Art Style Transfer:
A Replication for A Neural Algorithm of Artistic Style

E4040.2017Fall.APPP.report
Liutong Zhou, Zeyu Ye, Yaqing Wang
Columbia University

Abstract

Neural Style Transfer has shown very promising results in producing artistic-style images since Gatys et al. [1] proposed the original algorithm. In this paper, we introduce the application background of this algorithm, review and dissect the methods presented in the original work. Then, we provide a handy implementation based on top of tensorflow and reproduce the results. Furthermore, we compare the outcomes and explore possible extensions through experiments. Finally, we discuss the current limitations and provide directions for improvements in future work. This paper aims to introduce the neural style transfer and showcase the artistic effects it can produce.

1. Introduction

Neural Style Transfer is a pioneering algorithm that utilize the power of Convolutional Neural Network (CNN) to reproduce artistici-style images. It achieves so by separately extracting the neural representations of the content of a given photograph and the style of of an artwork, and fusing the content with the style. Since Gatys et al. [1] first proposed the neural algorithm for the creation of artistic images, it has been a hot topic both in academic literature and in industrial applications. Many follow-up works[2–4] in art style transfering are based on the finding that the CNN representations of content and style of an image are separable, which enables independent manipulations of style and content, specially creating perceptually amusing images. As of now, the latest update in Neural Style Transfer has introduced methods for a more fine-grained control over image stylization in different spatial regions and different spatial scales [5]. Jing et al. [6] provide a thorough review of the current progress in this research field.

In this project, we mainly focus on the initial work[1]. The goal is to reproduce similar results with those of their original work. In addition, we provide a handy implementation based upon tensorflow for ease of use *and explore possible extensions through experiments*.

The paper is organized as follows. In section 2, we summarize the methods and results of the original paper. In section 3, we lay out the road map for our work. In section 4, we present the implementation details. In section 5 we analyze and compare our outcomes with the original ones. Finally in section 6, we discuss our findings, *the current limitations and then provide directions for improvements in future work.*

2. Summary of the Original Paper

The original paper [1] proposed a method to generate a new image that simultaneously preserves the content of the input photograph and captures the style of the input artwork. In this section, we first articulate the mechanism of this approach and then showcase the experimental results from the paper.

2.1 Methodology of the Original Paper

Based on the key finding that the content and style of an arbitrary image can be decoupled in a CNN, the author proposed a method to generate a new image by synthesizing the style of a artwork and the content of a photograph. The overall process is outlined as follow: (i) find feature representations for the style of the artwork (style features) (ii) find feature representations for the content of the photograph (content features) (iii) start from a random white-noise image, iteratively update each pixel of the random image to penalizing the total loss, which is defined as a combination of (a) the difference between the style features of the generated image and that of the artwork (b) the difference between the content features of the generated image and that of the input photograph. Specifically, given a content image x_c, and a style image x_s the algorithm finds the new image x such that

$$x = arg\ \min_{x}\ L_{total} \tag{1}$$

where the total loss L_{total} is a linear combination of content loss and style loss

$$L_{total} = \alpha L_c(x, x_c) + \beta \mathrm{L}_s(x, x_s) \tag{2}$$

with α, β denoting the weighting factor for content loss and style loss respectively.

A feature map of an image x is the output from a hidden convolutional layer. If the convolutional layer l has N_l kernels, the feature map extracted from this layer is a matrix $F^l \in R^{N_l \times M_l}$ such that the ith row of F^l is the flattened output of the ith convolution with the ith kernel.

The content loss is thus defined as an average Euclidean distance between the feature map of the photograph and that of the image to be generated.

$$L_c(x, x_c) = \frac{1}{n} \sum_{l=1}^{L} \frac{1}{2} \left\| F^{(l)}(x) - F^{(l)}(x_c) \right\|_F^2 \tag{3}$$

A style feature of an image extracted from a hidden convolutional layer l is the covariance matrix $G^l \in R^{N_l \times N_l}$

$$G^l = \frac{1}{M^l} (F^l) \cdot (F^l)^T \tag{4}$$

so that $G^l_{i,j}$ measures the covariance between the i th filter response and the j th filter response at convolution layer l.

