Neural Style Transfer on Images

Huize Huang
Department of Statistics
Columbia University in the City of New York
hh2816@columbia.edu

Fengyang Lin
Department of Statistics
Columbia University in the City of New York
fl2542@columbia.edu

Yifei Xu
Department of Statistics
Columbia University in the City of New York
yx2577@columbia.edu

Xiyao Yan
Department of Statistics
Columbia University in the City of New York
xy2431@columbia.edu

Abstract

Image style transfer has been one of the novel applications of neural networks on texture transfer. The project implemented a style transfer algorithm to combine the content of an arbitrary photograph with the appearance of numerous well-known artworks. Furthermore, to study effects of different hyperparameters, we performed experiments on different setups of the network such as number of iterations, content and style weighting factor, learning rate and so on. Our work also deciphered roles for different layers by visualizing layer outputs and demonstrated a comparison on networks with different pre-trained model weights.

1 Introduction

1.1 Goal

The goal for the project is to implement a style transfer algorithm, which is a part of texture transfer. In other words, the algorithm should be able to transfer the style from one image onto another, while preserving the content of the target image. Accordingly, the problem can be divided into three parts. The first task is to extract the semantic image content of the target image with those pre-trained Convolutional Neural Networks for object recognition. The next step is to extract the image style from the source image and obtain the style representation. Lastly, rendering the semantic content of the target image in the style of the source image will generate the new style-transferred image. Furthermore, we would try different models and layers of neural networks to compare their performances on style transferring.

1.2 Related Work

Gatys et al. proposed A Neural Algorithm that use image representations derived from Convolutional Neural Networks optimized for object recognition. The algorithm allows people to separate and recombine the image content and style of natural images. [1,2] To reduce the run time, Johnson et al. then developed a real-time style transfer method with similar qualitative results but is three orders of magnitude faster. Their work also shows that with single-image super-resolution, replacing a per-pixel loss with a perceptual loss could provide visually pleasing results. [3] More recently, Dongdong et al. proposed StyleBank that composed of multiple convolution filter banks and each filter bank explicitly represents one style. [4]

Our implementation is mainly based on Gatys's work on Image Style Transfer. In this project, we would use the same architecture of Neural Networks and loss function as in Gatys's work to implement

our algorithm. [1,2] We also read Johnson and Dongdong's paper to gain a deeper understanding of the application and improvement of style image transfer. [3,4] TensorFlow's official tutorials on Neural style transfer and Victor's code on Kaggle serve as a starting point of our implementation. [5,6]

1.3 Data Description

Our task is not a typical machine learning problem. It is neither classification nor regression, thus we do not have to split the training/validation/test samples. In our task, there are two kinds of data, one is the target image (also referred as content image), the other is the style image (also referred as source image). Our goal is to extract "style", the painting features, from style image and then to apply them to the target image. As a result, we could create a new image which integrates the content of target image and the visual properties of style image together.

For the basic part of our project, we want to apply different styles of great artists to the photos we took in our daily life. We will pick two content images and two style images (paintings, photographs, etc), and test our algorithm over the 4 resulting combinations of content and style. This is the main goal of our project.

Style images are all from a kaggle dataset, Best Artworks of All Time, a collection of paintings of the 50 most influential artists of all time. `https://www.kaggle.com/ikarus777/best-artworks-of-all-time`. We picked Vincent Van Gogh's *Starry Night* and Pablo Picasso's *Two Girls Reading*. Base images are photos taken by our group members, including the view of Columbia University and New York City.

In our task, we also want to creatively apply the style of one artist to another artist, which may produce a strange result. In addition, we could also apply the style of one real photo to another photo. We believe that these are all interesting attempts.

2 Methods

2.1 Starting Point

The results of the original paper were generated on the basis of 19-layer VGG network. We started from reproducing the results with the setting in the original paper, i.e. the layers we picked from the VGG19 to extracts the contents and styles from the images and the hyperparameters including loss weighting factors.

To be precise, we implemented the following steps and architecture as proposed in Gatys's paper, which is also shown in Figure 1.

