Real-time Image Style Transfer

Zhaoyou Wang, Zhaoheng Guo
Stanford University
Applied Physics
{zhaoyou,zhaoheng}@stanford.edu

Sophia Lu
Stanford University
Statistics
sophialu@stanford.edu

Abstract

Artistic style transfer has long been an interesting topic in computer vision research. Recently several methods for style transfer based on convolutional neural networks have been proposed. This project aims at understanding and implementing some of the existing methods. More specifically we succeed in implementing the optimization based neural algorithm as well as the real-time style transfer by training an image transformation network. We demonstrate similar results can be generated using both methods.

1. Introduction

Image style transfer is a long-standing problem that seeks to transfer the style of a reference style image onto another input picture. In conventional computer vision methods, it is difficult to extract the texture information and apply style to a new image. However, the developments and understandings of convolutional neural networks (CNN) provide a different approach to phrase and solve this problem. Recently several image style transfer algorithms based on CNNs have been proposed.

Our project basically contains two parts. First we aim to implement the neural style transfer algorithm by Gatys *et al* [1] which is based on defining the stylized image as the solution to an optimization problem. Then we compare it with the Fast-Neural-Style algorithm proposed by Johnson *et al* [4], which solves the optimization problem in real-time by training a feed-forward image transformation network.

2. Related work

Gatys *et al* [1] first suggested to generate style-transform images using *perceptual loss functions*. In their work, the perceptual loss functions are not based on the differences between pixels from the output and the ground-truth images. Instead, lower level features from a pretrained network are considered as content representation and the correlation between feature maps are used as style representation. The content loss together with style loss are minimized to realize style transfer. However, one drawback with this is the computational complexity since generating every image requires a optimization process from the very beginning. This could take minutes or even longer to do depending on the computation resource.

Johnson *et al* [4] have proposed to implement photo style transform by using feed-forward CNN. In their work, the system consists of an image transform network and a loss network. The image transform network is a feed-forward CNN parameterized by weights W, and the loss network calculates the loss based on perceptual loss functions. Given an artistic style, the image transformation network generates the output image from the input content image. Then a loss network would calculate the loss function which describe how well the output image combines both the content and style. During the training, the style image is fixed and we use a large dataset as content image. The weights W, hence the feed-forward networks, are trained by optimizing the perceptual loss function. With a well-trained image transform networks, it will directly transform an input image into an image containing both the content and the given style without solving an optimization problem every time.

This real-time algorithm achieves its speed with the compromise of flexibility, since for each style a separate image transformation network needs to be trained. A recent work [2] from Google demonstrates the possibility of combining the flexibility of the neural algorithm of artistic style with the speed of fast style transfer networks to allow real-time stylization using any content/style image pair. The usage of large number (80000)

of paintings as the style images in the training is one of the key factors for their success.

Another very interesting work [6] focuses on improving the artistic style transfer to realize photo style transfer. By defining photorealistic loss, they confine the transformation to be locally affine in colorspace which prevent local edges in original content image to be distorted. They also introduce semantic segmentation to ensure that style transfer happens between similar semantic components which respects the semantics of the scene and prevents spillovers. However, this method is based on optimization and there hasn't been a real-time algorithm for this so far.

When applied to video stylization, the real-time algorithm has instability problem in the sense that two adjacent frames might have different local structure. In [3], a recurrent convolutional network is implemented which incorporates a temporal consistency loss to suppress the instability of prior methods.

3. Methods

The style transfer problem can be defined as: given content image y_c and style image y_s, how can we find a new image y that is both close to y_c in content and y_s in style. The key point in solving this problem is to mathematically define the content difference distance between two images as well as the style distance between two images. Therefore by reducing the two distances together, we generate a solution y which is the stylized image.

One obvious way to define the content distance is to use per pixel loss. The problem with this though, is that even the two images with the same content are relatively shifted by a few pixels, the distance might already be quite large. Another problem is that defining style distance in per pixel way is quite difficult. To solve these problems, instead of using per pixel loss, we will use the *perceptual loss* [1, 4]. More specifically, a pre-trained VGG16 network is used to extract feature responses from the input image and the distances between two images are defined as the difference between their feature responses at different VGG layers.

