Traditional Chinese Ink Painting Neural Style Transfer

Pu Ke
Stanford University
450 Serra Mall, Stanford, CA 94305
pk2532@stanford.edu

Abstract

Style transfer allows one style of an image to be transferred and applied to another image. A new research area called neural style transfer started to gain popularity since 2015 which makes use of the power of deep neural networks. Since then many researches in this area emerged, but most of them are trained and evaluated based on Western style images, and only a few discussed the neural style transfer applied to the traditional Chinese ink paintings. Traditional Chinese ink paintings are in a very different style because of its unique drawing techniques. In this project, we train models of several classical and popular neural style transfer networks, including the work by [3], [11] and [21] with traditional Chinese ink paintings, then we evaluate and compare the results. We also modify each of these methods by passing the edge detection map of the content images using HED [17] to evaluate if it can improve the results.

1. Introduction

Style transfer allows one style of an image to be transferred and applied to another image. In inference time, the input is a content image and a style image, and the output is an image with the style of the input style image but with the content of the input content image. Traditional approach like supervised learning does not work well for this problem. To use supervised learning we need a pair of input content image and the expected output image, but those kinds of data are very rare and hard to find.

In 2015 Gatys et al. [3] first make use of the power of convolutional neural networks to generate paintings with applying styles from famous Western artists. This first work proved that convolutional neural networks are capable to extract style and content features from images and opened a door for following investigations in the neural style transfer area.

However, most of those works only evaluate style transfer using the style images from Western artists and rarely discuss the results using traditional Chinese ink paintings. Actually the drawing techniques used in traditional Chinese ink painting could be very different from the Western paintings.

For example, traditional Chinese ink painting applies a technique named 'void', which means it leaves a lot of blank space in the painting. An example would be we can hardly find artists drawing sun, moon or clouds in those paintings. Very often the sky is just empty and takes half of the space in the whole painting. Traditional Chinese ink painting also focuses more on the strokes and lines. The color used in those paintings is also very different. Ink black is used to draw most of the objects in the painting and the very few colored objects also often comes with a gray shade.

Due to the large difference of drawing techniques, networks and hyperparameters that work well for Western color paintings may not work well on the traditional Chinese ink paintings.

In this project, we train and evaluate different classical and popular neural style transfer models and our modified versions on traditional Chinese ink paintings and compares the results both quantitatively and qualitatively. Specifically, we evaluate three models and their modified version:

- [3], which is often called original neural style transfer. This method is online, so we don't specifically separate training/testing time. The input is a content image and a style image. The output is the stylized image.
- [11], which is often called fast neural style transfer. At training time, the input is a set of content images and a style image, and the output is a model. At testing time, the input is a content image, and the output is the stylized image.
- [21], which is often called CycleGAN. At training time, the input is a set of content images in the one domain and a set of style images in the other domain, and the output is a model. At testing time, the input is a content image, and the output is the stylized image.

Apart from comparing those original works, we also make a modification to each method that instead of feeding in the original content images, we feed in the edge maps retrieved using a pretrained holistically-nested edge detection (HED) model [17], and evaluate how they work for traditional Chinese ink paintings. The intuition behind is that traditional Chinese ink paintings are mostly in black and white and have a lot of 'void' space, so we probably don't want to preserve color or too many irrelevant details from the content images.

2. Related Work

In 2015, Gatys et al. [3] first bring up the basic structure for style transfer using convolutional networks, and then extend on it in [2] [5] [4]. In a deep convolutional networks, some layers learn to retrieve semantic contents of an image while some layers focus more on the the texture of the images. By passing a content image and calculating the distance between the output image and the content image in those layers that look more on the semantic contents, and by passing a style image and calculating the distance between the output image and the content image in those layers that look more on the texture, we can summarize a loss weighted on those two kinds of distances and update on the output image to minimize the loss.

However, this kind of online method usually requires longer time in inference time as we need to continuously do forward update and back propagation to iterate on the output image, thus more efficient models [11] [14] [15] [16] are brought up which are usually mentioned as fast neural style transfer or offline neural style transfer. In those models, a feed forward network is added and trained to generate the output model. Nowadays usually an encoder-decoder architecture is used as this feed forward network [13]. During training time, the feed forward network is trained and the pretrained network used to provide content loss and style loss is kept unchanged.

