Automatic Comic Style Transformation of Image

Xiaoan Sun, 79879938, MEng *Student,* sunxiaoan2019@163.com, *UBC*

Abstract

Comic is a popular method of art expression. The creation of a comic requires high talent and is time-consuming so that developing a simple method for comic creation can benefit non-professional creators. This paper introduces a model that realizes the automatic comic style transformation of images. This model is based on the Neural Style Transfer algorithm. I apply the algorithm and improve it by realizing noise reduction and multiple style combination through adjusting the loss function. Additionally, I realize quality improvement, edges enhancement and color conversion of the generated images through changing the initial white noise image to a content-image-related image, using the Canny edge detector and adjusting the generated images in YCbCr color space. Moreover, I tune the hyper parameters of the models so that it can generate images in excellent comic style. Finally, I conduct two experiments and a survey to validate the effectiveness of my model. It proves that it can generate equally excellent or better comic style images than several existing models such as Deep Dream generator and Ostagram.

,

***Keywords*—Canny edge detector, comic art style, convolutional neural network, deep learning, Neural Style Transfer, total variation regularization, visual computing**

1. INTRODUCTION

COMIC is a popular method of values transmission, artistic expression and entertainment. For modern comic, it has more than eighty-year history since the publication of the first modern comic book "Famous Funnies". The popularity of comic experiences a remarkable increment due to the development of the film and television industry, especially the American comic. Unfortunately, comic creation requires high talent and it is a very time-consuming process. Even imitating existing comic is a difficulty to a majority of people. Therefore, creating an easy method for comic creation may bring benefits to the artistic creation of amateurs

To offer a convenient and simple method for comic creation, I develop a program that can realize the automatic comic style transformation of images. The program can reconstruct an image based on the images offered by the user. The user can decide the content and the style of the generated image separately. This program is based on the theory proposed in [1], whereas I adjust the algorithm to expand its flexibility and improve the quality of the generated image. Besides, I add two extra optional preprocessing steps and one optional post-processing step to improve the quality of the generated image as a comic style image. I also tune the hyper parameters in detail to seek an appropriate configuration for generating a comic style image.

The outline of the paper is as follows. In the next section, I will introduce the background and related work of style transfer. Section 3 introduces the materials and technologies used in developing the model. In Section 4, I will discuss the process of developing the model and the adjustments and improvements I realized for the model. In Section 5, I will share how I design the experiments to test the effectiveness of my model. In Section 6, I will present the results of all the adjustments, improvements and experiments. I will end the paper with a brief concluding section.

2. BACKGROUND AND RELATED WORK

Image style transfer is not a brand new research area. In 2000, paper as [2] had similar research on extracting and recreating texture from images. In [3], Amir Semmo *et al.* tried to transfer images into an oil paint look. Paper [4] tried to transfer the style of headshot portraits. There are some other similar researches on image style transfer, whereas most of them have mediocre performance and require manual modeling, which is time-consuming and difficult to operate.

In 2012, Geoffrey Hinton and his group attended the ImageNet Large Scale Visual Recognition Challenge with their AlexNet model [5] based on Convolutional Neural Network (CNN). The excellent performances of their model aroused the attention of researchers on the potentiality of deep learning and CNN in image recognition. In the same period, the rapid development of Graphic Processing Unit (GPU) offered increasing computing power to Deep Neural Network. Through this opportunity, some famous CNN models with excellent performance on visual computing were developed such as VGG-Net [6], GoogLeNet [7] and Resnet[8].

In 2015, Leon, A Gatys *et al.* discovered the potentiality of CNN in image style transfer [1]. They used CNN to separate the content and style of images and demonstrate an algorithm that can recreate an image with its content in the style of other images. This algorithm is known as "Neural Style Transfer" (NST), which is one of the core technologies to realize the automatic comic style transformation of images in this paper.

3. MATERIAL AND RELATED TECHNOLOGY

This section will introduce the technologies and dataset I used in developing the model

Visual Geometry Group Network. Visual Geometry Group Network (VGG Network)[6] can be considered as a "deeper version of AlexNet". It consists of 5 convolutional layers, 3

fully connected layers and 1 softmax layer. Each layer is connected by a max-pooling layer. Compared with AlexNet, VGG Network reduces the size of the convolution kernel but increases the number of convolutional layers. This structure allows the VGG Network to include less number of parameters but have a better performance in computer vision. Since VGG Network has a similar ability in visual object recognition benchmark with human performance, it is suitable to use VGG for extracting the feature maps from the image.

