Optimizing the Style Loss of Art Style Transfer

Yu Guo, Cong Yuan

Abstract

Content loss and style loss are two major parts in art style transfer. The content loss is trivial; however, the style loss is more interesting and tricky. The original art style transfer paper[1] use gram matrix for style representation which is the correlation between each feature maps in 5 layers of VGG19 convolutional neural network. Why would gram matrix preserve the style of artistic images is not explained in original paper[1]. Meanwhile, computing gram matrix is computationally expensive. Also, it will give us too many correlation between feature maps in gram matrix for style loss, and many of them are not necessary if the detailed information is not need. To analyze and address the questions we raised, we first applied PCA to decrease the dimension of gram matrices used for style loss. After that, we improved our method and applied SVD to decrease the dimension of feature map matrix before computing the gram matrix. Finally, we were thinking of adding weights to each feature map.

1 Introduction

Art style transfer has always been our interest for many years. After some research on the original art style transfer research paper [1]. We found out that the style loss is still a mysterious part. The original paper did not explain why gram matrix could preserve the style of the image. Meanwhile, computing gram matrix contributes a lot of computation to the entire style transfer process. Under this circumstance, we plan to dig deeper on style loss matrix, try to optimize the style loss of art style transfer and try to decrease the time needed in style transfer process. Several research papers has been published to analyze why gram matrix could preserve the art style and how to transfer or re-define the style loss and gram matrix. For example, one research paper found out that transferring style is similar to matching the feature distributions because matching gram matrix between output image and style image is equivalent to minimizing the maximum mean discrepancy[2]. Another paper applied a different technique called texture synthesis to transfer artistic style between images[3]. In this paper, our approaches are inspired by them and are more straight forward. We applied PCA and SVD to the gram matrix in order to decrease the dimension of gram matrix and decrease the time of computation. Also, we added weights to each element of the gram matrix in order to improve the style transfer performance. The detailed explanation is located in Approach section.

2 Approach

According to the original art style transfer paper, the artistic style from style image can be transferred to the content image by minimizing a loss function involved with content representation and style representation[1]. Five layers of CNN VGG19(Figure 1) is used to compute both content representation and style representation(Figure 2). The content representation is simply computing the result of five layers CNN. The feature map matrices of each layer of CNN is stored in one matrix called F matrix. The style representation is calculating the matrix multiplication between F matrix and the transpose of it self in each layer of CNN, which is called gram matrix. The gram matrix of each layer is resized to a 2d matrix that each row contains a correlation between two feature maps. The content loss is the reduction between the content representation of output image and the content representation of content image. The style loss is the reduction between the style representation of output image and the style representation of style image. The total loss that we are minimizing is the addition of content loss and style loss. Gradient descent is used to

minimizing the total loss. The pixel values of output image is changed according to gradient descent in each iteration. Also, different weights can be applied to content loss and style loss base on the expected output image one want to produce.

Our work is built upon this art transfer technique, we applied three approaches to optimize the gram matrix described above.