- **Content Representation**

  To extract and store the content feature, the content image $\vec{p}$ and a randomly-generated white noise image $\vec{x}$ image are passed through the network. P^l and F^l are their respective feature representation in layer l. We then calculated the squared-error loss between the two feature representations

$$\mathcal{L}_{content}(\vec{p}, \vec{x}) = \frac{1}{2} \sum_{i,j} (F_{ij}^l - P_{ij}^l)^2 \tag{1}$$

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

- **Style Representation**

  To extract and store the style feature, the style image $\vec{a}$is passed through the network. We use Gram matrix G^l to represent the style of the source image in layer l, where G_{ij}^l is the inner product between the vectorised feature maps i and j in layer l. Like what we did in content representation, we introduced $\vec{x}$ and minimise the mean-squared distance between the entries of the Gram matrices of $\vec{a}$ and $\vec{x}$. A^l and G^l are their respective stype representation in layer l. The contribution of layer l to the total loss is then

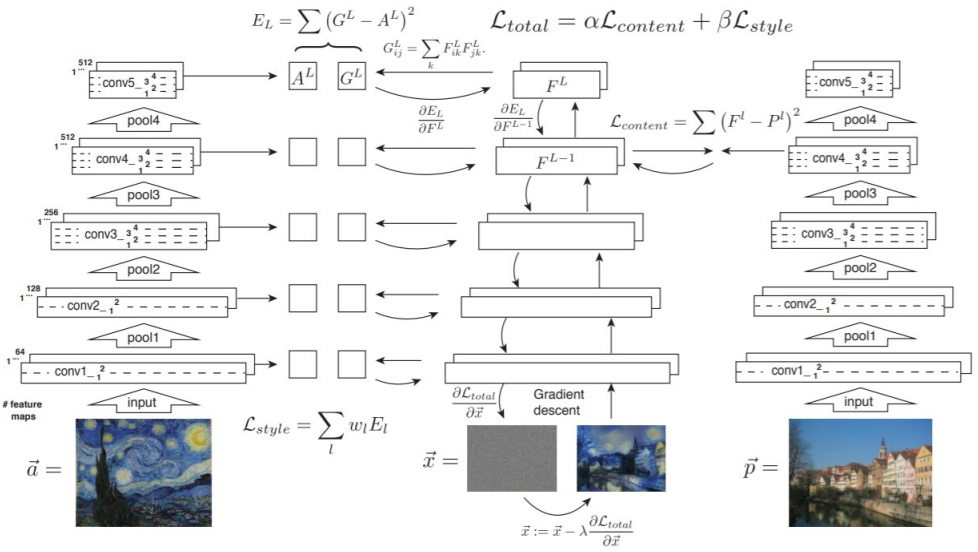


Figure 1: Style transfer algorithm from Gatys's paper.

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2 \tag{2}$$

and the total style loss is

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

where w_l are weighting factors of contribution of each layer to the total loss.

- **Transfer Representation**

  The total loss is then a linear combination between the content and the style loss.

$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}, l) = \alpha\mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta\mathcal{L}_{style}(\vec{a}, \vec{x}) \tag{4}$$

  where α and β are the weighting factors for content and style reconstruction, respectively.

With the total loss function, we are able to construct the network similar to Gatys' paper. This is the first step of our project and it helps us understand the idea of Neural Style Transfer better, therefore we are able to build our own framework for the later work.

2.2 Algorithm Improvement

After we built the framework, we tuned the following hyperparameters used in the algorithm.

- number of iterations
- different random seeds
- different weights on loss
- different learning rate
- the layers chosen from VGG19
- different pre-trained model weights

We compare the output images from different settings to find contributing factors and see if there are any interesting findings.

3 Results and experiments

In this section, we present our results and some experiments. The 2 content images are photos taken by our group members - the first one is a photography of our Morningside campus with a view of Alma Mater from behind during nightfall, the second image is a view of Manhattan. The two style images are well-known artworks. We pick Vincent van Gogh's *Starry Night* and Picasso's *Two Girls Reading* as the style images. In most following experiments, we pick *Starry Night* as the style image and *Nightfall at Columbia* as the content image.

3.1 Algorithm results

To validate our neural style transfer algorithm, we perform tests on 2 content images and 2 style images.