Content Reconstruction Loss. In a convolutional neural network, an input image x is encoded in each layer by the filter responses to the images.Suppose that in the layer l we have N_l distinct filters and M_l is the total length of the feature response. The response in a layer l can be stored in a matrix $F^l \in \mathcal{R}^{N_l \times M_l}$ where F^l_{ij} is the activation of the i^{th} filter at position j in the layer l. If we have an input image y and a generated image $\hat{y}$, the respective feature representations of them are $F^l_{ij}(y)$ and $F^l_{ij}(\hat{y})$. Then the content reconstruction loss in the l^{th} layer is defined as:

$$\mathcal{L}^l_c(y, \hat{y}) = \frac{1}{N_l M_l} \sum_{ij} (F^l_{ij}(y) - F^l_{ij}(\hat{y}))^2. \quad (1)$$

The definition of Eqs. 1 encourages the input and generated images to have similar feature representations, rather than forces them to match pixel-by-pixel.

Style Reconstruction Loss. The style representation is built by computing the correlations between different filter responses. The feature correlations are given by the Gram matrix $G^l \in \mathcal{R}^{N_l \times N_l}$, where G^l is calculated as the correlation between feature responses F^l:

$$G^l_{ij} = \frac{1}{N_l M_l} \sum_k F^l_{ik} F^l_{jk}. \quad (2)$$

where the normalization factor comes from averaging over different filters and the length of the feature response.

If we have an input image y and a generated image $\hat{y}$, the respective Gram matrices of them are $G^l_{ij}(y)$ and $G^l_{ij}(\hat{y})$. Then the style reconstruction loss in the l^{th} layer is defined as

$$E_l(y, \hat{y}) = \sum_{ij} (G^l_{ij}(y) - G^l_{ij}(\hat{y}))^2, \quad (3)$$

and the total style reconstruction loss is the weighted summation of the style reconstruction loss of every layer,

$$\mathcal{L}_s(y, \hat{y}) = \sum_l w_l E_l. \quad (4)$$

Total variation loss. There's one more loss to mention which gives penalty to local variation of the generate image y. For one dimension array it's defined as $\sum |y_{i+1} - y_i|^\beta$, where β is usually chosen to be 1 or 2. This can be easily generalized to two dimensional images. The total variation loss is useful to suppress noise of the output.

In order to generate the images that mix the content and the style, we combine Eqs. 1and Eqs. 4 together to construct a joint loss function. Let y_c and y_s represent the images that contribute the content and style respectively, we can define the joint loss function as

$$\mathcal{L}_{\text{total}}(y_c, y_s, y) = \lambda_c \mathcal{L}_c(y_c, y) + \lambda_s \mathcal{L}_s(y_s, y) + \lambda_{TV} \mathcal{L}_{TV}(y), \quad (5)$$

where the weights are hyper-parameters that need to be decided later on. The output image $\hat{y}$ is generated by optimizing the Eqs. 5:

$$\hat{y} = \arg\min_y \mathcal{L}_{\text{total}}(y_c, y_s, y). \quad (6)$$

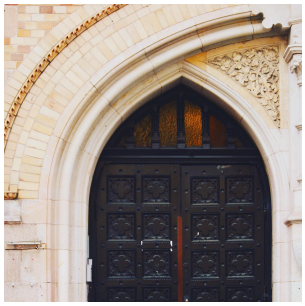

Figure 1. Original content image.

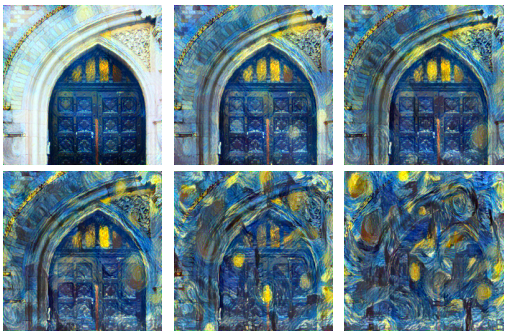

Figure 2. Neural algorithm with different relative style weights: 5×10^4, 2.5×10^5, 5×10^5, 10^6, 2.5×10^6, 5×10^6. Total variation loss is fixed at 10^{-5}.

Figure 3. Neural algorithm with different total variation loss weights: 0, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}. The relative style weight is fixed to be 10^5 here. For small TV loss weight, the image contains noise locally, while for medium values, the image is smoothed and for very large TV loss, the whole image becomes almost the same value.

Real-time algorithm. Even though the neural algorithm could generate nice stylized images, it's very computationally expensive since every time an optimization problem needs to be solved. Johnson *et al* further come up with a different approach of style transfer by training an image transformation network as generator and eventually generate stylized images in a feed-forward way. The basic idea is to fix the style image y_s and use a large amount of content images to train the generator so that it generalizes well to unseen content images. One drawback with this approach is the compromise in flexibility since for each given style a different image transformation network needs to be trained.