The style loss is then defined as the weighted sum of the Euclidean distance between the style feature of the artwork and that of the image to be generated.

$$L_s(x, x_s) = \sum_l^L w_l \, L_s^{(l)}(x, x_s) \tag{5}$$

where

$$L_s^{(l)}(x, s_s) = \frac{1}{4N_l^2} \left\| G^{(l)}(x) - G^{(l)}(x_s) \right\|_F^2 \tag{6}$$

It is worth mentioning that matching the Gram matrices is equivalent to minimizing the Maximum Mean Discrepancy (MMD)[1] with the second order polynomial kernel between two distributions, which has been proved theoretically in [7]. Therefore, matching the Gram matrices (4) is equivalent to matching the feature distributions between the style images and the generated images.

The original work used the VGG19 CNN [8], which consists of 16 convolutional and 5 pooling layers to extract image feature maps. A detailed illustration of the architecture is provided in Section 4 (figure 2). The pretrained weights is open to public and available online.

2.2 Key Results of the Original Paper

The original paper is the first to derive the neural representations of content and style of images. They provided a pioneering neural algorithm in separating content from style in whole natural images. Their method showed very promising experimental results in rendering a given photograph in the style of well-known artworks to achieve artistic style transfer. Figure 1 presents the visually appealing experimental results from the original work.

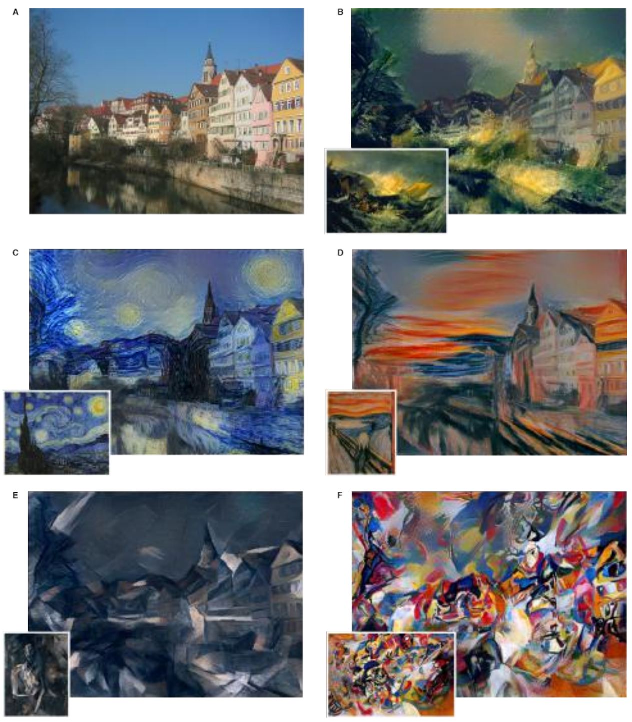


Figure 1 Photograph (A) rendered in the styles (at bottom left corner) of well-known artworks. The styles are (B) *The Shipwreck of the Minotaur* by J.M.W. Turner, 1805 (C) *The Starry Night* by Vincent Van Gogh, 1889 (D) *Der Schrei* by Edvard Munch, 1893 (E) *Femme Nue Assise* by Pablo Picasso (F) *Composition VII* by Wassily Kandinsky. (Figures are from the original paper [1])

3. Methodology

In this section, we we lay out the road map for this project. First, we declare the objectives of our work. Second, we outline our system design. Specially, we articulate the difference between our work and the original work and explain why we think our choice may be an improvement.

3.1. Objectives

Our work directly follows the original paper. The aim is to reproduce similar or even better results in the artistic style transfer stask. Our goals are listed below (i) Provide a user-friendly implementation of the original neural

[1] "Kernel embedding of distributions - Wikipedia." https://en.wikipedia.org/wiki/Kernel_embedding_of_distributions.

algorithm on top of tensorflow. (ii) Reproduce the experimental results using the same setting as proposed by the original paper (iii) Explore how different settings of parameter affect the visual results to get insight of the algorithm. (iv) Find possible extensions through experiments, discuss the current limitations and provide directions to improvement.

3.2. Problem Formulation and Design

Our design mainly follows the original work. Some slight changes are introduced according to recent findings that improve the original work . These changes are listed below

(i) Different from the original paper, we add a total variation denoising term to the total loss (equation 2) to help improve the overall quality of the generated image, as is suggested in [6]. The total variation denoising term is defined as the sum of the absolute differences for neighboring pixel-values in the input images. Tensorflow has provided an implementation of this function.

