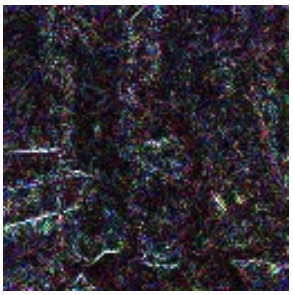


FIGURE 10: This is the first frame of the training video

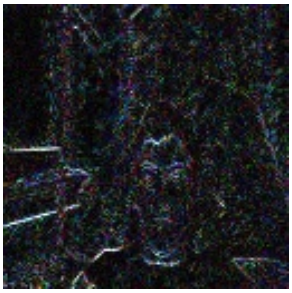
We ran multiple experiments to fine tune the results of video transfer. The first experiment was to play around with the temporal weight. The higher the temporal weight the lower the differences should be between frames. Below we have the the differences between two frames in the video. We can see that the videos with higher temporal weighting have much fewer differences, and therefore produce much more consistent videos11.



(a) Difference between consecutive frames with temporal weight of 1



(b) Difference between consecutive frames with temporal weight of 1,000



(c) Difference between consecutive frames with temporal weight of 1,000,000

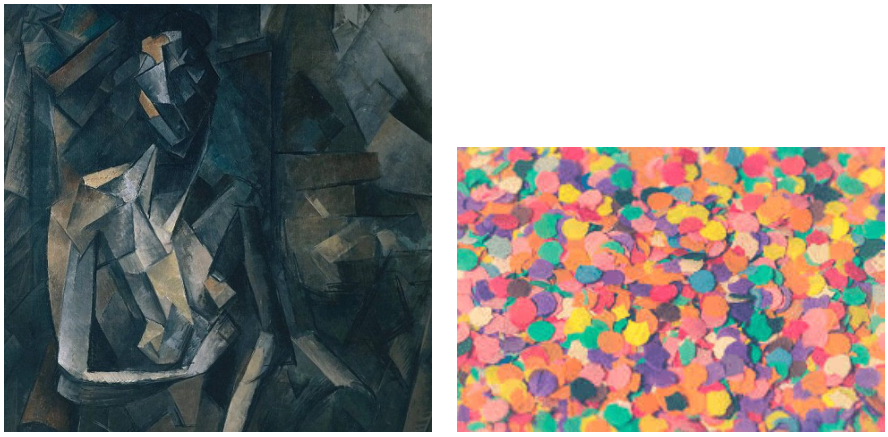
(d) Difference between consecutive frames with temporal weight of 1,000,000,000

FIGURE 11: Loss between frames

However, the higher the temporal weight the less style transfer seems to take place. This makes sense because the easiest way for the neural network to ensure minimal changes between frames is to simply not apply new style. This suggests that we need to maintain a balance between the style and the temporal weights (just like we found above for the content weight).

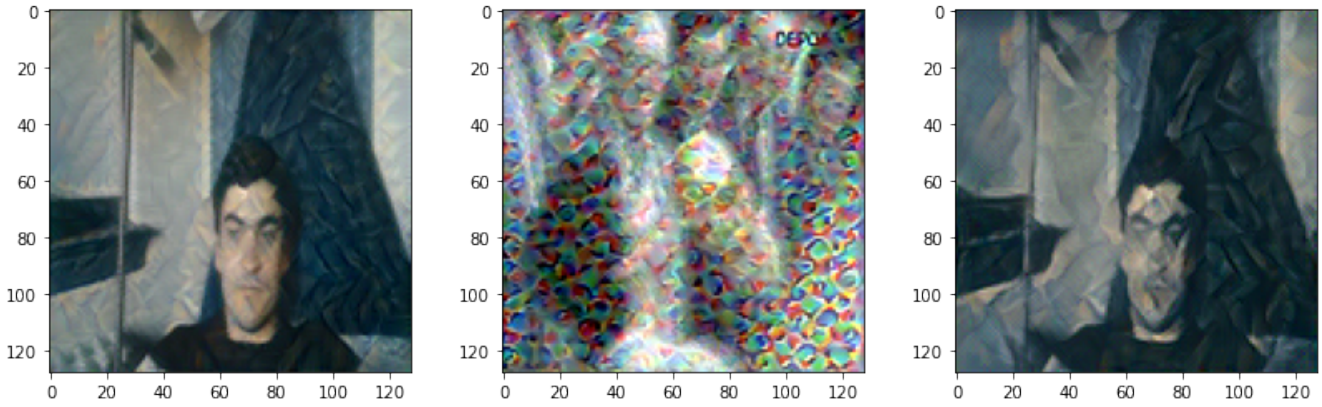
Our biggest issue with videos was the processing power required. iPhones shoot video at 30 frames per second, so style transfer for even a 5 second video is 150 times as computationally expensive as processing a single image at the same resolution. We came up with a few tricks to get around this. The first was to simply reduce the quality of the video. For our five second training video the highest resolution we could consistently train on was 128 by 128. If we tried 256 by 256 pixel resolution our notebook would run out of RAM. Another modification we made was to train the video on pairs of two images at a time instead of training on the entire video. This allowed us to consume less memory and train faster by having much smaller inputs. The main issue with this approach is that it did a much worse job of maintaining consistency between frames. Each 2 frame video was independent of the others and there were many jumps. Our solution to this was to overlap the 2 frame videos and take the average of the frames to decide our final video. This yielded much smoother results while still granting us fast training time.

