Style Transfer: Two Methods

Haodong Wu, Xiaoya Wang, Yinuo Zhang
Courant Institute of Mathematical Sciences, New York University
{hw1635, xw1963, yz3948}@nyu.edu

Abstract

Style transfer is to take an input image as the target style, an input content image to convert, then generate the combination of the two to produce artistically styled results. We have studied two methods of doing so. One uses a pre-trained model to generate the output and takes several minutes, commonly known as neural style transfer; the other one trains a model on a given style image and generates the output within seconds, known as real-time style transfer.

*For neural style transfer, we used pre-trained VGG-16 and VGG-19. We experimented with different configurations and found that VGG-19 with Adam optimizer yields the best result. For VGG-16 with 1000 epochs using LBFGS, generating output for our given example here of size 681*968 takes 2min with a Tesla P100, and VGG-19 with Adam optimizer with 4000 epochs takes around half an hour. Even though this method can generate visually appealing results, speed becomes a major concern.*

*For real-time style transfer, we first used pre-trained VGG-16 to extract features of the style features, then built a transformation model with in-network upsampling and downsampling. We experimented with different datasets and found out that for this task, larger datasets generate a better result. Once the model is trained (time taken varies on the size of the dataset), generating a result for a content picture of size 681*968 takes 0.52s on a Tesla P100.*

With the above exploration, we were able to achieve our goals when proposing the project and generated nicely-looking artistic pictures of our choice. Code written for this project can be downloaded through our GitHub repository.

1. Introduction

Style transfer has been a heated topic in recent years. The general idea is applying one artwork image to another normal image and generate a combined image of the two. The technique is widely used in different applications such as Snapchat filters, Instagram Story and could easily produce millions of masterpiece artworks. We used pre-trained VGG-16 network and VGG-19 network based on the method proposed by [2], extracted specific features and implemented loss functions for both style and content.

Additionally, we compared multiple optimizers, including SGD, Adam[4], Adamdelta, Adagrad, LBFGS for VGG-19 and different learning rate for some optimizers to see the transfer result. Based on the result from VGG-19, we continued to compare Adam and LBFGS on VGG-16.

However, training these images need several minutes, thus unscalable and expensive. We went on to study real-time style transfer[3] and gained corresponding results. The real-time style transfer can reduce the inference time within seconds, which enables the use of this technique in video, audio and other medium.

2. Work Related

The pioneer paper[2] by Gatys et al. proposes the first neural transfer algorithm, where they introduces the usage of several convolutional neural network layers to extract features. The original paper implemented such transfer system with VGG-19 network and chooses conv1-1, conv2-1, conv3-1, conv4-1, and conv5-1 layers to extract content features, as shown in Figure 1. Each layer learns the image where lower layers mostly represent detailed pixel information and higher layers capture the complex representation. This allows the system to distinguish between the style and the content of a picture. The objective is to reduce the stylistic difference between the output image and the style image while keeping the content of the input image.

Another paper proposes an approach to implement real-time style transfer algorithm[3]. Instead of looking at the pixel-level difference, Johnson et al. focus on the high-level discrepancy between the images. The model contains two main blocks: an image transformation network, which takes the content image as the input and generates the stylized image; the loss network, which extracts the style feature and content features then uses them to optimize the model by minimizing the perceptual losses. This approach not only generates the transformed image faster but gives comparable result to the one proposed by Gatys et al[2].

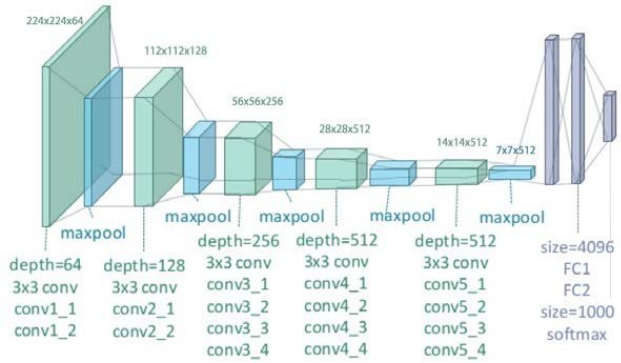


Figure 1: VGG-19 Network Visualizaiton, Source: ResearchGate

3. Methods

3.1. Neural Style Transfer

3.1.1 Image Preprocessing

First, we preprocess the content and style images by converting a PIL image in range [0,255] of shape (H,W,C), meaning (height, weight, channel) to a FloatTensor in range [0,1] of shape (C,H,W). We use the transform functions in pytorch to implement such transformation. To visualize the output image, we also implemented a reverse transform function that converts Tensor back to an image that can be printed out.