With the popularity of Generative Adversarial Network (GAN) on a lot of computer vision tasks, GAN also starts to play an important role in style transfer. many GAN models including cGAN, DCGAN, WGAN, StyleGAN, CycleGAN [10] have been actively utilized in the style transfer tasks.

Apart from per style per model neural methods mentioned above, multiple styles per model and arbitrary style per model are also explored [10]. [1] includes conditional instance normalization into the network, so it can take an additional vector input to provide the weight of each style that needs to be applied to the output image. [8] on the other hand can take any style image as the input, and lifts the limit that per style per model and multiple styles per model can only generate output images using the style that they have been trained on.

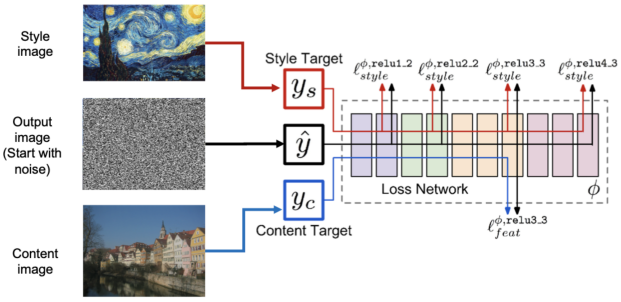

Figure 1. Architecture of original neural style transfer model. Figure is from cs231n lectures.

There are also a few works specifically in the ink painting style transfer domain. ChipGAN [7] introduces a model built upon CycleGAN [20]. Apart from the adversarial loss and cycle consistency loss CycleGAN has, it also adds on brush stroke loss and ink wash loss to favors the unique attributes of ink paintings. [18] investigates end-to-end painting generation using GAN. The work tackles the generation in two steps, first using SketchGAN to generate edges and then use PaintGAN to colorize the generated sketch. This work doesn't use any so-called content images. It only takes in style images and uses its edges as the ground truth for SketchGAN step. Using this model the input is a random noise instead of a concrete content image. Other works such as [19] also uses this two-step structure. Our work is inspired by all of those papers.

3. Methods

In this section, we will review the methods that we either train with or utilize in the code.

3.1. Original Neural Style Transfer (Gatys et al. [3])

In the Figure 1, a style image and a content image are fed into the network. The content loss and style loss are calculated in different layers. The content loss is calculated as the squared error loss between the two feature representations. The style loss is calculated as the squared Frobenious norm of the difference between the Gram matrices. A gradient descent is performed on the output image to minimize the weighted sum of the content loss and style loss.

In details, the content loss of layer l can be calculated as

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

where F_{ij}^{l} is the activation of the ith filter at position j in layer l.

The style loss of layer l can be calculated as

$$E_{style} = \frac{1}{4N_l^2 M_l^2}\sum_{ij}(G_{ij}^{l} - A_{ij}^{l})^2 \tag{2}$$

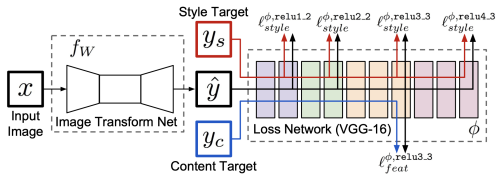

Figure 2. Architecture of fast neural style transfer model. Figure is from cs231n lectures.

where G^l_{ij} is the inner product of the feature map i and j in layer l.

Putting content loss and style loss together, the total style loss is

$$L_{style}(\tilde{a}, \tilde{x}) = \sum_{l=0}^{L} w_l E_l \tag{3}$$

where w_l is the weighting factors for layer l.

Specifically in the code we use, the pretrained model used to evaluate content and style loss is a VGG19 model. And the content layer to compute content loss is conv4, and the style layers to compute style loss are conv1, conv2, conv3, conv4 and conv5. The code implementation is built upon the tutorial https://pytorch.org/tutorials/advanced/neural_style_tutorial.html.

