CS231N Project Final Report - Fast Mixed Style Transfer

Xueyuan Mei
Stanford University
Computer Science
xmei9@stanford.edu

Fabian Chan
Stanford University
Computer Science
fabianc@stanford.edu

Tianchang He
Stanford University
Electrical Engineering
th7@stanford.edu

Abstract

This project explores methods for artistic style transfer based on convolutional neural networks. As a technique that combines both artistic aspects and recognition (content) aspects of images, style transfer has always been an interesting topic for researchers in the field of computer vision. With the rapid growth of Deep Convolutional Neural Networks, style transfer can now be accomplished in a speedy manner. Based on papers that relate to fast-style style transfer as well as mixed-style transfer, we have developed our own implementation of style transfer that manages to transfer images with mixed styles or even unseen styles in real-time. We experimented with 3 different implementations involving fast mixed and arbitrary style transfer techniques and compared their performances. Moreover, because not everybody can readily access massive compute clusters, we also explored ways to rescale these techniques and apply them to smaller-scale compute infrastructure.

1. Introduction

1.1. Fast Style Transfer

Among the applications of convolutional neural networks (CNN) and visual recognition, style transfer has been a very heated topic. Style transfer is the technique of separating and recombining the content and the style of an arbitrary image. The topic is particularly interesting because it creates artificial intelligence that inter-plays the content and the style of an image to produce artistic results of high quality. The problem used to be difficult because it was hard to extract texture information using conventional computer vision techniques. With the advent of CNNs, we are able to tackle the problem in a more sophisticated manner. Our project aims to explore the CNN structures of Style Transfer algorithms proposed by the Gatys paper [1] and also apply the algorithm with a Fast-Neural-Style proposed by Johnson [2] so that the transfer for an arbitrary image to a certain style can be done in real-time.

1.2. Mixed Style Transfer

In the perspective of user experience, it is not only preferred to have style transfer accomplished in a quick manner, but it is also important that the style transfer is able to deal with multiple styles. To achieve this purpose, we decide to add additional feature layers so that our style transfer model can tackle situations in which multiple styles are involved. We first take the approach that utilizes a one-hot conditional vector to specify which styles are incorporated. Later on, we also experimented with approaches that handle unseen style images.

2. Related Work

Gatys [1] first introduced in 2015 a deep neural network approach that extracts neural representations to separate and recombine the content and style of arbitrary images. CNNs are some of the most powerful procedures for image processing tasks, and have recently reached human-level performance in classification tasks. The neural layers of a CNN can be understood as a set of image filters that extracts higher level features from the pixels of an input image. CNNs develop a representation of the image that makes content information increasingly explicit along the processing hierarchy as they train, such that the input image is transformed into representations that increasingly care about the actual content of the image compared to the individual values of its pixels.

Although [1] showed that the style and content of an image can be disentangled and applied independently, the method is computationally expensive. Johnson′s work [2] was able to speed up style transfer by training a feed-forward network to replace the optimization-based method of Gatys, and ended up being many times faster, allowing the transformation of video input in real-time. Furthermore, Keiji [5], Huang et al [6] and Ghiasi et al [7] proposed methods that augment Johnson′s work to take a style choice or style image as inputs to the feed-forward network. Keiji [5] proposed that the network can take an additional conditional vector input indicating the style and the styles can be

mixed at test time. Huang et al [6] proposed the network can learn a set of adaptive instance normalization parameters representing a style. Ghiasi et al [7] expands on this idea and used a Inception v3 network to extract the style as the normalization parameters thus achieving arbitrary style transfer.

3. Problem Statement

For this project we explore different ways to build neural network architectures that can generate multiple styles in a fast manner. It is worth clarifying that when we refer to generating multiple styles, we mean that there should be exactly one instance of the network that is responsible for generating any style from a set, as opposed to having multiple instances where each individual instance corresponds to one particular style.

The data we used consisted of 13 GB's worth of images from Microsofts COCO 2014 dataset and artworks of various artistic styles collected from the web, many of which come from WikiArt.org. We expect a good algorithm will likely, for example, incorporate many aspects of a painter's unique style, from the movement of brushstrokes to the use of light and darkness in color choice. Evaluation of our implementation will take both speed and quality into account. However deciding whether or not a particular style transfer was done well is mostly a subjective process and as such we can only rely on human evaluation.