Total variation regularization. Rudin *et al.* proposed the Total variation regularization in [9]. The core idea is reducing noise existing in signals by reducing the total variation since signals with excessive spurious details have high total variation. Therefore, reducing the total variation of the signal can remove unwanted details while preserving important details. The noise that exists in an image can also be eliminated through this method. More precisely, by reducing the absolute gradient between one pixel and its neighboring pixels, it is possible to reduce the high gradients caused by noise existing in the image.

Canny edge detector. Canny edge detector based on the algorithm proposed by John F. Canny in [10]. The basic idea is similar to the Sobel operator. It will detect sharp changes in the gradient image. What different is the Canny edge detector will decrease the edges to 1 pixel wide by finding the local maximum in perpendicular direction. Besides, it uses hysteresis thresholding to retain the completeness of detected edges. More specifically, it has two thresholds T_1 and T_2 (T_1>T_2). Any edge with a value that larger than T_1 will be preserved. If an edge has a value between $[T_2, T_1]$, it will be preserved only if the edge connects with any preserved edge. This idea enables the Canny edge to detect a more complete edge than the Sobel operator can do.

ImageNet dataset. ImageNet is a dataset with 14 million hand-annotated images and at least 1 million of these images have bounding boxes provided. These images belong to more than 20,000 categories [11], [12]. Its huge amount of data and high quality of images make it always a suitable choice for image recognition model training.

Google Colaboratory. Google Colaboratory is a research tool open by Google. It is a Jupyter notebook IDE that offers free but high-quality GPU (Tesla K80) for model training. This IDE supports programming in Python and includes several useful libraries, which makes it a suitable tool for developing this model.

Image acquisition. All of the images used in the experiment are randomly chosen from the Internet. The only requirement is the image should have resolution larger than 600×800 pixels.

4. METHODS

This section will introduce the method used to develop the model. I will introduce the principle of NST. Then I will discuss the adjustments I make on the algorithm. At the end of the section, I will discuss the preprocessing and post-processing methods I used to improve the generated image and hyper parameter tuning of the model.

4.1 Neural Style transfer

The concept of NST is purposed by Leon, A Gatys *et al.* in [1]. The purpose is combining the style of an image with the content of another image to realize style transfer. The style here refers to the texture, color and visual mode of an image. Fig. 1 shows the general process of NST. The input is a white noise image. Through continuously modifying the pixel values of the input image, the loss function that represents the differences between the input image and two target images (one image provides the content and the other image provide the style) is optimized. The final generated image is the output result.

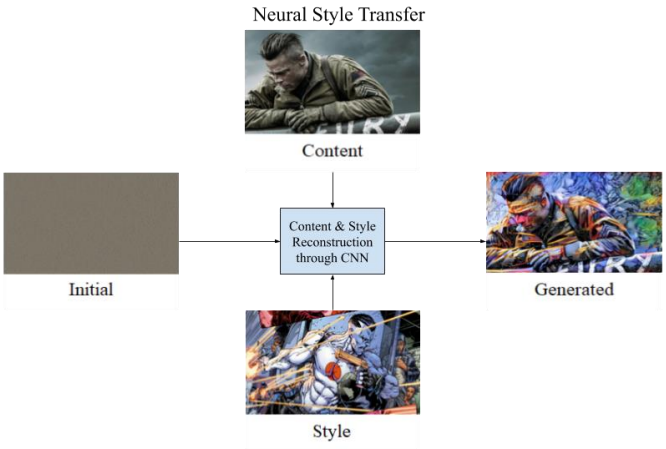

Figure 1: Process of Neural Style Transfer

Therefore, the most important part of NST is the loss function because it defines the purpose of optimization. Through the definition in [1], the loss function is

$$L_{total} = \alpha L_{content} + \beta L_{style} \quad (1)$$

where L_{total} is the total loss function. $L_{content}$ is the loss of content, which means the difference between the generated image and the content image. L_{style} is the loss of style, which refers to the difference between the generated image and the style image. The α and β are the weighting factors for content and style during reconstruction, which can also function as normalization to accelerate the training process.