3.2. Fast Neural Transfer (Johnson et al. [11])

The way how the content loss and style loss are calculated is similar to the original neural style transfer paper [3]. To speed up the inference time, a feed-forward network is trained to produce the output image. Specially, the feed-forward network follows the encoder-decoder architecture, where downsampling procedure is applied to reduce the spatial extent of feature maps, and then upsampling procedure is applied to produce the final output image. Apart from the content loss and style loss, which in this paper are called perceptual loss function, this method also adds pixel loss and total variation regularization. However, for the purpose of comparison, we cast them to zero in our work. The code for my experiment is built upon https://github.com/gordicaleksa/pytorch-neural-style-transfer-johnson [6]. The pretrained model we used to evaluate content and style loss is also a VGG19 model.

3.3. CycleGAN (Zhu et al. [21])

Although also belonging to the per-style-per-model category like the two methods above, CycleGAN differentiates itself as it's a Generative Adversarial Network. The objective of CycleGAN is to learn an algorithm that can translate between two domains with unpaired images. Specifically, CycleGAN uses two generators and two discriminators.

In the Figure 3, generator G is responsible to translate images from domain X to domain Y, and the generator F is

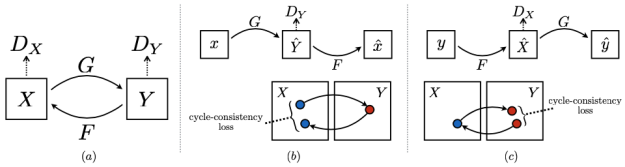

Figure 3. (a) The model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs in distinguishable from domain Y, and vice versa for D_Y and F. To further regularize the mappings, two cycle consistency losses are introduced: (b) forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$. Figure is from paper [21].

responsible to translate images from domain Y to domain X. D_X and D_Y are the corresponding discriminators for generators F and G.

The loss is composed of two types of loss. One is adversarial loss, and the other is called cycle consistency loss. In details, for the generator G, adversarial loss is what any common GAN would have, and it can be calculated as

$$\begin{aligned} L_{GAN}(G, D_Y, X, Y) = E_{y \sim p_{data}(y)}[log D_Y(y)] \\ + E_{x \sim p_{data}(x)}[log(1 - D_Y(G(x)))] \end{aligned} \tag{4}$$

The cycle consistency loss is more special. Without cycle consistency loss we can imagine that the translated image from domain X would look like domain Y, but it's not guaranteed to look like the original image from domain X. And cycle consistency loss is added to constrain the algorithm to let the generated image not diverge from the input image too much. The definition can be interpreted as when an image is converted to the another domain and back again, the result image should look like the original input image. Here is the equation to compute cycle consistency loss:

$$\begin{aligned} L_{cyc}(G, F) = E_{x \sim p_{data}(x)}[||F(G(x)) - x||_1] \\ + E_{y \sim p_{data}(y)}[||G(F(y)) - y||_1] \end{aligned} \tag{5}$$

Then the full loss equation is defined as:

$$\begin{aligned} L_(G, F, D_x, D_y) = L_{GAN}(G, D_Y, X, Y) \\ + L_{GAN}(F, D_X, Y, X) \\ + \gamma L_{cycle}(G, F) \end{aligned} \tag{6}$$

where γ is the weight to control the contribution of these two kinds of loss.

The code for our experiment is built upon https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix [9].

3.4. HED (Xie et al. [17])

Holistically-Nested Edge Detection (HED) is the method we choose to get the edge map of the content images. It's a deep learning model that leverages convolutional neural networks and deeply-supervised nets. It solves several problems of the common used Canny Edge Detector, such as it doesn't require manually finding and setting the lower bound and upper bound to the hysteresis threshold. HED achieves state-of-the-art results on the BSD500 dataset and the NYU Depth dataset, and it only takes 0.4s per image processing.

The target loss contains two types of loss – side-outputs loss and fusion loss.

Side-output loss is defined as:

$$L_{side}(W, w) = \sum_{m-1}^{M} (a_m l_{side}^{(m)})(W, w^{(m)}) \tag{7}$$

W is the collection of all standard network layer parameters, and w is the collection of the weights of each side-output layer.

Fusion loss is defined as:

$$L_{fuse}(W, w, h) = Dist(Y, Y_{fuse}) \tag{8}$$

Combining side-output loss and fusion loss together, the objective function is:

$$(W, w, h)^* = argmin(L_{side}(W, w) + L_{fuse}(W, w, h)) \tag{9}$$

The code for our experiment is built upon https://github.com/sniklaus/pytorch-hed [17] [12].