(ii) Instead of limiting the input style to a single artwork, we allow input of multiple artworks in order to simultaneously learn and combine multiple styles, so that the synthesized image can have multiple styles. This is achieved by simply changing the style loss (equation 5) to be a weighted sum over all input style artworks.

(iii) Besides the second-order[2] optimization algorithm L-BFGS, which is chosen as the optimizer in the original paper, we introduce the adaptive first-order optimizer Adam as an alternative. We aim to compare the pros and cons of these two optimization algorithms in the art style transfer application

The whole system is designed to follow scikit-learn's syntax. In a nutshell, the whole art style transfer process can be done in three steps: (i) instantiate an Artist class (ii) call the method fit_transform to optimize the loss (iii) call draw to visualize the results. A detailed demonstration is provided in Appendix 8.2.

4. Implementation

This section describes the details of our implementation. First, we talk about the implementation of the VGG19 class, which lays the foundation for image processing and feature extraction. Then we introduce the architecture of our software design and a detailed view of how the algorithm is implemented.

4.1. Deep Learning Network

Directly following the original paper, we build the VGG19 deep learning model for extracting image feature maps. The architectural block diagram is shown in figure 2. The whole deep architecture consists of 5 sequential groups of layers, each of which is a tack of convolution layers and pooling layers. Specifically, the model is built by stacking the following layers: input -> [conv -> conv -> pool] *2 -> [conv -> conv -> conv -> conv -> pool]*3, where each convolution layer uses 1-by-1 sized kernels that moves spatially with full stride. After convolution, the outputs are padded spatially to remain the same size, and are fed into the pooling layer. All pooling layers share the same setting, a 2-by-2 filter size and a predefined aggregation mode of either average or max.

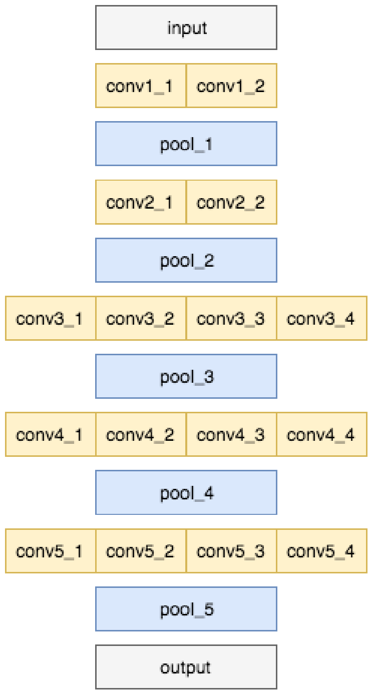


Figure 2. The whole structure of the network

4.2. The Neural Style Transfer Algorithm

The overall style transfer algorithm is implemented in three steps. First, get the content representations for the photograph and the style representations for the artwork (equation 4). Second, get the content loss (equation 3) and style loss (equation 5), and hence the total loss (equation 2). Third, optimize the total loss over the image to be generated (equation 1). Figure 3 provides an flow procedure of the algorithm.

Specifically, we set the output of the 'conv4_2' layer as the content features. The output of the 'conv2_1', 'conv3_1', 'conv4_1', and 'conv_5_1' layers are fed into equation 4 to get style features. We used the pixel-wise squared L2 loss as the loss function and updated parameters with L-BFGS optimizer.

---

[2] L-BFGS is based on Hessian. Theoretically it provides a better estimation at every update step. Though calculating Hessian is more computation-intensive, each update step is adjusted to converge better towards the local minima.

In addition to the above default setting, we provide many alternative options: (i) we provide Adam as an alternative optimizer. (ii) we provide three kinds of content loss functions, each of which used a different normalization coefficient, namely no normalization, normalize by filter map size and normalize by the square root of filter map size (iii) in addition to the random initialization, we provide the choices to initialize the target image as content image or style image. (iv) we provide the flexibility to change content weight, stylte weight, and layers to extract content and style features. (v) we provide the functionality to use multiple style images at the same time to concurrently form the target image so that it can have multiple styles. This is achieved by simply changing the style loss (equation 5) to be a weighted sum over all input style artworks.