Feature Map Extraction. To calculate the loss of style and the loss of content, we need to find appropriate expressions to represent the style and the content of an image. For this purpose, extracting features maps from the image is necessary. To a CNN model, each filter on a convolutional layer will return a high activation value if it detects a specific image patch in the image. When the filter traverses the entire image, it generates a map reflecting the activation value generated by the filter at each location in the image. This map is called the feature map. Fig.2 shows an example of feature map extraction.

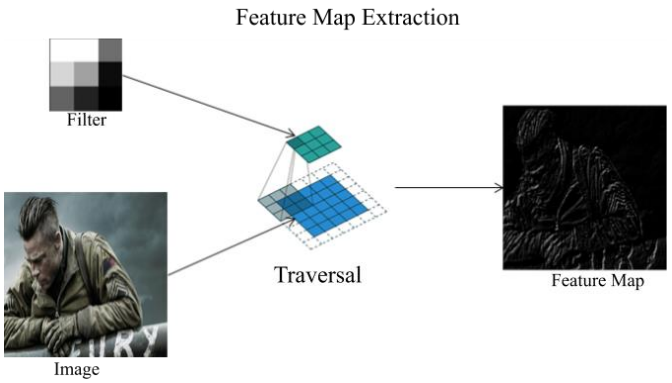

Figure 2: Feature Map Extraction

Content loss. Through inputting an image into a pre-trained model such as VGG-19 trained with the ImageNet dataset, it is possible to obtain the feature maps generated on a hidden layer. These features maps can together represent the content of an image to some degree. Therefore, the content loss is the squared Euclidean distance between activation values at the same location of each feature map of the generated image and the content image

$$L_{content}(C,G) = \frac{1}{2}\sum_{i,j}(a_{i,j}^{[l](C)} - a_{i,j}^{[l](G)})^2 \qquad (2)$$

where $a_{i,j}^{[l](C)}$ is the j^{th} activation value on the i^{th} feature map of the layer l^{th}. (C) represents the content image and (G) represent the generated image.

Style loss. Feature map cannot represent the style feature. In [1], the style feature is defined as the correlation between different filters, which can be evaluated through calculating the inner product between vectored feature maps k and k'

$$G_{k,k'}^{[l]} = \sum_j a_{j,k}^{[l]} a_{j,k'}^{[l]} \qquad (3)$$

where $a_{j,k}^{[l]}$ is thej^{th} activation value on the k^{th} feature map of layer l^{th}. If the correlation value is high, it means a texture that can be detected by filter k and k' always occurs in the image. In this case, it is possible to generate a Gram matrix $G^{[l]} \epsilon R^{N_{[l]} \times N_{[l]}}$ that represents the style feature of an image, where $N_{[l]}$ is the number of filters of the layer l^{th}. The style loss is the mean-squared distance between the Gram matrix of the style image and the Gram matrix of the generated image. For each layer, the content loss should be

$$L_{style}^{[l]}(S,G) = \frac{1}{4N_{[l]}^2 M_{[l]}^2}\sum_{k,k'}(G_{k,k'}^{[l](S)} - G_{k,k'}^{[l](G)})^2 \qquad (4)$$

where $M_{[l]}$ is the size of feature map. (S) represents the style image. Therefore, the total style loss function is

$$L_{style}(S,G) = \sum_l w^{[l]} L_{style}^{[l]}(S,G) \qquad (5)$$

where $w^{[l]}$ is the weighting factor of each layer.

The content loss and style loss constitute the total loss function of NST. Through using the gradient descent algorithm to reduce the total loss, it is possible to generate an image with similar style to the style image and similar content to the content image.

4.2 Model improvement

The image generated through NST can realize initial comic style transformation. Nevertheless, the quality of generated images is unsatisfactory and the visual effect is not as expected. Moreover, there is obvious noise existing in the generated images and it takes a long time to generate a relatively satisfactory image. To solve these problems, I adjust the total loss function of my model, add two preprocessing steps and one post-processing step, and tune the hyper parameters.

Total loss function adjustment. Since the loss function defines the purpose of optimization, through adjusting the total loss function, it is possible to improve the generated images. I tried adding three extra parts to the total loss function.