4. Dataset and Features

The dataset used in this work contains 1976 landscape photos and 1976 landscape themed traditional Chinese ink paintings. Then the dataset is partitioned 9/1 where 1778 landscapes photos and paintings are used as the training set, and 198 landscapes photos are used as the test set.

Collecting style images for this paper is actually a hard problem, as there are no organized and open source traditional Chinese ink paintings dataset. There are some other works using images queried through Google Search API, however we can not guarantee those images are real traditional Chinese inking paintings without a big load of manual work, meaning they are not fake paintings, computer synthesized images, or images drawn by modern artists that don't use the traditional ink techniques. Thus we require the dataset is well sourced. Within those constraints, we retrieved 1976 landscape themed Chinese ink paintings from the dataset collected by paper [18]. The author collected dataset from four museums, which we find is credible. The landscape photos are collected from the ChipPhi dataset that

Figure 4. Example landscape photos as content images

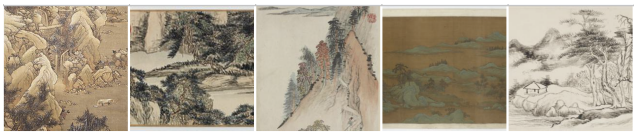

Figure 5. Example landscape themed traditional Chinese ink paintings as style images

ChipGAN paper [7] provides, where we combine trainA and testA photos from their scenery folder.

The 1976 landscape photos come from different sizes so we center crop it to 256x256 with potential scaling if the height/width is smaller than 256. The original 1976 Chinese ink paintings were already preprocessed by the author and have 512x512 resolutions. We downsampled it to 256x256 to match the content images and also to reduce computation expense.

Figure 4 and 5 show some example images after preprocessing.

5. Experiments

5.1. Training Details

We trained 6 different models in total:

- Original NST: We set the style weight to 5e4 and the content weight to 1. Each pass takes 500 iterations. The other hyperparameters are the default ones used in the original paper.
- Original NST + HED: The hyperparameters remain the same with above. Instead of feeding the content image, we feed in the edge map of the content image retrieved through HED.
- Fast NST: We set the style weight to 5e4 and content weight to 1, same as the original NST for the fairness of comparison. We feed in 1778 content images for training. Batch size is 4, and the model is trained for 2 epochs. The other hyperparameters are the default ones used in the original paper.
- Fast NST + HED: The hyperparameters are the same with above. Instead of feeding a set of 1778 content images, we feed a set of the edge map of the 1778 content images retrieved through HED.

Figure 6. The one style image that Original/Fast NST(+) HED models take.

- CycleGAN: We feed in a set of 1778 content images and a set of unpaired 1778 style images. Batch size is 1 and the model is trained for 85 epochs with learning rate 0.0002 without learning rate decay. The other hyperparameters are the default ones used in the original paper.

- CycleGAN + HED: The hyperparameters are the same with above.Instead of feeding a set of 1778 content images, we feed in a set of the edge map of the 1778 content images with the unpaired 1778 style images.

Original NST, Original NST + HED, Fast NST, Fast NST + HED only takes one style image, and that one image is chosen from our training set. Figure 6 shows how it looks like.

5.2. Results

We compare different approaches against each other in both qualitative and quantitative methods. We show the generated output images of 6 images from our test set through those 6 different methods and analyse their visual effects. We also show results of human study by 10 people. Then we compare and analyze the nearest neighbor test results, inference speed, training time, training loss trend for each method.

Figure 7. Example edge detection maps of 6 test content images.