We have generated two examples that show our video style transfer in action. The first is the video of me dancing in my chair. The second, in honor of Argentina reaching the world cup finals, is Diego Maradona lifting the world cup in 198613. This video is grainy due to the old age and therefore has a lot more movement in the style transfer. The first video uses the Picasso painting we have been using throughout our paper. The second uses an image of confetti12. Finally, we also include the first image processed with 5000 iterations (as opposed to the 500 we were testing with). This took almost two hours to train but yielded much better style transfer results.



(a) Style image For training video (b) Style image For Maradona video

FIGURE 12: Style Images



(a) Video of me dancing (500 iterations) (b) Video of Diego Maradona Lifting the World Cup (c) Video of me dancing(5000 iterations)

FIGURE 13: Content Videos

If given more time to work on the video style transfer we would likely have attempted to implement a few further optimizations. The first optimization is to increase the length of the video we can process by loading only a few frames of video at a time. Secondly, we would have experimented with variable training "windows" and found which yielded the smoothest results. Finally, we would have experimented with more powerful hardware. RAM and processing power were a major limitation in my experimentation and we likely would have been able to experiment with a larger variety of videos in higher resolution.

5 Conclusion

In this project, we first study how the activation of each layer in a pretrained VGG-19 model represents the style and content of a image. We find that it is necessary to include deeper layers that can better represent the style or the content of a input image. By fixing either the style weight or the content weight and modify the other one, we learn the trade off between style and content. By adjusting the weights, we can make the final output image be more closer to the style or the content input.

Then we test the performance of the Neural Style Transfer model proposed by Gatys et.al by using different loss function, using features obtained from different pretrained model, using different optimization methods. We realize that in style transfer, for different input content image and style image, we need to fine tuning it for better results. For the input content and style that we use for test, we don't see a slight difference for different loss functions but I think L1 Loss somehow performs better. In the original paper the author uses L-BFGS as the optimizer while after our test we find that several other optimizers with a proper learning rate can also works well with this problem. Among them we choose Adam that performs the best. The choice of pretrained model can slightly affect the result. VGG models performs better than the ResNet models, which indicates that the skip connection in ResNet may cause some problem in style transfer. By adding two regularization term, we can improve our generated image by making it more photorealistic. We also manage to do style transfer on large scale image by combining the method of image stitching.

Finally, we implement video transfer for video. In addition to experimenting with the same factors as above (such as optimizers and loss functions, yielding similar results as found in image transfer), we also experiment with the addition of a temporal loss function and different methods of training and validation.

References

[1] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 2414–2423, 2016.

[2] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. CoRR, abs/1603.08155, 2016.

[3] Fujun Luan, Sylvain Paris, Eli Shechtman, and Kavita Bala. Deep photo style transfer. CoRR, abs/1703.07511, 2017.

[4] Haozhi Huang, Hao Wang, Wenhan Luo, Lin Ma, Wenhao Jiang, Xiaolong Zhu, Zhifeng Li, and Wei Liu. Real-time neural style transfer for videos. CVPR, 2017.