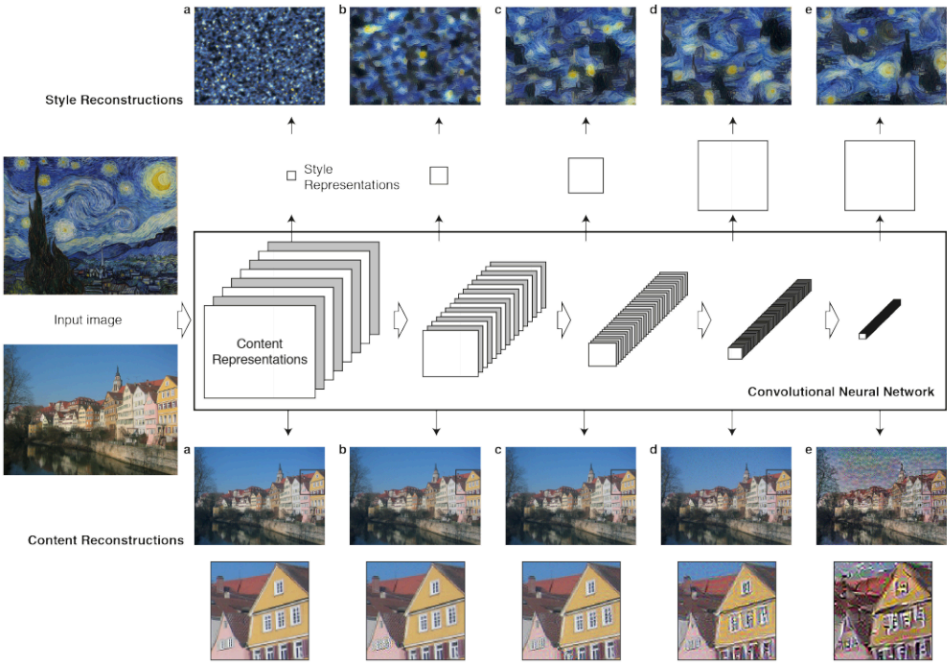

Figure 1

$$L_{total}(S, C, G) = \alpha L_{content}(C, G) + \beta L_{style}(S, G)$$

$$L_{style}(S, G) = \sum_{l=0}^{L} w_l * L_{GM}(S, G, l)$$

$$L_{content}(C, G, L) = \frac{1}{2} \sum_{ij} (a[L](C)_{ij} - a[L](G)_{ij})^2$$

$$L_{GM}(S, G, l) = \frac{1}{4N_l^2 M_l^2} \sum_{ij} (GM[l](S)_{ij} - GM[l](G)_{ij})^2$$

Figure 2

2.1 Principal component analysis (PCA)

The four dimension of the F matrix in each layer of CNN described above is firstly defined as: (1) first dimension a is the batch size; (2) second dimension b is the number of featured maps;(3) third dimension c and fourth dimension d is the number of rows and columns in featured map matrices. Then the code resize the 4D matrix F(a, b, c, d) feature map matrix to 2D matrix F(a*b, c*d). then compute the gram matrix by calculating the multiplication of F and transpose F.

Our intuition of applying PCA (Figure 3-5)to gram matrix is to decrease the time needed of minimizing the style loss by decreasing the number of rows in gram matrix. According to the functionality of PCA, it could reduce the number of rows in our gram matrix which is the number of correlations in our case, and preserve the most of information. Therefore, after applying PCA, the output image should be more smooth with less detail. The result of applying PCA in art style transfer is shown in Experiment session.

The hyper-parameter K in PCA is the threshold that control how much detail information we want to keep from original matrix. K must be integer in a range of (0 , a*b), when K close to 0, the result should be look like the content image itself, when K is close to c*d, the output should be like the original model output without PCA.

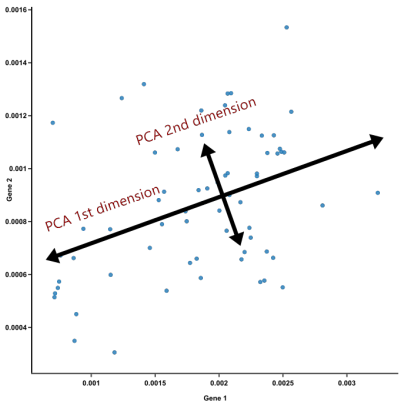

Figure 3

$$\frac{1}{n}XX^T = S = U\begin{bmatrix}\lambda_1 & & \\ & \ddots & \\ & & \lambda_d\end{bmatrix}U^T$$

Figure 4

$$\frac{1}{n}XX^T = \frac{1}{n}\begin{pmatrix}v_{1x} & \cdots & v_{nx}\\ v_{1y} & \ddots & v_{ny}\\ v_{1z} & & v_{nz}\end{pmatrix}\begin{pmatrix}v_{1x} & v_{1y} & v_{1z}\\ & \ddots & \\ v_{nx} & & v_{nz}\end{pmatrix} = \begin{pmatrix}cov(x,x) & cov(x,y) & cov(x,z)\\ cov(y,x) & cov(y,y) & cov(y,z)\\ cov(z,x) & cov(z,y) & cov(z,z)\end{pmatrix}$$

Figure 5

2.2 Singular value decomposition(SVD)

In linear algebra, the singular value decomposition (SVD) can decompose any matrices to a matrix multiplication of three matrices. The diagonal entries of the diagonal matrix in SVD are known as singular values of original matrix. [5] Therefore, instead of applying PCA to gram matrix directly, we could also apply SVD to decrease the dimension of F matrix described above before we computing the gram matrix. By doing so, the time used in computing gram matrix is decreased as well as the time needed to minimize the style loss.