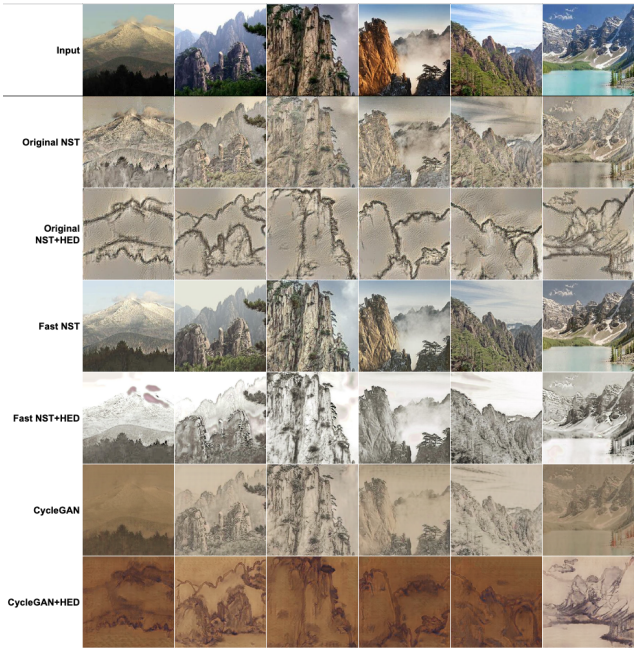

Figure 8. 6x6 output images of 6 test content images among 6 different methods.

5.2.1 Visual Quality Comparison

Figure 7 shows the edge detection map of 6 content images from our test set. Figure 8 shows the result of generated output images of 6 content images from our test set, compared between 6 methods.

We can see that for Fast NST, while the generated images are whiter, the images still keep some extent of their original colors and backgrounds. For example, the skies are still blue, although in a traditional Chinese ink paintings the skies are most likely left blank. This actually might be due to we didn't use a best style loss and content loss weight ratio, so the style image doesn't affect the generated image much.

For Original NST + HED, while all of the background color is washed off, the important details are also dropped. For example, only the contour is depicted, and the stone or the tree texture is not presented.

We also notice that for Fast NST + HED, some of the generated image has pinky patches. And for the last input in

Methods	Average Score
Original NST	3.9
Original NST+HED	5.4
Fast NST	5.22
Fast NST+HED	2.7
CycleGAN	1.4
CycleGAN+HED	2.3

Table 1. Human study conducted by 10 participants.

Figure 8, CycleGAN generates a picture where the trees are in red color instead of green. This mysterious appearance should also be further investigated.

5.2.2 Human Study

We asked 10 people to give rank 1 (the best) to 6 (the worst) score for the 6 set of generated images, in terms of how close they feel the result images are to traditional Chinese ink paintings. 7 people think CycleGAN has the best result. 2 people think CycleGAN + HED has the best result. 1 person think Fast NST + HED has the best result. 5 people think Original NST + HED has the worst result. 5 people think Fast NST has the worst result. Table 1 shows the average score for each method, based on the scores given by 10 people on the 6 test images.

5.2.3 Nearest Neighbor Test

Figure 9 shows the five nearest neighbors in the training dataset for each generated image. The first column shows the generated images through 6 methods with the same input content image. The other five columns show their corresponding 5 nearest neighbors in the training dataset. The nearest neighbor is based on the L2 distance of pixels. We can see that all the models scale well, such that they are not just memorizing the images in the training set.

5.2.4 Stylization Speed

Table 2 shows the average time to do one image generation with resolution 256x256 for different methods in inference time, running with GPU that the default cloud machine Google Colab provides.

The fastest method in inference time is CycleGAN which takes only 0.008s. The slowest method in inference time is Original NST + HED, which takes 6.336s. This is reasonable because it's doing forward and back pass online. The overhead of HED processing is about 0.172s, which is comparatively small for Original NST methods.

Please note, our number might be different from the numbers on other papers like [10], mainly because of two

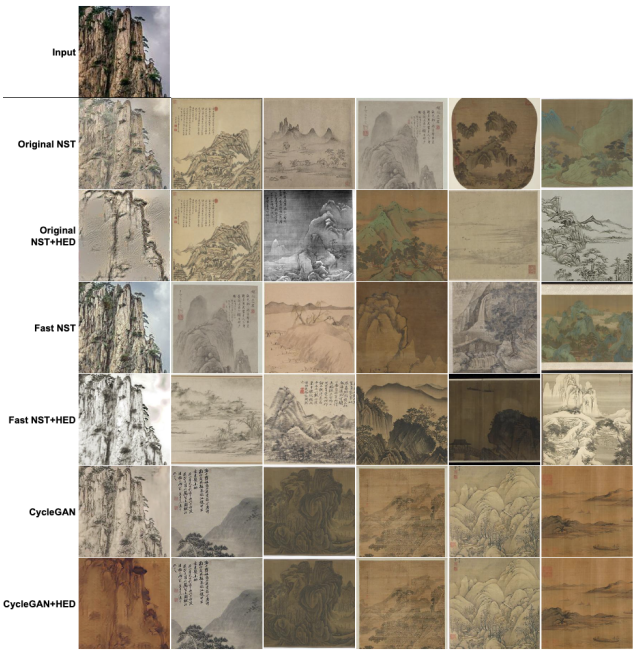

Figure 9. Nearest Neighbor Test showing the five nearest neighbors in the training dataset for each generated images. The first column shows the generated images through 6 methods with the same input content image. The other five columns show their corresponding 5 nearest neighbors in the training dataset.