Let the image transform network be parameterized by W and the output of it denoted by $f_W(x)$ with an input image x. Then loss function is the same as before: $\mathcal{L}_{\text{total}}(y_c, y_s, f_W(y_c))$. By training the network weights W with this perceptual loss function, eventually we would get a network that learns to add certain style to the input content image.

4. Experiments

4.1. Neural style transfer

The neural algorithm [1] for style transfer is implemented first as a baseline. There are many hyperparameters that requires tuning by hand. The most important one is the relative weight between content loss and style loss. The total variation loss weight will change how noisy the generated image is. Also following the choice in [1], we treat loss from different VGG layers with the same weight.

Weights sweeping. We run the neural algorithm on the content image 1 with different relative style weight and the sweeping results obtained is shown in Fig. 2. As expected, the style weight controls how much the content image gets stylized and at very large style weight, the content information is almost completely lost. This experiment result can be used in choosing the visually best style weight, which is very useful for training the real-time image transformation network later on. Usually it takes a few minutes to converge on a Tesla K80 with 300-500 iterations while for the real-time approach, it takes a few hours to get feedback about whether a parameter choice is good or not.

We can also perform the same experiment to pick the total variation loss weight. From Fig. 3 we can see that a proper weight could both suppress noise of the generated image and without forcing most the pixels to have similar values. However, later on it turns out that the total variation loss is more important for neural algorithm than the real-time approach, since in neural algorithm the optimization starts from a blank image with only white noise which requires extra penalty to remove while in real-time style transfer, the input to the image transformation network is always the content image itself and therefore suppressing noise is not as important here.

Reconstruction from different layers. Another way to better understand the neural algorithm is to look at the reconstructed image for content (set style weight to be 0) or style (set content weight to be 0) alone from different

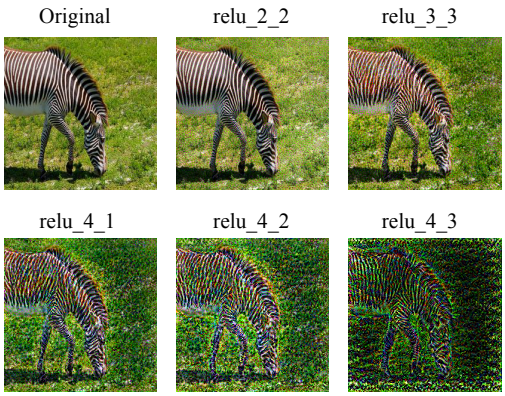

Figure 4. Content reconstruction from different layers of the VGG16 network.

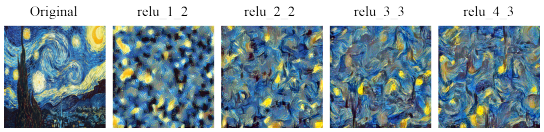

Figure 5. Style reconstruction from different layers of the VGG16 network. Different from the content case, this time we use multiple layers: relu_1_2, relu_1_2+relu_2_2, relu_1_2+relu_2_2+relu_3_3, relu_1_2+relu_2_2+relu_3_3+relu_4_3.

layers of the VGG16 network output. More specifically, we reconstruct the content image from layers 'relu_2_2', 'relu_3_3', relu_4_1', 'relu_4_2' and 'relu_4_3'. As we can see from Fig. 4, for lower level feature maps, details of the original content image can be reproduced while for higher level feature maps detailed information is lost and the overall profile is kept. Similarly we can reconstruct from style image only. From Fig. 5, when higher level features are used, only the overall style information is kept without details in the original style image. This result gives instructions on how to choose the VGG layers. For style transfer purpose, we want to superpose an overall style onto the content image while keeping sufficient details. Therefore, a reasonable choice would be use relu_2_2 as content layer and relu_1_2+relu_2_2+relu_3_3+relu_4_3 as style layers.

4.2. Real-time style transfer

We implemented a image transformation network with 3 convolution layers, 5 residual layers and 3 deconvolution layers. This architecture is similar to the one used in [4], with the differences that instance normalization instead of batch normalization is used to achieve better results [8] as well as replacing transpose convolution with a nearest neighbor upsampling followed by a convolution, which is shown to be able to remove the checkerboard artifacts caused by transpose convolution [7].