3.1.2 Feature Extraction

There are two models tested in project, namely VGG-19 and VGG-16 [6]. We only care about the intermediate layers and don't use fully connected layers as suggested in [2]. The reason is that intermediate layers are good at extracting features even though there is background noise.

3.1.3 Loss Function

The loss function contains two parts. One is content loss and according to [2], it's defined as

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

where the p and x are the input image vectors and l is the certain layer. This is calculating the mean square loss of the images.

Before defining the style content loss formula. We need to define a gram matrix, which is the core criteria to calculate style loss. It is the inner product of the vectorized feature maps i and j in layer l[2].

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

The loss of a certain layer is

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

3.1.4 Training

In VGG-19, we used Adam optimizer and customized the weight of different layers to calculate content loss, which gives us more freedom to adjust the weight of different layers. Since Adam appears to converge relatively slowly, we setup 4000 iterations. We also adjusted the distribution of the style weight ratio, changed the optimizers, and learning rate. (See Experiment)

In VGG-16, we used the LBFGS optimizer instead as the original paper suggests. LBFGS can generate visually pleasing result with fewer iterations and we set it to be 1000. We experimented with style inputs of different dimensions as well. We also adjusted the content weight and style weight to find the combination that yields the best output. Same as the experiment for VGG-19, we tested out different optimizers (see experiment).

3.2. Real-Time Style Transfer

Compared to Gatys approach which optimizes the loss function with many forward and backward iterations, real-time algorithm[3] feeds the input image forward through the transformer network with only one iteration. The system comprises of two parts - **Image Transformation Network**, which is a deep residual convolutional neural network, and **Loss Network**, which is built on top of VGG-19[6].

3.2.1 Image Transformation Network

Image transformation network starts with three convolutional layers with downsampling, allowing the feature filters to have much more effective information of the input image. Since the output should be the same size as the input, 2-stride convolutions are followed by upsampling layers. Between downsampling and upsampling lies five residual blocks. They helps the model to learn the difference between the content image and the stylized image and make it easier to train a very deep neural network. The transformer net takes the content image as the input and gives back the transformed image which combines the input image with the artistic feature in the style image. The parameters of this model is optimized by minimizing the loss function computed by loss network.

3.2.2 Loss Network

Similar to what we did in neural style transfer algorithm, the loss network leveraged the pre-trained image classification

Figure 2: Training with Batch Normalization

model, VGG, to optimize the loss function. Specifically, we use the $relu1_2$, $relu2_2$, $relu3_3$, $relu4_3$ to extract the style information and $relu2_2$ to get the content representation.

3.2.3 Instance Normalization

In the paper, convolutional layers are followed by batch normalization to speed up the training time and tackle the vanishing gradient. However, in our implementation, we realized that batch normalization will render anomalous pattern(Figure 2) in the final result. Instead, we use the Instance normalization, inspired by Ulyanov et al. [7]. This enables us to generate coherent images because instance normalization works on per-image basis and thus makes the each image distribution towards Gaussian.

3.2.4 Loss Function

We learn the weights of the transform net by extracting the style feature and content feature using VGG-16, calculating the style distance and the content distance, and then minimizing both distances at the same time.

Let $\phi_j(x)$ be the activations of the j-th layer of the loss network ϕ when processing the image x and $\phi_j(x)$ be the feature map of shape $Cj \times Hj \times Wj$. The feature reconstruction loss ℓ_{feat}[3] can be defined as the Euclidean distance between feature representations of the content image and that of the transformed image:

$$\ell_{feat}^{\phi,j}(\hat{y}, y) = \frac{1}{C_j H_j W_j} \|\phi_j(\hat{y}) - \phi_j(y)\|_2^2$$

and the style loss[3] can be computed using the Gram matrix[3] defined in the neural style transfer model. Basically, Gram matrix helps to find out the distribution of the feature maps in a single layer, which enforces the match of the image distribution between the transformed image and the style image.

$$G_j^{\phi}(x)_{c,c'} = \frac{1}{C_j H_j W_j} \sum_{h=1}^{Hj} \sum_{w=1}^{Wj} \phi_j(x)_{h,w,c} \phi_j(x)_{h,w,c'}$$

$$\ell_{style}^{\phi,j} = \|G_j^{\phi}(\hat{y}) - G_j^{\phi}(y)\|_F^2$$

3.2.5 Training and Testing

We used the Adam optimizer and trained images on MSCOCO dataset[5]. During the prediction, real-time system is able to generate a stylized image of size 224×224 in only 7.3 ms, overwhelming faster than the neural transfer algorithm. We also experimented with different sets of hyper parameters and explored how the number of the iteration, the size of the dataset, style weight and content weight affect the model performance.