4. Approach

4.1. Deep Representation of Image

Gatys et al [1] proposed that we can construct deep representations of an image using a neural network and separate the content and style of the image in such a way that we can compare the difference of content and style with another image independently. Their approach was to use the response layers of a pretrained VGG-16 network [3].

The image content representation is chosen to be the values at certain response layers. Suppose a layer with N_l filters can produce N_l feature maps, each of size M_l and the response is $F_l \in R^{N_l \times M_l}$. If this layer is chosen to represent the image content, then the content difference between two images I_1 and I_2 at this layer is

$$L^l_{content}(I_1, I_2) = \sum_{i,j}(F_{1,ij} - F_{2,ij})^2 \tag{1}$$

Similarly, the style representation is calculated using the response of certain layers in the network. Since the response layers of images of different sizes cannot be compared directly, [1] proposed to form the Gram matrix of representations with

$$G^l = F^l F^{lT} \tag{2}$$

Thus we have $G^l \in R^{N_l \times N_l}$. The style difference between two images at this layer is

$$L^l_{style}(I_1, I_2) = \sum_{i,j}(G_{1,ij} - G_{2,ij})^2 \tag{3}$$

4.2. Style Transfer

The style transfer problem is then transformed to the problem of constructing an image I that has similar content to the content target I_c and similar style to the style target I_s. Once we determined which layers are used to represent content and style i.e. R_c and R_s, we can construct the total loss as

$$L_{total}(I, I_c, I_s) = \alpha L_{content}(I, I_c) + \beta L_{style}(I, I_s), \tag{4}$$

where

$$\begin{aligned} L_{content}(I, I_c) &= \sum_{l \in R_c} w_l L^l_{content}(I, I_c) \\ L_{style}(I, I_s) &= \sum_{l \in R_s} w_l L^l_{style}(I, I_s) \end{aligned} \tag{5}$$

and α, β are hyperparameters that give different weights to the two losses. Usually the weights associated with each layer w_l is taken to be uniform. The image I can then be constructed by backpropagating the loss to the image with the weights of the network fixed.

4.3. Fast Style Transfer

The approach described in previous sections proved to be computationally expensive since for each image the update process has to be carried out for many iterations through deep CNN such as the VGG network. It was proposed by Johnson et al in [2] that it would be preferred to use a feed-forward network that transforms the original (content) image to the stylized image. The process is illustrated in Figure 1.

In this approach, the w_l of each layer is taken to be inversely proportional to the size of the response layer. The image transformation network has structure of convolution - residual blocks - convolution layers. Also, the image transformation network was trained and the loss was constructed using the layers shown in the figure. After training, the weights of the transformation network can then be extracted and are used to convert a content image using a single forward pass.

4.4. Fast Mixed Style Transfer

The previous approach is limited mainly in that 1) It can only train one network per one style, and 2) It cannot mix styles together. This is tackled by [5] by introducing an alternative architecture of the feed-forward network that can work with multiple styles. The architecture is illustrated in

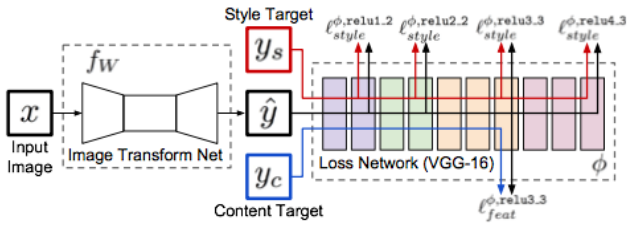


Figure 1: Illustration of training a feed-forward network to transform the image [2].

Figure 2. The additional input is a vector that has length equal to the number of styles and is one-hot in the chosen style. The vector is duplicated and concatenated into the network. At training it will effectively select a subset of the subsequent network to train for the chosen style. At test time the conditional vector can be used to choose or mix multiple styles to apply to the content image and can achieve fast mixed style transfer.

In our project, we implemented the fast mixed style transfer model with Tensorflow based on [4]. A total of 7 styles were chosen and the network was trained with different content weights.