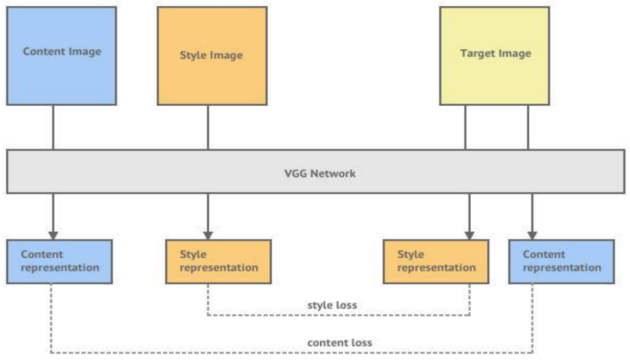


Figure 3. Style transfer algorithm

In a practical experiment, we adopted the following procedure to reproduce similar results as those of the original paper. First, we trained the model for 1000 iterations using avg pooling, as described in the original paper, to generate image of different styles. Then we set the ratio of content weight and style weight as 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, and matched the style representation on (A) layer 'conv1_1', (B) layer 'conv1_1' and 'conv2_1', (C) layer 'conv1_1', 'conv2_1' and 'conv3_1', (D) 'conv1_1', 'conv2_1', 'conv3_1' and 'conv4_1' (E), 'conv1_1', 'conv2_1', 'conv3_1', 'conv4_1', 'conv5_1'. For each pair of choices, we trained the model for 4000 iterations and saved the final output image for comparison. At last, we trained the model for 4000 iterations using max pooling and avg pooling as pooling layers, respectively. In the process of training, we record the loss and output image each 100 iteration for comparison.

4.2. Software Design

Our software is initially designed to follow the syntax of scikit-learn, for the purpose of user friendliness. The so called *styletransfer* package consists of 5 modules, namely: (i) artist (ii) io (iii) layers (iv) vgg19 and (v) optimizer. A hierarchy diagram of the whole package is attached in Appendix 8.2. All modules, except artist, are intentionally designed to be mutually independent and exclusive, for the purpose of mobility, extensibility and ease of maintenance.

To ensure the visual quality of generated image and ease of use for our users, we fine tuned the default settings including the default weights, initialization type, layers for matching the style representations, and the ratio of content weight and style weight.

In general cases, a user of our package only needs to go through three steps to accomplish the intractable style transfer task. (i) Instantiate an artist object of the class Artist (ii) Call method fit_transform, just like what a user of scikit-learn would normally do (iii) Upon completion of training, call draw() method to get the output image. We provide a demonstration of the usage of our package on jupyter notebook, which is hosted in our repository [9].

5. Results

5.1. Project Results

The major finding of the original paper is that content and style representations in the Convolutional Neural Network can be extracted separately. Thus, new images can be produced by adjusting both representations independently. To demonstrate this finding, the authors used a photograph of the "Neckarfront" in T¨ubingen, Germany and adopted several famous artworks to create new images containing the content of the original photograph but in different art styles.[1]

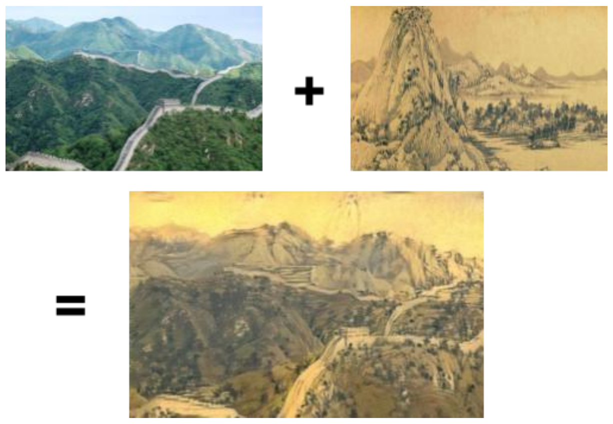


Figure 4. Our Art Style Transfer that maps the Style the Chinese painting named "Dwelling in the Fuchun Mountains" by Gongwang Huang (upper right) to Content the Great Wall (upper left), and forms the recomposed artistic-style image at the bottom

We demonstrate our success in extracting content and style from different images and combining them together to achieve style transfering in figure 4. For 4000 iterations, it took around 20 minutes on a node with Tesla K80 provided by Google Cloud, using L-BFGS optimizer.

For our project, we further chose photographs of different contents and paintings of different styles. Figure 5 shows our results of combining the photograph of "Donald Trump" and four different well-known artworks. The painting that provided the style for the respective generated image is shown in the corner of each new image. The top left is adopted from Composition VII by Wassily Kandinsky. The top right is from Guernica by Pablo Picasso. And the bottom left and right are from Green Mountain by Qiu Dingfu and The Starry Night by Vincent van Gogh respectively.