4. Experiment

4.1. Neural Style Transfer

4.1.1 Baseline

In VGG-16, we used Eiffel Tower as content image and Starry Night as style image. (Figure 3) We trained the model with 1000 epochs using the LBFGS optimizer. We will use this as the baseline as it is the most commonly seen. We also generated other images of other styles with the same configuration, which can be found in the attachments.

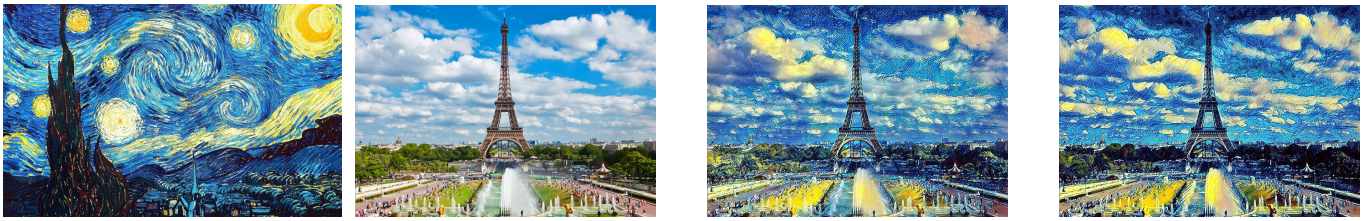
In VGG-19, we used the same combination of style and content images for direct comparison. Figure 1 shows the result after running VGG-19 with Adam optimizer and 4000 epochs. This setup generated a more visually pleasing output in our case as shown.

4.1.2 Tuning Hyperparameters

We want to see how important each layer is on the training result, so we setup a different ratio and trained the result. It turns out that the original ratio has a darker color that resembles the original style image. For example, the sky color in Eiffel Tower image is dark blue while the other ratio set render the sky with light blue (Figure 5). The reason is that we lower the weight of the first layer and increased the weight of the deep layers, the system captures less style feature of the image.

Similarly, we also play around with the weight of content versus style. At first, the content is 1e2 and style is 1e8, which generates a ratio of 1:1000000. Next, we reset the content to be 10 and style to be 1e9, thus increasing the ratio and emphasizing more about style. The output of the latter visually looks more appealing because it highlights more on style, which makes sense.

Furthermore, we noticed that for Adam optimizer, it needs higher learning rate to achieve a better result because the image is too large, leading to a larger total loss. 0.1

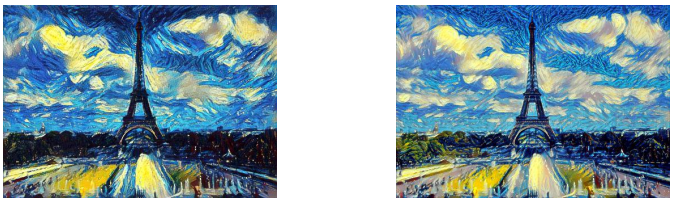


(a) Style (b) Content (c) VGG-16, LBFGS (d) VGG-19, Adam

Figure 3: Starry Night + Eiffel Tower for Neural Style Transfer

(a) Style (b) Content (c) VGG-16, LBFGS (d) VGG-19, Adam

Figure 4: Waves + Washington Square Park for Neural Style Transfer



(a) Original ratio set (b) A different ratio set

Figure 5: Different ratio parameter comparison

(a) Dancing girl (b) Picasso's painting

Figure 6: Images used to test optimizers

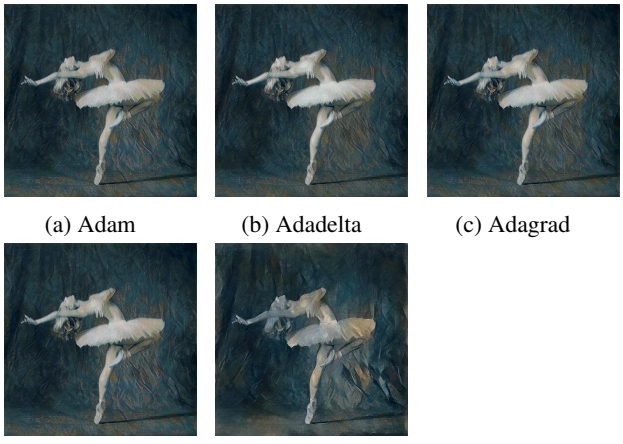
learning rate for image size around 1000*1000 is better while a 0.007 learning rate for image size 500*500 is better.