Methods	Time (s)
Original NST	6.164
Original NST+HED	6.336
Fast NST	0.312
Fast NST+HED	0.484
CycleGAN	0.008
CycleGAN+HED	0.180

Table 2. Average inference time.

reasons 1) For our convenience, the inference time is collected when running test images on Google Colab instead of the AWS g4dn.xlarge machine where we train our models and which uses the latest generation NVIDIA T4 GPUs. 2) For some models, we are not using the implementation code provided by the original paper.

5.2.5 Training Time

We are using a pretrained HED model for edge detection, so we don't account for the time for it. Training Fast NST with original content images takes about 4 hours. Training Fast NST with edge maps takes about the same time. Training CycleGAN with original content images for 85 epochs takes 23.5 hours. Training CycleGAN with edge maps for

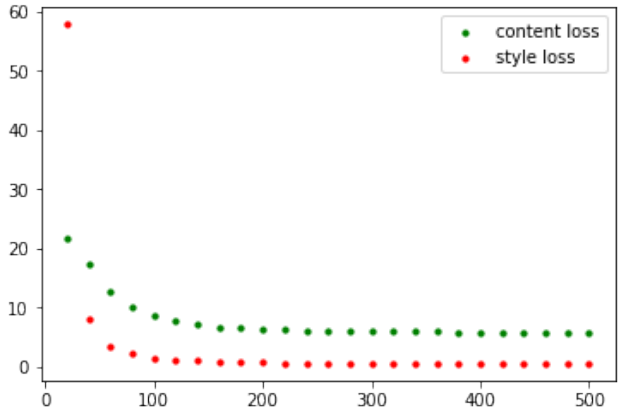

Figure 10. Training loss for original NST.

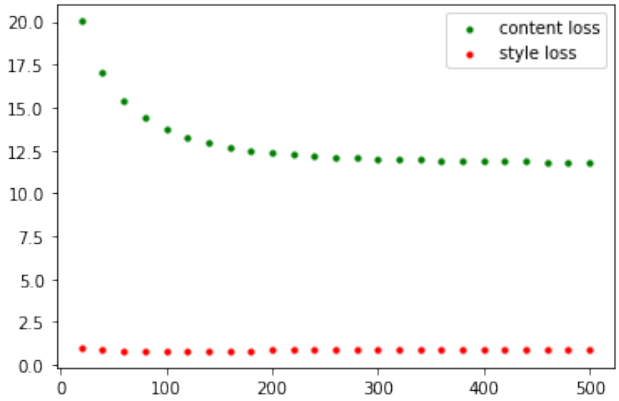

Figure 11. Training loss for original NST + HED.

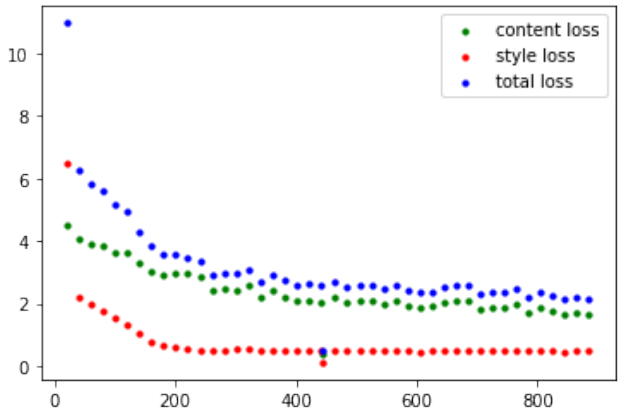

Figure 12. Training loss for Fast NST.

85 epochs takes about the same time.