The results are delightful. In each generated image, you can see the obvious sketch of Donald Trump clearly, and they all blend well with the respective painting's style. We also did the same thing for other photographs using these styles, and they are all successfully rendered with the style of the artwork.

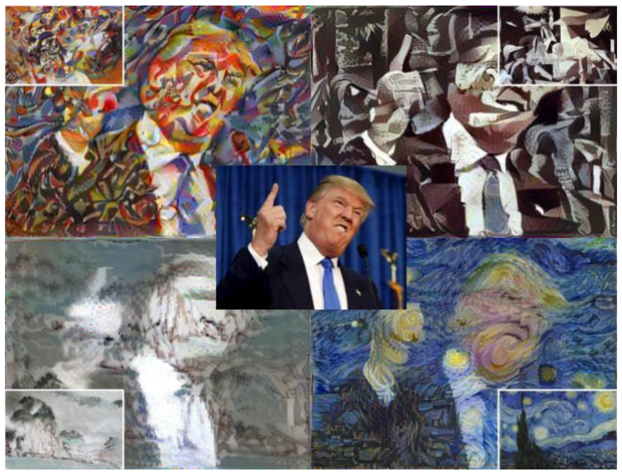


Figure 5: Images that combine the content of a photograph of Donald Trump with the style of several well-known artworks.

Next, we tried adjusting the content and style weight ratio while trying out different layers of CNN for style representation. Detailed results are demonstrated in figure 6. The columns shows the results of increasing relative weightings between content and style representation, and the numbers on the top indicate the ratio. With the number increases, the generated image reveals more content of the original photograph and emphasis less on the style of the artwork as expected. The rows use increasing layers of CNN for style construction (See Section 4.1 Deep Learning Network). It is obvious that the more higher layers are used, more complex the style is observed and merged, due to "the increasing receptive field sizes and feature complexity along the network's processing hierarchy" [1].

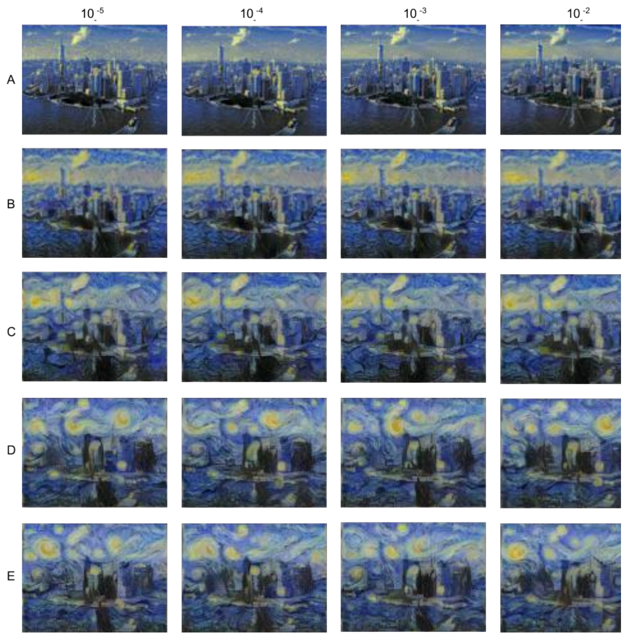


Figure 6: Detailed results for the style of the painting *The Starry Night* by Vincent van Gogh.

5.2. Comparison of Results

The original paper suggested that using average pooling instead of max-pooling would give a more appealing result [1], so in this section we tried to simulate these two methods and verify the authors' statement.

For both pooling methods, we adopted type (E) style representation and used 0.01 as the ratio of content and style weight. We recorded the resulting images for every 500 iterations. As seen in the Figure 7, implementation with average pooling blended in with the Starry Night style more thoroughly and quickly than the one with max pooling. Average pooling results in a more bluish picture than max pooling, and in the end of 4000 iterations, the edge of Alma Mater is blurred with the background, which is more intriguing than the other one.

In addition, we recorded and plotted the loss along the way for both methods. From the y-axis in figure 8, we can see that the loss generated from max pooling are more than 10 times larger than that of average pooling, and it converges slightly earlier than max pooling. Therefore, we can conclude that average pooling does perform better in this task than max pooling.