Like PCA, we have a hyper parameter S for SVD that is the threshold for singular values in diagonal matrix (Σ) of SVD. If the singular value in Σ is smaller than the threshold, we turn

it to 0 because it contains little information from original matrix. By doing so, the dimension of the original matrix F is decreased. Also, the gram matrix can be computed by a shorten form of SVD multiplication.(Figure 6-7)

$$X = U\Sigma V^T$$
$$XX^T = U\Sigma V^T (U\Sigma V^T)^T =$$
$$= U\Sigma V^T V \Sigma^T U^T = U\tilde{\Sigma}^2 U^T$$

Figure 6

$$XX^T = S = U \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_d \end{bmatrix} U^T$$

Figure 7

2.3 Weighted gram matrix

If we use gram matrix directly as style representation, we are treating every correlation between two feature maps with the same weight. However, intuitively it is not making sense because art style is not uniformly distributed in one art image. Some rows of gram matrix should have more weights when it contains more style information and vice versa. Therefore, we could add a weight vector when we compute the gram matrix, initialize the values in weight vector and train it.

$$\mathrm{G(X)} = \mathrm{X}^* X^T$$

$$G(X) = \begin{bmatrix} x_{11} & \cdots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nm} \end{bmatrix} \begin{bmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{bmatrix}$$

Figure 7

$$G(X) = W * X * X^T$$

$$G(X) = \begin{bmatrix} w_{11} & \cdots & w_{1n} \end{bmatrix} \begin{bmatrix} x_{11} & \cdots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nm} \end{bmatrix} \begin{bmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{bmatrix}$$

Figure 8

3 Experiment

we run our model on Google Colab, the Gpu spec is on the table(Figure 9). From the original model[1], We applied our method on top of the tutorial from pytorch[6]. We test the result of our method using our own dataset. We used a picture of SFU campus and some famous artwork images available online.

The original gram function is defined as (Figure 10).

Instance Type	GPUs	GPU Memory (GB)	vCPUs	System Memory (GB)	Instance Speed	On-Demand Hourly Rate	1-yr Reserved Effective Hourly Rate
Lambda gpu.4x	4 GPUs	11 GB / GPU	8 vCPU Cores @ 3.5 GHz	32 GB DDR4	Up to 10 Gbps	$1.50	$1.25

Figure 9

```
def gram_matrix(input):
    a, b, c, d = input.size()
    features = input.view(a * b, c * d)
    G = torch.mm(features, features.t())
    return G.div(a * b * c * d)
```

Figure 10

3.1 Principal component analysis (PCA) Experiment

We applied PCA (Figure 11)to decrease the number of rows in gram matrix and test it by transferring several famous art style to SFU campus image.

PCA is working as we expect except that the runtime of entire process is actually longer than before. After analyzing it, we found that it because of two reasons.

1. The Pytorch does not support sparse matrix on gpu computation, so we have to extract the dataset and put it back to CPU. Then computing PCA of the dataset using Numpy; otherwise, it will produce memory error. This process takes much time.

2. There exist a computation trade off in our approach. Applying PCA need extra computation but it can reduce the computation needed in optimizing part. If the extra computation needed for PCA is smaller than the computation saved in optimizing, the time needed for completing the entire process of art style transfer would be short than original, vice versa(Figure 12).

```python
def gram_matrix(input):
    a, b, c, d = input.size()
    features = input.view(a * b, c * d)
    G = torch.mm(features, features.t())
    G = G.div(a * b * c * d)
    G_mean = torch.mean(G,0)
    G = G - G_mean.expand_as(G)
    U,S,V = numpy.linalg.svd(torch.t(G).cpu().detach().numpy())
    k = int((G.size()[0])/2)
    U = torch.Tensor(U).to(device)
    result = torch.mm(G,U[:,:k])
    return result
```