The first part is a total variation loss (TV loss) to reduce the noise generated in the image

$$L_{TV} = \sum_{h,w} |I_{h+1,w} - I_{h,w}| + |I_{h,w+1} - I_{h,w}| \qquad (6)$$

where $I_{h,w}$ is the pixel value at the location (h,w) on the generated image.

The second part is an extra style loss component to mix the style of two images so that it may generate an image with a better visual effect.

$$L_{style}(S_2,G) = \sum_l w^{[l]} L_{style}^{[l]}(S_2,G) \qquad (7)$$

where (S_2) represents the second style image.

The third part is adding an edge loss component to the total loss function to emphasize the edges of objects in the image and make them clearer in the generated image. This adjustment is realized through using the Canny edge detector to extract the gray-scale edge map from the content image and the generated image and calculate their squared Euclidean distance

$$L_{edge}(C,G) = \sum_j (I_j^{(C)} - I_j^{(G)})^2 \qquad (8)$$

where $I_j^{(C)}$ is the j^{th} pixel value on gray-scale edge map of (C).

All of these three extra loss components have a weighting factor. The first and second adjustments turn out to be effective, whereas the third adjustment is not successful. I will discuss this part in Section 6 in detail.

Preprocessing and post-processing. We can preprocess the initial image and the content image and post-process the generated image to improve the quality of the generated images.

For the initial image, I use a content-image-related image instead of the white noise image. This step is realized through combining the white noise image with the content image

$$Image_{initial} = \gamma Image_{white} + (1-\gamma) Image_{content} \quad (9)$$

where γ controls the proportion of white noise in the initial image.

For the content image, I enhance its edges before inputting it to the model. This step is used for making the edges clearer in the generated image since edge loss control is not genuinely effective. Specifically, I extract the gray-scale edge map from the content image, dilate the edges through morphological operation and cover them on the content image to generate a new content image with enhanced edges.

For the generated image, I convert it to the color of the content image. This step is realized by adjusting them in the YCbCr color space. I combine the luma component of the generated image with the blue-difference and red-difference chroma components of the content image. As a result, the color of the generated image will have similar color to the content image.

These preprocessing and post-processing steps turn out to be effective. The performances of these steps will be discussed in Section 6

Hyper parameters tuning. Hyper parameters can affect the quality of generated images and the training process to a great extent. For this reason, it is necessary to tune the hyper parameters. For my model, I tune the layers used for extracting the style features and content features, the weighting ratio of style and content, the iteration of training and the learning rate. I will discuss the details in Section 6.

5. Experiment Design

I conducted two experiments for testing my model.

Self-comparison. The first experiment is self-comparison, which is used to compare the effect of adjustments on the total loss function, preprocessing and post-processing methods and hyper parameter tuning. I used the control variates method to observe the effects, setting a default parameter configuration obtained through previous rough experiments and adjusting one hyper parameter each time.

It is difficult to find authoritative standards for qualitative estimation or principles that define how comic style should be. Therefore, I summarized five principles from my experience and suggestions in [13] and [14] to evaluate the comic style of an image:

1) The image should be realistic.
2) The edges should be clear.
3) The color should have a relatively high contrast ratio.
4) The image should include enough details.

These principles are reasonable although they are relatively subjective and not applicable to all comics.

Comparison with other NST models. The second experiment is a comparison between my model and the other four existing models. The first and the second models are the official tutorial model of NST in TensorFlow[15] and PyTorch[16]. As the official tutorial model, they provide relatively reliable hyper parameters and models for realizing NST so that they could serve as a suitable reference. I adjusted these two models so that they have input images with the same size and training iteration number as my model. Both of these two models run on Google Colaboratory to ensure they have an equal hardware condition. The third model is the Deep Dream Generator [17] and the last model is the Ostagram[18]. These two models are famous style transfer applications and they can realize a pretty satisfactory style transfer. Since it is hard to find the source codes of these two models, I directly used them and did my best to tune the provided parameters. Although this approach will affect fairness, it can serve as a reference to judge the quality of my model to a certain extent.

To evaluate the models, I calculated the average structure similarity of each model for qualitative evaluation and the average training time for quantitative evaluation. Nevertheless, these data can offer limited information on the performance of each model. Therefore, I conducted a survey through questionnaires. I prepared different examples of comic style transfer using these five models. Survey answers are disturbed randomly to ensure fairness. The quality measures are the ability of style transfer and the degree of 'comic style'. I also collected opinions from participants on images generated through my model at the end of the survey.