4.2.1 Training details

We resize the 80k images to 256×256 in Microsoft COCO [5] training set and train our network with batch size 4 and two epochs over the training data. The content weight is set to be 1 and the style weight is roughly between 10^4 and 10^5 depending on which style we choose. The total variation weight is set to be 10^{-7} which doesn't seem to influence the result much since during training we start from content image instead of pure noise as input to our image transformation network. As stated before, we choose relu2_2 for content loss, relu1_2, relu2_2, relu3_3, relu4_3 for style loss with equal weights and use Adam with learning rate 10^{-3}. Our implementation is based on PyTorch and the training takes about 6 hours on Tesla K80 GPU.

Some small details that have big impacts on the final results:

1. Preprocessing the input images matters, *i.e.*, subtract the mean values and properly normalize the data of each color channel. The reason is that pretrained neural networks depends on input image having certain distribution to correctly excite the neurons and extract lower level features.

2. Since the last layer is a convolution, the output image is not guaranteed to be bounded for RGB display. Projecting the values in range [0, 255] and also converting to uint8 before showing the image could get rid of noisy artifacts and save unnecessary debugging time.

3. Although not very clear why this happens, from comparing experiment results we find that allowing the image transformation network to work in the range of [0, 255] instead of rescaling all the images between [0, 1] actually generates much better results and also gives better stability during training.

4.2.2 Results

We trained the image transform network for 3 different styles and the comparison between the real-time results and the baseline results are shown in Fig 6. The style weights for each result are 10^5, 8×10^4 and 8×10^3 respectively.

Figure 6. Comparison between baseline results and real-time results for 3 different styles.

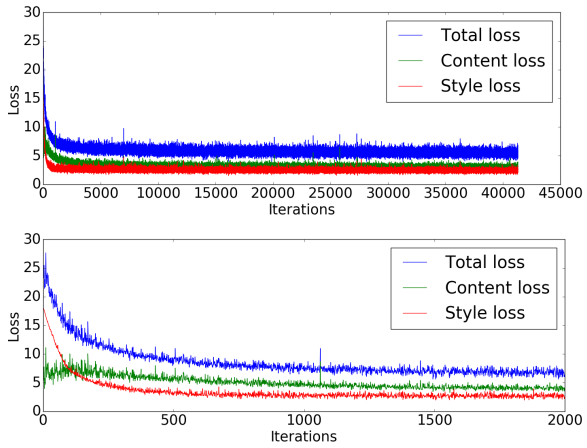

Figure 7. Loss function during the whole training process (top) and zoom in of the beginning stage (bottom).

From the comparison, out real-time image transform network could generate good quality images compared with the baseline results. The baseline results seems slightly noisy which is possibly caused by the small total variation regularization we used, since the optimization starts from a random noisy image.

A typical learning curve is shown in Fig. 7. The loss drops very quickly during the first few thousands iteration and then decreases very slowly learning to generalize well. Also as expected, at the beginning the style loss drops monotonically from a high value while the content is loss much lower (since we start from the content image as input) and then decrease slowly.

5. Conclusion

So far we have successfully implemented the neural style transfer as well as the real-time style transfer. We have also made some minor modifications compared to the original paper to achieve better results based on some later work. The comparison between their results shows that the real-time image transformation network is working pretty well.

For future works, there are several things that seem interesting:

1. Implementing the photorealistic style transfer and training a real-time feed-forward network for it.

2. Modifying the structure of the image transform network may generate better results. One possibility is to add extra **ATTENTION** layers to apply different weights to different features.

3. Better understand how arbitrary style transfer works and explore the possibility of it in the context of photorealistic style transfer instead of paintings.

6. Contribution

Zhaoyou Wang implemented the PyTorch code and wrote the experiments related parts in the manuscript. Zhaoheng Guo made the poster and wrote the other parts of this manuscript. Sophia Lu suggested this project and did some early stage literature research.

The authors want to thank Yang Song for his generous share of computation resources.

References

[1] L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[2] G. Ghiasi, H. Lee, M. Kudlur, V. Dumoulin, and J. Shlens. Exploring the structure of a real-time, arbitrary neural artistic stylization network. *arXiv preprint arXiv:1705.06830*, 2017.

[3] A. Gupta, J. Johnson, A. Alahi, and L. Fei-Fei. Characterizing and improving stability in neural style transfer. *arXiv preprint arXiv:1705.02092*, 2017.

[4] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, pages 694–711. Springer, 2016.

[5] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[6] F. Luan, S. Paris, E. Shechtman, and K. Bala. Deep photo style transfer. *arXiv preprint arXiv:1703.07511*, 2017.

[7] A. Odena, V. Dumoulin, and C. Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016.

[8] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.