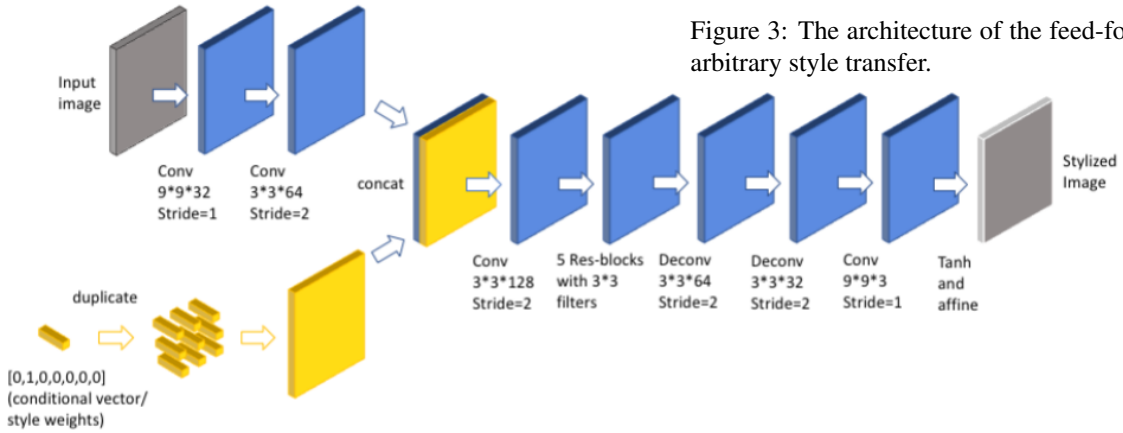


Figure 2: The architecture of the feed-forward network for fast mixed style transfer.

4.5. Arbitrary Style Transfer

To generalize the style transfer problem, the feed-forward network can take the style image as input as well, as illustrated in Figure 3. The style prediction network uses 4 convolutional layers, 3 inception modules and a convolutional layer followed by a global average pooling layer to extract the style as two 256-dimensional vectors feeding into the image transformation network as the scale and offset to be applied to the response after an instance normalization layer.

As mentioned in Section 2, [7] demonstrated this possibility. For our project, we attempted to replace the pre-trained style extraction network with a smaller and trainable network described above because we have much smaller compute infrastructure available. We were able to train the network on 5 styles only. Our implementation was limited in terms of GPU memory since Tensorflow only supports static computational graphs.

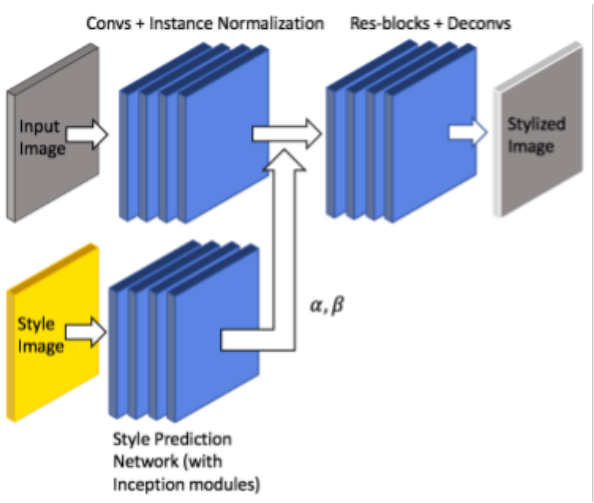


Figure 3: The architecture of the feed-forward network for arbitrary style transfer.

5. Experiments

5.1. Fast Style Transfer

We first trained the network with artwork shown in Figure 4 as the style target. Qi Baishi was an influential Chinese painter known for his whimsical, minimalistic, and often playful style in his watercolor works and we trained the network with one of his paintings. About half of the images

in the COCO dataset were used as content images and the process took about 7 hours on a Tesla K80, which is also used in all subsequent training. Then we used the network to generate the result shown in Figure 5. The stylized image seems to capture the texture and artistic style well. One can easily observe the modest palette choice and the broad brushstrokes evident among the clouds.

We also tried feeding a transformed image back into the feed-forward network and generated the images shown in Figure 7. If the network can truly stylize the image, the result after feeding through the network twice should look roughly the same as that after feeding it once. It is interesting to see that the network retained much of the characteristics of the images that are already stylized.

After gaining confidence in our trained fast style transfer system, we experimented our system with more content images. In Figure 6, we can see that our style transfer model works well when applied to different kinds of content images, such as those that depict animals, buildings, and nature. For all stylized images in the collection, it appears clear that the overall style matches that of the Chinese traditional ink paintings.