Figure 2 shows the rendered images, each of them is a mixture of the selected content image and style image. In terms of details of implementation, here we present results after 1000 iterations using an Adam optimizer with a learning rate of 0.03. We use VGG19 as our pre-trained model. The default style layers are ('conv1-1', 'conv2-1', 'conv3-1', 'conv4-1' and 'conv5-1') and 'conv4-2' is the default content layer. We also set random seed to 111 and α/β ratio to 1×10^{-3}.

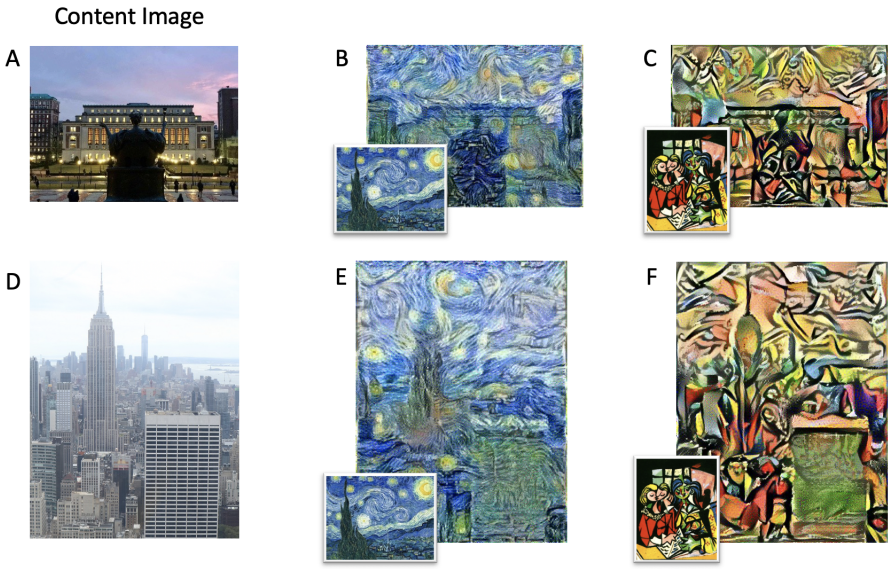


Figure 2: Images that combine the content of a photograph with the style of several well-known artworks. A and D are original content images, while B, C, E, F are the rendered images with the style image shown in the bottom left corner of each panel. A: *Nightfall at Columbia* D: *Good morning Manhattan*. B, E are rendered versions of A in the style of *The Starry Night* by Vincent van Gogh, 1889, C, F are rendered versions of D in the style of *Two Girls Reading* by Pablo Picasso, 1934.

Furthermore, in order to understand the features extracted by different layers of the neural network, we visualize the layer outputs in 3.

To study effect of different setups of the network on the transferring results, we then perform several experiments on number of iterations, random seed, content-style loss ratio and learning rate. We also try different style layers, content layers and pre-trained model weights to design different transfer algorithms.

3.2 Effect of number of iterations

Since the algorithm is updated constantly in iterations, the number of iterations decides how many times the results is updated to match the content image and style image. We ran the algorithm