4.1.3 Optimizer Experiments

Although Gatys et al. [2] suggest to use the Limited-memory BFGS (L-BFGS) for the model, we attempted the other four commonly used optimizers in our VGG-19 model:

- SGD (Gradient Descent)
- Adam[4]
- Adamgrad[1]
- Adadelta[8]

To save running time and memory, we chose two relatively small size of content and style images, shown on Figure 6. We run 4000 epochs for all optimizers, with a learning rate of 0.1.



(a) Adam (b) Adadelta (c) Adagrad

(d) SGD (e) L-BFGS

Figure 7: Transformed images with the five optimizers under VGG-19

Figure 7 shows the final output images using different optimizers. In terms of visual effect, L-BFGS obviously

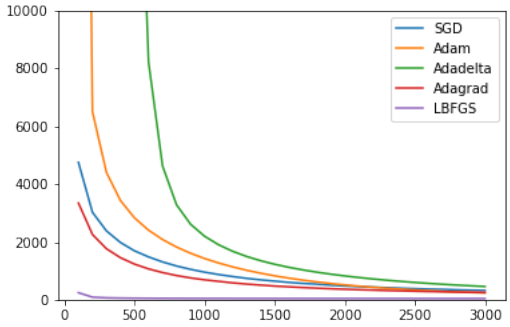


Figure 8: Optimizer Convergence Graph

stands out among other optimizers. The texture is captured only by L-BFGS. The other four output images barely have any difference. However, speaking of how fast the loss decreases, Adagrad has the best speed if not counting on LBFGS (Figure 8). What needs more notice is that LBFGS drops from over 50,000 loss to less than 100 after the first 100 epochs. Therefore, the LBFGS has a better result if we are given fewer epochs because it converges much faster than other optimizers. Adam and Adamgrad also gives decent results even though it needs more epochs to converge.

4.2. Real-Time Style Transfer

For real-time style transfer, we trained models on two styles: Mosaic and Udine, since these two have very distinctive features. There are many decisions to be made during training. We tested different configurations (style weights, normalization functions, datasets, epochs, etc).

4.2.1 Dataset

For the choice of dataset, we tested on COCO 2014 (both train and val) and COCO 2017 (val) datasets[5]. In our case, we only care about the size of the dataset, so we picked them as they vary greatly in sizes. The COCO 2014 training set has 80k images, validation set has 40k images; the COCO 2017 validation dataset has 5k images. We used the hyperparameters from [3] for this experiment. We found that with the smaller datasets, the brightness of the output is significantly impacted (Figure 10 and Figure 11), so we went on to train the models with the larger dataset, which is the COCO 2014 training set.

4.2.2 Tuning Hyperparameter

We experimented with different epochs first, and we found that the model starts to overfit when epoch is greater than 2, so we trained the model with 2 epochs. We then tested out different style weights and content weights, and found out

Table 1: Prediction time on image of size (2560,1440)

Model	Inference Time
Neural Style Transfer	**11min 32s**
Real-Time Style Transfer	0.52s

that there is a very fine line at when content weight is 1 and style weight is 1e5. When the style weight is smaller (1e4, 1e3), the network almost does not pick up any information from the style image; When the content weight is higher, a similar situation happens. We had to stick to this configuration. However, we notice that the model seems to pick up edges with high color contrast more, thus resulting in the tile-like patterns shown in Figure 9 and 10.

4.2.3 Layer Selection

In the loss net, we can choose different layers from VGG-16 to represent the content information and the style information. Ideally the higher layer would forget the image details, like edges and dots, and stores the semantic information. We tried two approaches - one using $relu2-2$ to extract the features and the other using $relu3-3$. The results, shown in Figure 12 are comparable to each other, with $relu2-2$ capturing more tiny details and $relu3-3$ rendering coherent image with less noise. For the image of fluffy cat, it's probably better to use a relatively higher layer since too much details combined with the style feature makes it look messy.

4.2.4 Training time and Prediction time

The training time taken on COCO 2014 Validation set (40k images) with Tesla P100 is 41 minutes and the prediction time for a image of size 2560×1440 is only 0.52 second, which is a great improvement on the speed compared to the first method.