I will discuss the results in the following section.

6. Result

In this section, I will discuss the results of self-comparison and comparison with other NST models.

6.1 Self-comparison

1) Total loss function adjustments

Total variation loss. In Fig. 3, we can see that adding a total variation loss can eliminate the noise generated in the image and enhance the contrast ratio to some degree if given an appropriate weighting. Through the experiment, the ratio $\frac{\text{Weighting of TV loss}}{\text{Weighting of } content\ loss}$ should be controlled within [0.1, 1], which could eliminate the noise and avoid the image getting blurry or losing details simultaneously.

Extra style loss. Although the idea of mixing two style images works, the effect of an extra style loss is hard to conclude. If the style images are close in color or texture, the effect will not be obvious. If the style images have obvious differences, this adjustment can generate an eye-catching visual effect, whereas it may cause texture division in some areas. Therefore the use of extra style loss should depend on the situation. Fig. 4 shows a sample.

Generated image with TV loss (Letf) vs Generated image without TV loss (Right)

Figure 3: Comparison between images with/without TV loss. The noise of marked areas has been obviously reduced.

Generated image with extra style loss

Figure 4: Generated image with two style images and one content image.

Edge loss. Theoretically, adding an edge loss component can reduce the edge loss in the generated image. In fact, the effect is negligible and it leads to increment in running time. Therefore, I eventually removed this adjustment.

2) Preprocessing and post-processing

Content-image-related initial image. Through the experiment, a content-image-related initial image can generate an image with better visual effect in the short term and a high proportion of content image (i.e. low γ) can effectively avoid the generated image become excessively abstract even with high style weighting as shown in Fig.5.

Edges enhancement of the content image. In Fig.6, it is obvious that the generated image with edges enhancement has clearer edges. Besides, this preprocessing can play retain part of details to some degree. However, we can observe that not all edges have been enhanced. Moreover, since the enhancement will be applied to all the edges detected, it will make the edges that are originally redundant more obvious. In general, this preprocessing is effective.

Generated image from white noise image (Left) vs generated image from content-image-related image (Right)

Figure 5: Generated image with white noise image/content-image related image. The content-style weighting ratio is 10^6. As we can observe, the image generated from content-image related image (Right) is relatively robust to high style weighting.

Generated image with edges enhancement (Left) vs Generated image without edges enhancement (Right)

Figure 6: Generated image with/without edges enhancement. The blue marks show redundant edges and the red marks show details and edges improvement

Color conversion. Color conversion can make the generated image close to the content image so that it can make the generated image more realistic. Since the luma component of the generated image has been retained, this post-processing will not cause an obvious effect on the contrast ratio. However, this method will restrict the visual effect benefiting from the style image. Therefore the use of color conversion should depend on the situation. Fig. 9 displays some samples.

3) Hyper parameter tuning

Learning rate and iteration. A higher learning rate can accelerate the training process and receive a better generated image in the short term. Nevertheless, an excessively high learning rate will cause the generated image to become excessively abstract and cause detail loss in the generated image. This result does not qualify the principles I set so that it should be avoided. Fig. 7 displays the effect.

As a result, I set the learning rate to 5 to balance efficiency and quality. When the learning rate equals 5, the model can

generate a relatively satisfactory result within 4000 rounds. Although the image keeps improving and the total loss is still decreasing after 4000 rounds, the effect is not proportional to the time consumption.

Figure 7: The comparison among images with different learning rates. The marked area shows the detail loss occurs in images with high learning rate.

Feature extraction layer and content-style weighting ratio. First, I reproduced the experiment of [1], which explores the effect of different layer for extracting content features and the effect of different content-style weighting ratio on the generated images, and obtain similar conclusions. I display some samples in Fig. 8. In summary: 1. Lower content-style weighting ratio will lead to a more abstract generated image. 2. Extracting content features from the shallower convolutional layer will lead to a higher pixel-level similarity between the generated image and the content image, which will retain more details but have a poorer visual effect. Through testing, I eventually chose "block3_conv3" for extracting the content feature map and set the content-style weighting ratio to 10^3 since they can retain enough details of the face and generate a relatively eye-catching visual effect simultaneously.

Then, I compared the texture generated from layers of different depths and find that textures generated in the middle layer can obtain a relatively high contrast ratio and visual effect. Therefore I enhanced their weightings when calculating the total style loss.