Note that at test time it took approximately 0.7s with Tensorflow to generate an output image. Most of the elapse time is spent in setting up the computational graph from scratch, and initializing the network variables. However it has been demonstrated in the industry that the feed-forward network can run in real-time on mobile devices by hard-coding the network in software.

Figure 4: The style target: Shrimps painted by Qi Baishi.

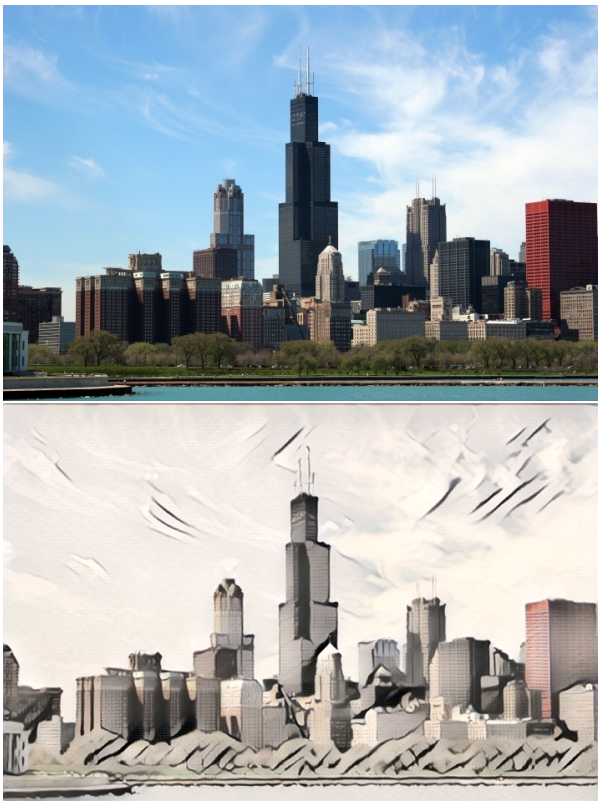


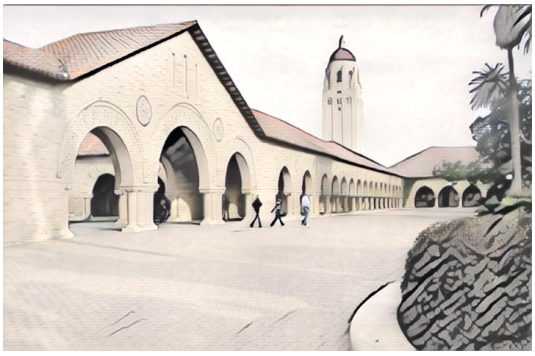
Figure 5: The original image and the stylized image.

Figure 6: The original images (left) and the stylized images (right).

5.2. Experiments with Different Content Weights

To explore how the weights of content loss effect the results of style transfer, we tried to train with different content weight values. Figure 8 shows the results of different values of content weights α. From Figure 8c, we can see that when the content weight is relatively low ($\alpha = 1e0$), there is barely any detail from the content image, and only the contour of the mountain can be observed. As the content weight increases, more and more details of the content image appear. Looking at an extreme case where the weight $\alpha = 100$ at figure 8e, we can see that the stylized image now has little style effects; instead, it appears to be very

(a) The original image.



(b) The output from feeding 7a into the network

(c) The output from feeding 7b into the network.

Figure 7: Experiments with the feed-forward network.

(a) α = 1e0 (b) α = 1e1

(c) α = 1.5e1 (d) α = 2e1

(e) α = 1e2 (f) original image

Figure 8: Images with different content weights

similar to the original un-stylized image. The texture of this image is almost the same as the original, with only tiny differences in the overall tone of color. After several rounds of weight tuning, we found that content weight $\alpha = 1.5e1$ results in a high quality style transfer effect with content and style well-balanced. Therefore, we choose the content weight $alpha = 1.5e1$ to be our general style transfer parameter for all other experiments.