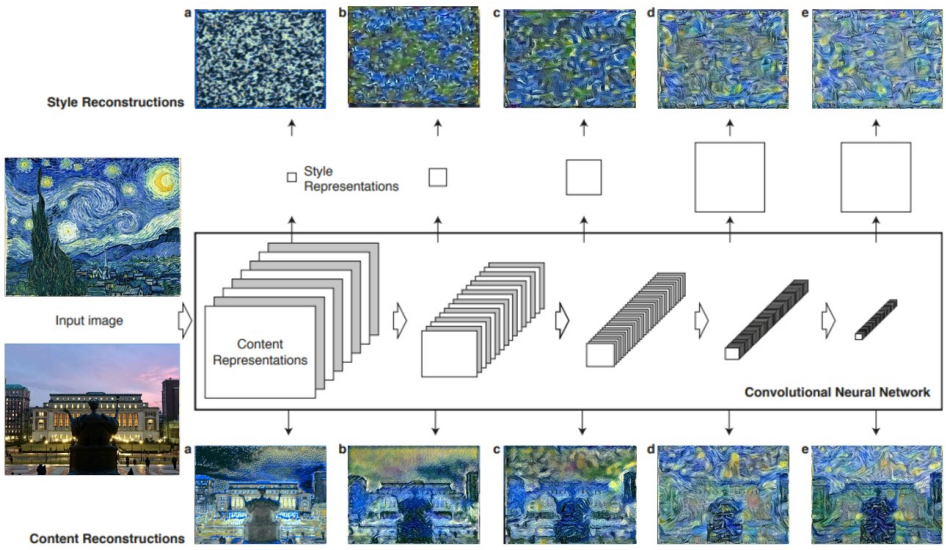


Figure 3: Layer outputs

for 15000 iterations, and saved the result of the 10^{th}, 50^{th}, 100^{th}, 500^{th}, 1000^{th}, 5000^{th}, 10000^{th}, 12000^{th}, 14000^{th} and 15000^{th} iteration. The result is shown in Figure 4.

Comparing these images generated from different numbers of iterations, we could see that as the number of iteration raises, the new generated image will match more closely to the appearance of the style image, effectively giving a texurised version of it, but show less of the base image's content. Before the 500^{th} iteration, we can see the texture of Van Gogh, but the style of the *Starry Night* is not that clear. The sky is still pink as it's in the content image. Then, from the 1000^{th} iteration, the style and texture of *The Starry Night* is more clear. The sky is more similar to the sky in the *Starry Night*. But then after the 5000^{th} iteration, it's hard to recognize the Alma Mater's arms and head.

After considerations on balance of content image and style image, as well as the computing resources, we choose 1000 iterations as our best parameter.

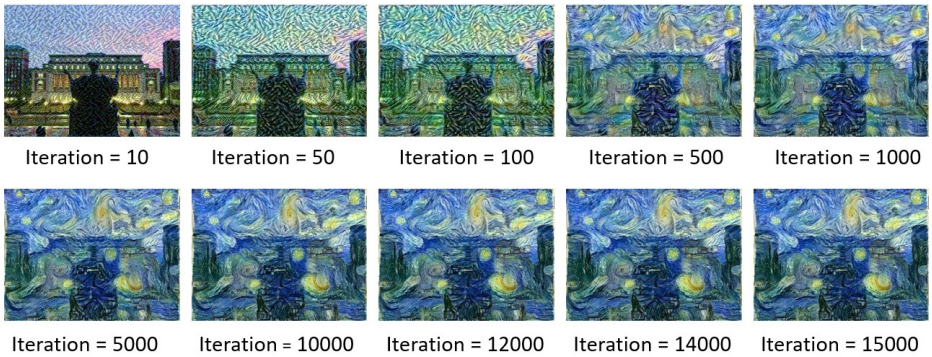


Figure 4: Comparison of different numbers of iterations

3.3 Effect of Random seed

In the algorithm, some parts are influenced by the randomness. Thus, we think different random seed may result in different outputs. We set and changed different random seed in our experiments. Comparing the results with random seed = 111, 222, 333 and 999, we found that the randomness does contribute to the final result. Some differences between the image generated with random seed of 111 and 999 are shown in Figure 5. We could see, when the random seed = 999, the Alma Mater is

bluer and the moon is darker. When the random seed = 111, the Alma Mater is darker and the moon is more white.

Thus, the random seed makes difference to the output. We controlled the random seed and set it to 111 in our other experiments.