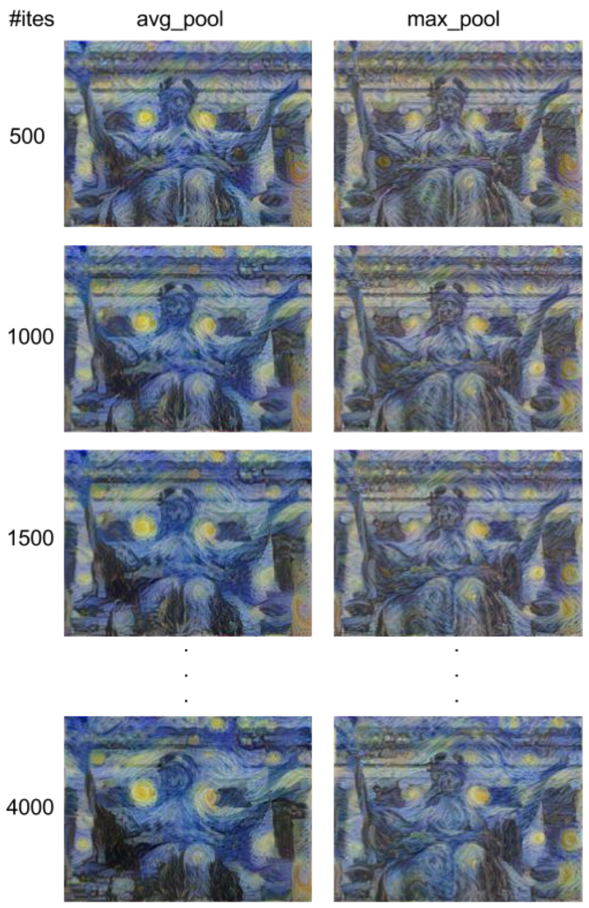
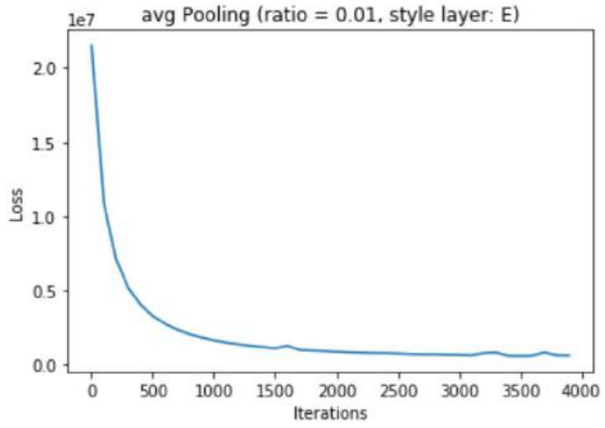
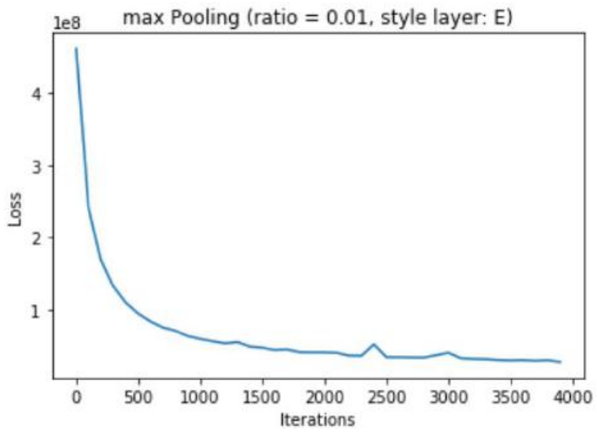


Figure 7 A comparison of visual effects under different pooling methods



(a) Loss curve under average pooling



(b) Loss curve under max pooling

Figure 8 A comparison of loss curves under different pooling methods

5.3. Discussion

Although the original algorithm has achieved remarkable results in art style transfer, there are several problems that remain to be solved.

First, the computation intensity is a major obstacle in face of industrial applications. Practical applications in the market expect real-time computation, while our experiment takes more than 10 minutes to get a satisfactory outcome.

Second, it usually requires a tedious work of parameter tuning in order to get a visually appealing result. This tuning process at least includes multiple rounds of adjustment of the content weight, style weight, layer weight and even the choice of representative layers for extracting features.

There have been related work both in improving the computation time, of which the method is known as Fast Neural Style Transfer [10,11], and in mitigating the necessity of parameter tuning [12]. However, they either trade the flexibility in exchange for speed or needs more convincing explanation.