Figure 8: Three samples of the generated image with different content feature layer and content-style weighting ratio

These are all of the improvements I realized for my model and I demonstrate a few cases in Fig. 9.

Figure 9: Two samples of eventually generated from the model. Images on the left side are the original generated image. Images on the right side are generated images with the color conversion. The style image and content image are displayed at the corner

6.2 Comparison with other models

1) Qualitative evaluation

I calculated the average SSIM index compared with corresponding content images for each model. Result values are presented in Table 1. In terms of content comparison, my model has similar SSIM value to Ostagram, whereas TensorFlow tutorial model and Deep Dream Generator has higher SSIM values. This result suggests that my model still has the space to become more realistic.

Besides, as shown in Fig. 10, compared with other models, we can observe that my model has a higher contrast ratio and clearer edges. Nevertheless, the problem is images generated through my model are prone to include unnecessary edges, which is especially obvious in group 3. Moreover, the face details are relatively blurry in groups 2 and 5 compared with the TensorFlow tutorial model and Deep Dream Generator.

Table 1: Content SSIM of different model

Model	SSIM content
My model	0.36
TensorFlow tutorial model	0.422
PyTorch tutorial model	0.18
Ostagram	0.362
Deep Dream Generator	0.413

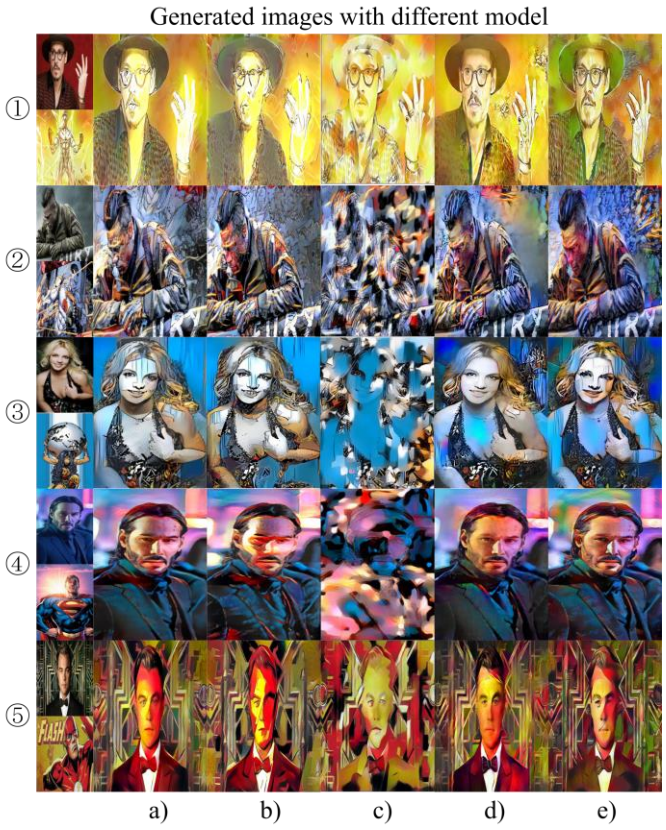

Figure 10: Generated images with different models. Columns: a) Deep Dream, b) Ostagram, c) Pytorch tutorial model, d) TensorFlow tutorial model, e) My model. The content images and the style images are shown on the left side.

2) Quantitative evaluation

For quantitative evaluation, I roughly recorded the average time to generate the images displayed in Fig. 10. Result values are presented in Table 2. Since my model, the TensorFlow tutorial model and the Pytorch tutorial model runs on Google Colaboratory and Deep Dream Generator and Ostagram are web application, the results may vary due to network quality. For Deep Dream Generator and Ostagram, I only recorded conservative time estimation since it is difficult to calculate precise time due to network fluctuation. Table 2 shows the results. Compared with other models, my model takes more time for generating a satisfactory image. It requires at least twice as long as the running time of TensorFlow tutorial model and the Pytorch tutorial model.