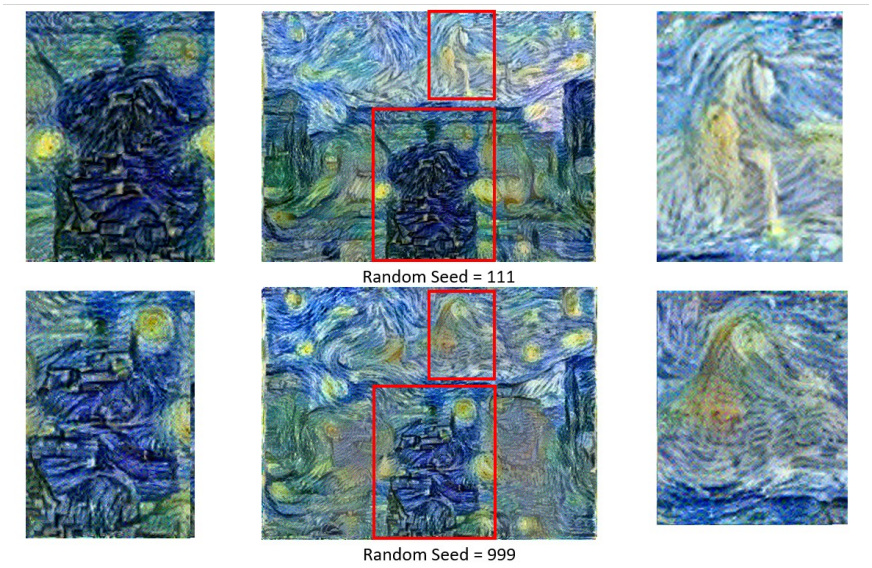


Figure 5: Comparison of different random seed

3.4 Trade-off between content-style matching

Since total loss is the objective of the algorithm, the ratio of content and style loss controls the power of style transfer algorithm. Intuitively, as the style weight factor increases and the style loss takes a greater part of total loss, the rendered image will be closer to the style image. Figure 6 presents an example of effect of changing the content-style loss weight. From left to right, the α/β ratios are $1 \times 10^{-4}, 1 \times 10^{-2}, 1 \times 10^{2}, 1 \times 10^{4}$ with higher emphasis on the style. The result is consistent with what we expected - a higher emphasize on the style leads to stronger style "transfer", as the rightmost image demonstrates more curly brush strokes that Vincent van Gogh is famous for. Tuning the content-style loss weight ratio is the same as tuning the power of style transfer algorithm. The hyperparameter serves like the level of filter in real-life application and worth further exploration.

In our other experiments, α/β ratio is set to 1×10^{-3}.

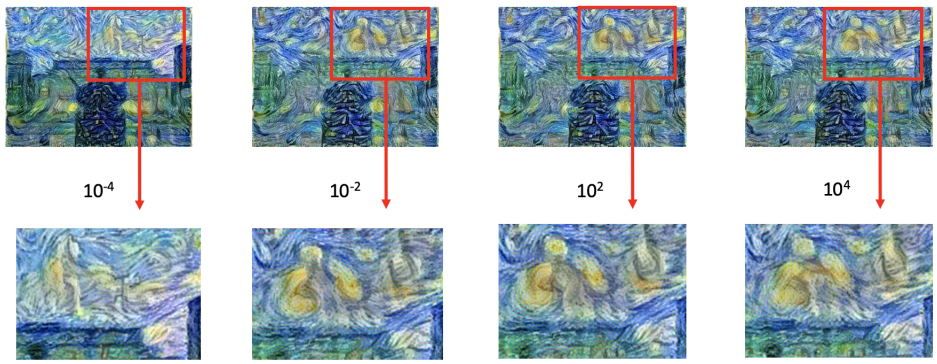


Figure 6: Relative weighting of matching content and style of the respective source images

3.5 Effect of learning rate

In our project, learning rate in the optimizer is a crucial parameter since it controls the speed model learns from the style image. We tuned this parameters from 0.01 to 0.1 in the experiments. In Figure 7, we show the results on lr=0.01, lr=0.05 and lr=0.1 at their 1000 and 2000 iterations. Comparing the figures in each column, we could see that under all three learning rate, the model learns more style information from the style image. Looking at each row, we found that as the learning rate raises, the created image will become more abstract at the same iteration. Specifically, with lr = 0.01, we could hardly conclude which style image it applied since it is too close to the original photo; with lr = 0.05, we could surprisingly see that the lights in front of the Butler library are changed to the yellow "moons" in the *Starry Night*; while with lr = 0.1, we could not recognize the content of image anymore.