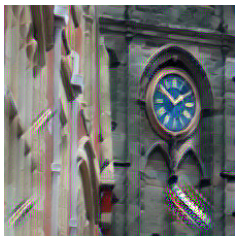
5. Discussion

5.1. VGG-19 vs VGG-16

Because the implementations are different for calculating total loss, there are some variances when comparing the loss results. Therefore, we try to compare the results based on visual appearance. When training the same content and style images, the VGG-19 using Adam optimizer generates visually better output, as shown in Figure 3 and 4. One of the reason might be that these two models have different loss calculation where in VGG-16 the style loss is calculated simply by MSE function in pytorch but in VGG-19, the loss is customized to include style layer ratio set, which enables us to tune the weight of each style layer. Even with the same amount of iterations, VGG-19 with Adam still

(a) Style

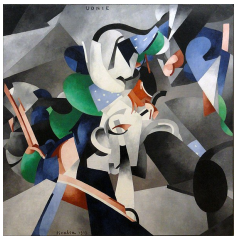
(b) Result 1



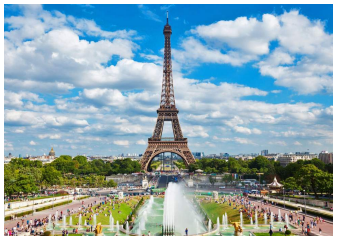
(c) Result 2

(d) Result 3

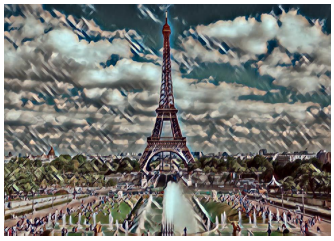
Figure 9: Intermediate Results During Training



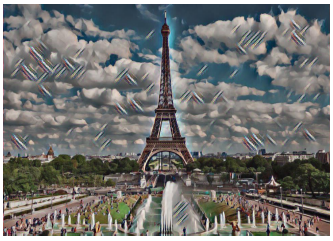
(a) Style



(b) Content



(c) Result, style weight = 1e5 on the dataset (5k images)



(d) Result, style weight = 1e5 on the dataset (80k images)

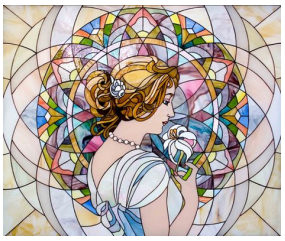
Figure 10: Udine + Eiffel Tower for Real-Time Style Transfer

(a) Style

(b) Content

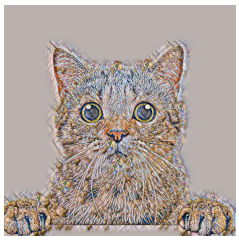
(c) Result

Figure 11: Udine + Washington Square Park for Real-Time Style Transfer

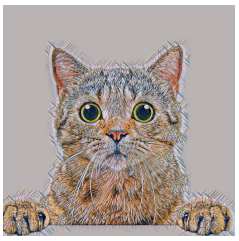


(a) Style

(b) Content



(c) Result using relu2-2 to extract content features



(d) Result using relu3-3 to extract content features

Figure 12: Mosaic + Cat with relu2-2 and relu 3-3 as the content representation

generates better results that resemble the style image more (color, brightness, etc) comparing to VGG-19 with LBFGS.

With further investigation, we believe that in our case, for smaller images (Figure 7), LBFGS generates better results and for larger images (Figure 3 and Figure 4 Adam gener-

Figure 13: Training with Starry Night

ates better results (see the attached pictures).

5.2. Real-time neural transfer

Through various experiments on different configurations, styles and architectures, we have found that the models trained on a reasonably large dataset can generate satisfactory results (40k 80k images). We found that this model is particularly good at learning styles that contain edges with sharp contrast (mosaic.jpg and udine.jpg). When we attempt to train it with Starry Night, we get results that look like there's too much noise on the background (see Figure 13) and it does not yield visually pleasing output.

Some comparison can be made against the neural style transfer is that this model can handle large pictures like a breeze. The Washington Square Park picture is a 2K picture and the outcome is generated with only 0.52 second on a Tesla P100, where it would be impossible to directly feed this picture into VGG-19 and train since the GPU will run out of memory. The speed and scalability advantage enables this model to transfer videos in real-time. However, this would be beyond the scope of this project.

References

[1] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization, 2011. 4

[2] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style, 2015. 1, 2, 4

[3] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution, 2016. 1, 2, 3, 5

[4] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015. 1, 4

[5] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015. 3, 5

[6] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015. 2

[7] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis, 2017. 3

[8] Matthew D. Zeiler. Adadelta: An adaptive learning rate method, 2012. 4