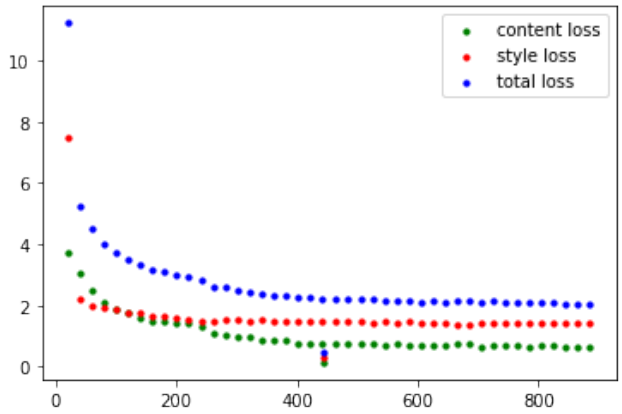

Figure 13. Training loss for original NST + HED.

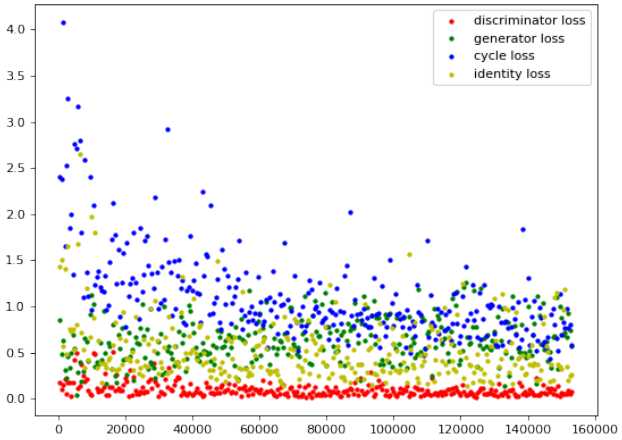

Figure 14. Training loss for CycleGAN for content image-¿style direction. We don't show training loss plot for the opposite direction.

5.2.6 Training Loss Comparison

Figure 10 - 15 show the training loss for different algorithms. Please note, for Orignal/Fast NST (+) HED, the loss is plotted every 20 iterations. Each batch has 4 samples and the model is trained for 2 epochs. For CycleGAN (+) HED, the loss is plotted every 500 iterations. Each batch has only 1 sample and the model is trained for 85 epochs. So the flavor of the training loss images might look very different.

Looking at the training loss images, we can see that original NST + HED is not trained very well. The style loss was almost not changed. It can explain why this method was evaluated one of the worst methods by the human study participants. For the other training process, all the loss is more or less converged during the training, although for the CycleGAN + HED, the trend might not be converged very well as we can see it still has a lot of up-and-down scatter points at the end of the training process. We think it might help if

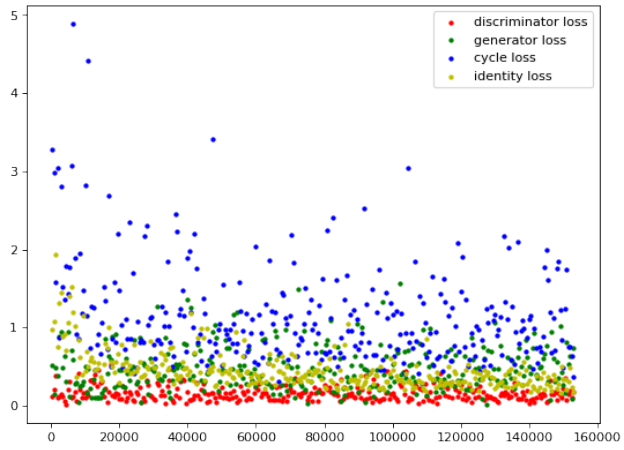

Figure 15. Training loss for CycleGAN+HED for content image-¿style image direction. We don't show training loss plot for the opposite direction.

we can train with learning rate decays and more epochs in future work.

6. Conclusion and Future Work

Among all the methods, the one that achieves the best visual effect to transfer traditional Chinese ink painting style is CycleGAN, and the one that achieves the worst visual effect is Original NST + HED. The fast method in terms of inference time is CycleGAN (+) HED, the slowest method in terms of inference time is Original NST (+) HED. CycleGAN (+) HED requires the longest training time, while online method like Original NST (+) HED requires the least training time.