Figure 11

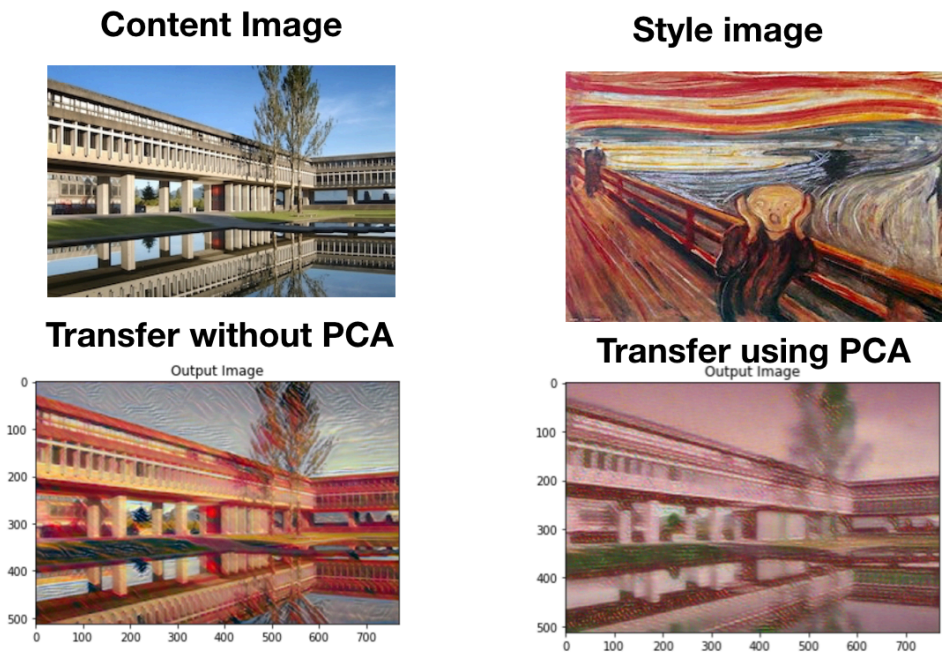

Figure 12

3.2 Singular value decomposition (SVD)Experiment

We tried to apply SVD to F matrix both on Pytorch and Tensorflow(Figure 13), but we got a memory error on both two methods even if we use sparse matrix. The error message is shown below:

”RuntimeError: CUDA out of memory. Tried to allocate 577.50 GiB (GPU 0; 15.90 GiB total capacity; 456.42 MiB already allocated; 14.73 GiB free; 21.58 MiB cached)"

We guess it happens because the number of columns in F matrix is too big. So we need a lot of memory to compute the SVD. Therefore, we do not have a clear result if applying SVD would reduce the number of computation in art style transfer.

```
def gram_matrix(input):
    a, b, c, d = input.size()
    features = input.view(a * b, c * d)
    G = torch.mm(features, features.t())
    U,S,V = numpy.linalg.svd(G.cpu().detach().numpy())
    S_mean = numpy.mean(S)
    S[S < S_mean] = 0
    G_cpu = numpy.matmul(U,S)
    G_cpu = numpy.matmul(G_cpu,V)
    G = torch.Tensor(G_cpu).to(device)
    print(G)
```

Figure 13

3.3 Weighted gram matrix Experiment

Because of the time and knowledge shortage, we do not have an entire method to apply this approach in the original model. We could use the same technique in GAN to train one more variable, but it is too difficult and time consuming. We expressed our idea in this paper as a reference. Hopefully, we could achieve it in the future.

4 Conclusion

Despite the great success of art style transfer, the runtime of style transfer was far from perfect. we come up with 3 ideas to improve the performance of the nature style transfer but only one of them is working for now. but it is useful in some certain circumstance like app on mobile devices that it is very importance to reduce run time but the result can be less detailed in the meantime. The other two ideas is theoretically working but need more effort and time.

5 Citation

[1] Gatys, Leon, et al. “A Neural Algorithm of Artistic Style.” Journal of Vision, vol. 16, no. 12, Jan. 2016, p. 326., doi:10.1167/16.12.326.

[2]Li, Yanghao, et al. “Demystifying Neural Style Transfer.” Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, 2017, doi:10.24963/ijcai.2017/310.

[3]E. Wang and N. Tan, “ Artistic Style Transfer,” 2016.

[4]“Principal Component Analysis.” Wikipedia, Wikimedia Foundation, 28 Nov. 2019, https://en.wikipedia.org/wiki/Principal_component_analysis.

[5]“Singular Value Decomposition.” Wikipedia, Wikimedia Foundation, 24 Nov. 2019, https://en.wikipedia.org/wiki/Singular_value_decomposition.

[6]“Neural Transfer Using PyTorch¶.” Neural Transfer Using PyTorch - PyTorch Tutorials 1.3.1 Documentation, https://pytorch.org/tutorials/advanced/neural_style_tutorial.html.