5.3. Mixed Style Transfer

We trained the network for 7 styles. For each content image seen at training time, a conditional vector was randomly generated to choose a style. To evaluate our mixed style transfer implementation, we chose 4 style images and tested how mixed style transfer works on these 4 styles. To eliminate the effects of content images, we chose to apply these styles to one single content image. The content image we chose is shown in Figure 9. The resulting stylized images are shown in Figure 10, where the style images are the 4 images shown at each of the four corners, and the 9 images shown at the center are the stylized images. The 4 stylized images closest to the style images are the stylized images resulted from style transfer consisting of one style. Each image between those 4 stylized images is the mix-stylized image with 50% style weight for each of the two nearest single style images. Finally the image in the center is the mixed-style image of the four style images with 25% weights. As seen in Figure 7, the mix-stylized result of 2 style images consists of the style textures from both of the style images. For example, the left image, which is stylized from two style images - *Shrimp* and *Rain Princess* - depicts the traditional Chinese paint style but with more color and warmth from the style of *Rain Princess*. Another interesting result is the stylized image in the center; after careful examination of the image, we could identify all 4 style textures in the same image, even though their contributions might appear subtle at first glance.

Note that the network runs in approximately 1.5s on our Tensorflow implementation with the addition of conditional vectors. It is possible to achieve real-time performance in an optimized scenario. This implementation of the style transfer technique has the advantage of using less memory and

being able to mix multiple styles. It was demonstrated in [5] that a spatial mix can be achieved with non-uniform conditional vectors, but this lies outside the scope of this project.

Figure 9: Content image for mixed style transfer.

Figure 10: Mixed style transfer images with style images on the four corners.

5.4. Style Transfer for Arbitrary Styles

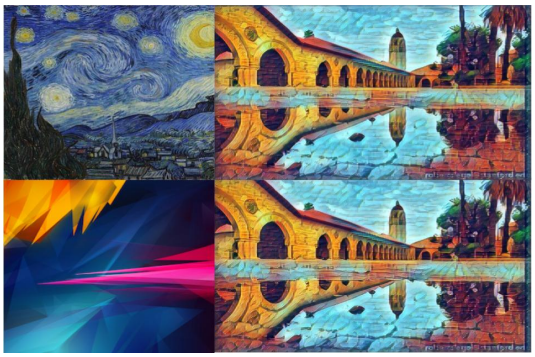
By applying the implementation described in Section 4, we trained the network with 4 styles. Since Tensorflow only supports static graphs, the style extraction networks are built for each style target although they share the same set of variables. This causes more GPU memory to be occupied with more styles, hence the limited styles. Similar to training a fast mixed style transfer network with conditional vectors, for each content image seen at training, one style image was randomly chosen as the style target.

The results are shown in Figure 11 and 12. This network was trained more efficiently and thoroughly by preprocessing the style images to have the same size and feeding them into the same computational graph.

We can see that the network overfits to the styles seen at training time and failed to stylize the image to arbitrary input styles, which appeared random to the style extraction network. This is because the original work [7] used 80,000 style images, while we were only able to train 5 due to hardware restrictions.

Figure 11: Content image for arbitrary style transfer.

(a) seen styles



(b) unseen styles

Figure 12: Style transfer with arbitrary styles

6. Conclusion

In this project we implemented fast style transfer as in [2]. We tuned the weights of different components of the loss and concluded that the image quality is sensitive to the weight. We also implemented mixed style transfer with conditional vectors [5] to train on 7 style targets. The mixing of styles was achieved by manipulating the conditional vector at test time.

Finally, we extended the technique of arbitrary style transfer [7] by replacing the pretrained Inception v3 network with a lightweight trainable network and demonstrated the possibility of solving the arbitrary style transfer problem with a smaller network. Future work involves training the network on a large number of styles and experimenting with different architectures of the style prediction network and different methods of integrating the style prediction network into the feed-forward network.

References

[1] Image Style Transfer Using Convolutional Neural Networks, Gatys et al, CVPR 2016

[2] Perceptual Losses for Real-Time Style Transfer and Super-Resolution, Johnson et al, arXiv: 1603.08155

[3] Very Deep Convolutional Networks for Large-Scale Image Recognition, Simonyan et al, arXiv: 1409.1556.3

[4] Fast Style Transfer, Logan Engstrom, `https://github.com/lengstrom/fast-style-transfer/`, 2016

[5] Unseen style transfer based on a conditional fast style transfer network, Yanai, Keiji, 2017

[6] Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization, Xun Huang, Serge Belongie, arXiv: 1703.06868, 2017

[7] Exploring the structure of a real-time, arbitrary neural artistic stylization network, Ghiasi et al, arXiv: 1705.06830, 2017