However, we want to disclaim that the model is trained under many constraints and the final result might not be the truth. For example, we use many default hyperparameters of the original papers, but actually we should spend more time on fine tuning on the hyperparameters to achieve best results for each method for the fairness of the comparison. For example, the style image and the content image weights used for Original/Fast NST (+) HED methods, which is like the most important hyperparameters for the model, is chosen based on several trials of us, and we didn't exhaustively searched the space to find out the best value. Also, we early stop on the CycleGAN (+) HED training on its 85 epochs, and we should train for longer with learning rate decay if time and computation resources permit.

What's more, our motivation to use edge maps instead of the original content images is coming from that when we saw the results of Fast NST, the color of the original content image is well preserved which is something we don't want in a ink painting. So we thought about using an edge detection map to 'wash off' the color of the content painting. It did improve the result for Fast NST, as it shows in Figure 8. However it's not always working, as it shows in Figure 8 for Original NST+HED. We need to probe into the reasons behind either the idea of adding HED itself is not right, or it's because the wrong hyperparameters for training.

In the end, as we collect a lot of logs and evidence during training, we can utilize them for even better analysis. For example, the opposite direction's generator and discriminator in CycleGAN (+) HED can also give useful information to probe into the convergence of the model and help spot the issues.

7. Contributions & Acknowledgements

The public code that I make use of or take references on are

https://pytorch.org/tutorials/advanced/neural_style_tutorial.html

https://github.com/gordicaleksa/pytorch-neural-style-transfer-johnson

https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix

https://github.com/sniklaus/pytorch-hed

Except those, all the other code are written by myself, including data preprocessing preparation, evaluation, visualization, connection code for HED with the other models and to make things work on both Google Colab and AWS EC2, and also a small amount of modifications to the public code.

References

[1] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A learned representation for artistic style. *CoRR*, abs/1610.07629, 2016. 2

[2] Leon A. Gatys, Matthias Bethge, Aaron Hertzmann, and Eli Shechtman. Preserving color in neural artistic style transfer. *CoRR*, abs/1606.05897, 2016. 2

[3] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015. 1, 2, 3

[4] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Texture synthesis and the controlled generation of natural stimuli using convolutional neural networks. *CoRR*, abs/1505.07376, 2015. 2

[5] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423, 2016. 2

[6] Aleksa Gordić. pytorch-nst-feedforward. https://github.com/gordicaleksa/pytorch-nst-feedforward, 2020. 3

[7] Bin He, Feng Gao, Daiqian Ma, Boxin Shi, and Ling-Yu Duan. Chipgan: A generative adversarial network for chinese ink wash painting style transfer. In *Proceedings of*

the 26th ACM International Conference on Multimedia, MM '18, page 1172–1180, New York, NY, USA, 2018. Association for Computing Machinery. 2, 4

[8] Xun Huang and Serge J. Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. *CoRR*, abs/1703.06868, 2017. 2

[9] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, 2017. 3

[10] Yongcheng Jing, Yezhou Yang, Zunlei Feng, Jingwen Ye, and Mingli Song. Neural style transfer: A review. *CoRR*, abs/1705.04058, 2017. 2, 6

[11] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. *CoRR*, abs/1603.08155, 2016. 1, 2, 3

[12] Simon Niklaus. A reimplementation of HED using PyTorch. https://github.com/sniklaus/pytorch-hed, 2018. 4

[13] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2016. 2

[14] Manuel Ruder, Alexey Dosovitskiy, and Thomas Brox. Artistic style transfer for videos. *CoRR*, abs/1604.08610, 2016. 2

[15] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor S. Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. *CoRR*, abs/1603.03417, 2016. 2

[16] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *CoRR*, abs/1607.08022, 2016. 2

[17] Saining Xie and Zhuowen Tu. Holistically-nested edge detection. In *IEEE International Conference on Computer Vision*, 2015. 1, 2, 4

[18] Alice Xue. End-to-end chinese landscape painting creation using generative adversarial networks. *CoRR*, abs/2011.05552, 2020. 2, 4

[19] Chengyu Zheng and Yuan Zhang. Two-stage color ink painting style transfer via convolution neural network. In *2018 15th International Symposium on Pervasive Systems, Algorithms and Networks (I-SPAN)*, pages 193–200, 2018. 2

[20] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, abs/1703.10593, 2017. 2

[21] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017. 1, 3