Albeit difficulties, the past three years have witnessed a thriving research trend in the academia and a huge passion from the industry. Some applications have been made online [13–16]. We believe this research field will remain at the forefront of computer vision and deep learning. Promising directions to improving the current work include but are not limited to finding real-time solutions for image synthesis, finding auxiliary strategies for parameter tuning and extending the style transfer mechanism to other image processing tasks.

6. Conclusion

In this paper, we reviewed the current research state of Neural Style Transfer, dissected the original Neural Style Transfer algorithm part by part, provided an enhanced yet user-friendly implementation of the algorithm on top of tensorflow, and successfully reproduced the original paper's results.

Based on our handy implementation, we experimentally explored how different settings of parameter affect the visual results and discussed the current limitations of the algorithm. Finally, we indicate potential directions for improvements and provide an outlook for future research.

7. References

1. Gatys LA, Ecker AS, Bethge M. A Neural Algorithm of Artistic Style [Internet]. arXiv [cs.CV]. 2015. Available: http://arxiv.org/abs/1508.06576
2. Gatys LA, Ecker AS, Bethge M. Image Style Transfer Using Convolutional Neural Networks. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016. pp. 2414–2423.
3. Li C, Wand M. Combining Markov Random Fields and Convolutional Neural Networks for Image Synthesis. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016. pp. 2479–2486.
4. Selim A, Elgharib M, Doyle L. Painting Style Transfer for Head Portraits Using Convolutional Neural Networks. ACM Trans Graph. New York, NY, USA: ACM; 2016;35: 129:1–129:18.
5. Gatys LA, Ecker AS, Bethge M, Hertzmann A, Shechtman E. Controlling Perceptual Factors in Neural Style Transfer. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2017. pp. 3730–3738.
6. Jing Y, Yang Y, Feng Z, Ye J, Song M. Neural Style Transfer: A Review [Internet]. arXiv [cs.CV]. 2017. Available: http://arxiv.org/abs/1705.04058
7. Li Y, Wang N, Liu J, Hou X. Demystifying Neural Style Transfer [Internet]. arXiv [cs.CV]. 2017. Available: http://arxiv.org/abs/1701.01036
8. Simonyan K, Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition [Internet]. arXiv [cs.CV]. 2014. Available: http://arxiv.org/abs/1409.1556
9. Zhou L. Replication-for-A-Neural-Algorithm-of-Artistic-Style [Internet]. Github; Available: https://github.com/LiutongZhou/Replication-for-A-Neural-Algorithm-of-Artistic-Style
10. Johnson J, Alahi A, Fei-Fei L. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. Computer Vision – ECCV 2016. Springer, Cham; 2016. pp. 694–711.
11. Ulyanov D, Vedaldi A, Lempitsky V. Instance Normalization: The Missing Ingredient for Fast Stylization [Internet]. arXiv [cs.CV]. 2016. Available: http://arxiv.org/abs/1607.08022
12. Risser E, Wilmot P, Barnes C. Stable and Controllable Neural Texture Synthesis and Style Transfer Using Histogram Losses [Internet]. arXiv [cs.GR]. 2017. Available: http://arxiv.org/abs/1701.08893
13. deepart.io - become a digital artist [Internet]. [cited 17 Dec 2017]. Available: https://deepart.io/
14. Style Transfer Using Deep Learning. In: Algorithmia [Internet]. [cited 17 Dec 2017]. Available: http://demos.algorithmia.com/deep-style/
15. Pikazo - Make Anything Art. In: Pikazo App [Internet]. [cited 17 Dec 2017]. Available: http://www.pikazoapp.com/
16. Instapainting [Internet]. [cited 17 Dec 2017]. Available: https://www.instapainting.com/ai-painter

8. Appendix

8.1 Individual student contributions - table

	zy2232	lz2484	yw2902
Last Name	Ye	Zhou	Wang
contribution	1/3	1/3	1/3
Responsibility 1	Package development and algorithm implementation with a focus on VGG19 module	Package development and algorithm implementation with a focus on architecturing and the Artist module	Package development and algorithm implementation with a focus on IO, Optimizer, layers modules and the jupyter notebook demo
Responsibility 2	Robustness control and bug fix	Documentation and secondary encapsulation	Cross Platform Testing and bug fix
Responsibility 3	report writing	report writing	report writing

8.2 The hierarchy diagram of the styletransfer package