Table 2: Running time of different model

Model	Time for 600x800 pixel images
My model	735.68(4000 iteration)
TensorFlow tutorial model	352.08s (4000 iteration)
PyTorch tutorial model	298.4s (4000 iteration)
Ostagram	<180s
Deep Dream Generator	<90s

3) Survey

There were eventually 17 valid replies from participants with the age ranges between 19 to 26 years. I used different examples displayed in Fig. 10 for different questions. In half of the questions, I asked them to rank the top 3 (non-tied) images that close to comic style. In the rest questions, I asked them to rank the top 3 (non-tied) images that realize style transfer from a given image to the presented photo. I also collected their advice on images generated through my model at the end of the survey. Survey results are presented in Fig. 11.

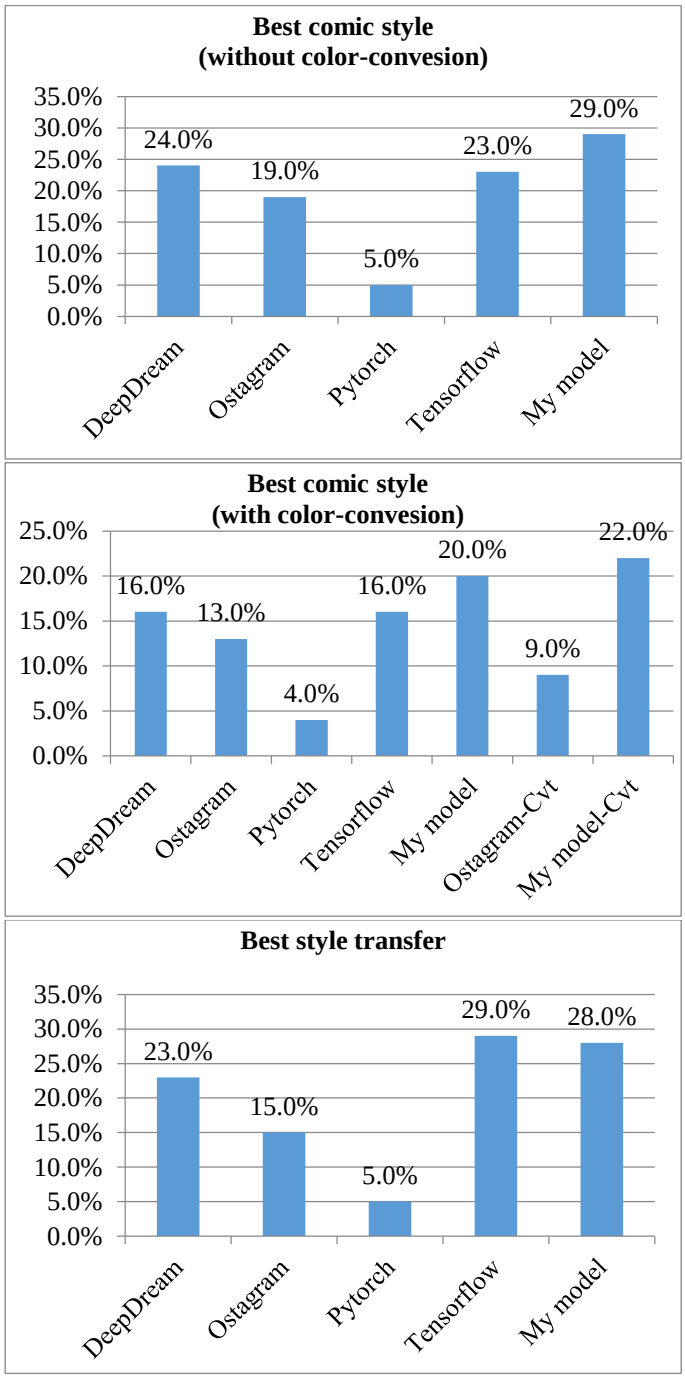

Figure 11: Survey results

As we can see my model realizes the best comic style transfer and also does well in style transfer. Besides, including color conversion into the model seem to be a wise choice. The advice collected from the user is similar to my analysis in qualitative evaluation. The advice majorly concentrated on the loss of face details and redundant edges that occurred in the images.

7. Conclusion

Comic creation is difficult for non-professional creators since it requires high talent and is a very time-consuming process. In this paper, I discuss how to develop a model that can realize automatic comic style transformation of images. The core technology is the Neural Style Transfer. Besides, I adjust the total loss function through adding second style cost component to expand the flexibility of the model and adding a total variation loss to reduce noise generated in the image. Moreover, I accelerate the training process through replacing the initial white noise image with a content-image-related image and enhance the edges in the generated image through using the Canny edge detector and morphological operation. I also set an optional color conversion function that can convert the color of the generated image to the color of the content image. Furthermore, I tune the hyper parameters of my model to make the generated image looks more comic style. In the end, I conduct two experiments and a survey to validate the model and prove that it has an excellent performance in generating comic style images.

This model can offer non-professional creators an easy method with high flexibility to create a comic style image, which may be useful for comic creation.

Nevertheless, the speed of generating images is relatively slow. Besides, it is easy to include redundant edges in the generated images, whereas the details of the face are not clear enough.

For future work, I will focus on accelerating the training process, retaining necessary details of the face and remove redundant edges from the generated image. Additionally, I will experiment with the effect of using different datasets, CNN models such as Resnet 50 and optimizers such as L-BFG

8. Acknowledgment

I want to thank Professor Rafeef Garbi for her patient guidance and useful suggestions. I also want to thank my friend Yun Hua for helping me check the grammar of this report.

References

[1] L. Gatys, A. Ecker, and M. Bethge, “A Neural Algorithm of Artistic Style,” Journal of Vision, vol. 16, no. 12, p. 326, 2016.

[2] J. Portilla and E. P. Simoncelli, “A Parametric Texture Model Based on Joint Statistics of Complex Wavelet Coefficients,” Int. J. Comput. Vis., vol. 40, no. 1, pp. 49–70, 2000.

[3] A. Semmo, D. Limberger, J. E. Kyprianidis, and J. Döllner, “Image stylization by oil paint filtering using color palettes.,” in Computational Aesthetics, pp. 149–158, 2015.

[4] Y.-C. Shih, S. Paris, C. Barnes, W. T. Freeman, and F. Durand, “Style transfer for headshot portraits.,” ACM Trans. Graph., vol. 33, no. 4, p. 148:1-148:14, 2014.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in Advances in Neural Information Processing Systems 25, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., pp. 1097--1105. 2012,

[6] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” CoRR, vol. abs/1409.1556, 2014.

[7] C. Szegedy *et al*., “Going Deeper with Convolutions,” CoRR, vol. abs/1409.4842, 2014.

[8] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[9] L. I. Rudin, S. Osher, and E. Fatemi, “Nonlinear total variation based noise removal algorithms,” Physica D: Nonlinear Phenomena, vol. 60, no. 1–4, pp. 259–268, 1992.

[10] J. F. Canny, “A Computational Approach to Edge Detection,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 8, no. 6, pp. 679–698, 1986.

[11] J. Deng, W. Dong, R. Socher, L. Li, Kai Li and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, pp. 248-255, 2009.

[12] “ImageNet,” ImageNet. [Online]. Available: http://image-net.org/. [Accessed: 16-Apr-2020].

[13] Clayton, “What Art Style Should You Pick?,” How to Draw Comics, 01-Apr-2019. [Online]. Available: https://www.howtodrawcomics.net/post/what-art-style-should-you-pick. [Accessed: 16-Apr-2020].

[14] M. Pesko and T. Trzcinski, “Neural Comic Style Transfer: Case Study,” CoRR, vol. abs/1809.01726, 2018.

[15] “Neural style transfer: TensorFlow Core,” TensorFlow. [Source Code]. Available: https://www.tensorflow.org/tutorials/generative/style_transfer. [Accessed: 16-Apr-2020].

[16] A. Jacq, “Neural Transfer Using PyTorch,” Neural Transfer Using PyTorch - PyTorch Tutorials 1.4.0 documentation. [Online]. Available: https://pytorch.org/tutorials/advanced/neural_style_tutorial.html. [Accessed: 16-Apr-2020].

[17] “Human AI,” Deep Dream Generator. [Online]. Available: https://deepdreamgenerator.com/. [Accessed: 16-Apr-2020].

[18] Ostagram. [Online]. Available: https://www.ostagram.me/static_pages/lenta?last_days=1000&locale=en. [Accessed: 16-Apr-2020].

[19] R. Yuan, “Neural Style Transfer: Creating Art with Deep Learning using tf.keras and eager execution,” The TensorFlow Blog. [Source Code]. Available: https://blog.tensorflow.org/2018/08/neural-style-transfer-creating-art-with-deep-learning.html. [Accessed: 16-Apr-2020].