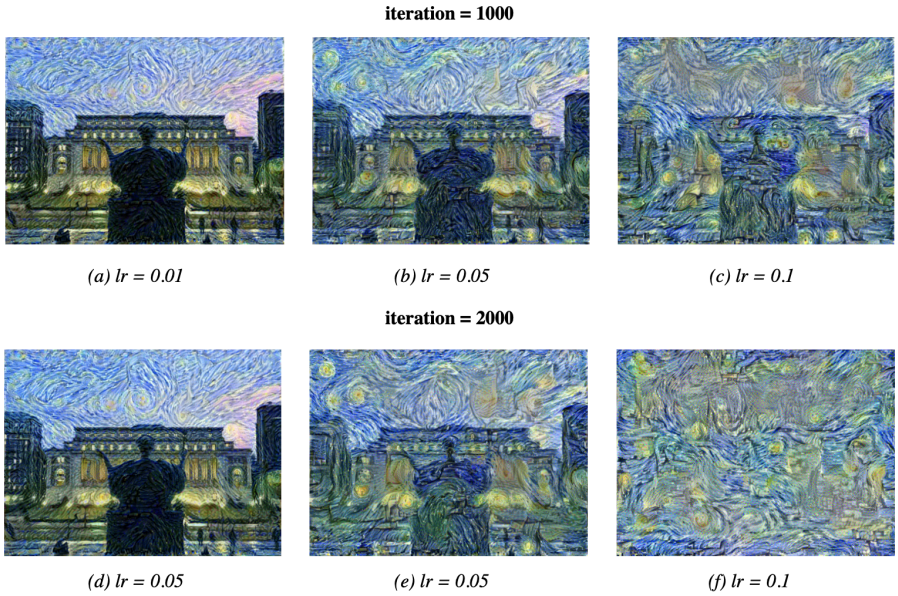


Figure 7: Comparison of different learning rate

Furthermore, with a lower learning rate such as 0.01, the model learnt more slowly and even at iteration = 5000, we had the similar image as the original photo. With a higher learning rate such as 0.1, the model learnt too fast and induce too many noise, which would produce a image losing the base image content.

According to the results of our parameter tuning, we choose 0.03 as our best learning rate.

3.6 Effect of different layers chosen from the pre-trained model

Since we would only use some layers from the pre-trained model, it is obvious to think the choice of different layers may influence the final output. It is believed that higher layers tend to study higher-level structures than lower ones, and this difference would promise smoother, or maybe more abstract output for the model with higher layers.

Comparison of different layers used as content layers is shown in Figure 8. After 1000 iteration, output based on features from the content layer ‘conv4-2’ somehow still preserved the outline of buildings, while these couldn’t be figured out in output from content layer ‘conv5-2’.

According to the result, we chose ’conv4-2’ as vest content layers.

To verify the knowledge of each layer containing different aspects of features for an input, we compared different style layers set, using different style layers set. By changing the first layer of the style layers set from ‘conv1-1’ to ‘conv1-2’,we coule detect slight difference in Figure 9. The output

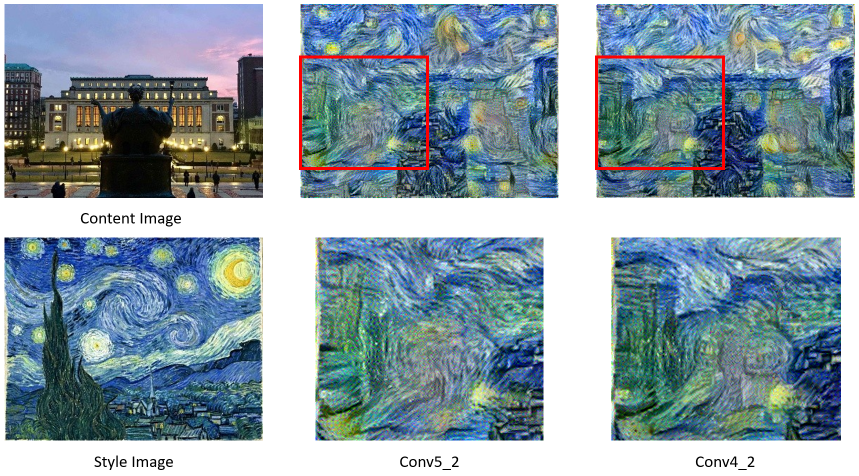


Figure 8: Comparison of different content layer

from ‘conv1-2’ is bluer than the other one. This might result from ‘conv1-2’ caring more about blue part of the style image.

Because the difference of using different style layers set is not so obvious, we would still use the default style layers set (‘conv1-1’, ‘conv2-1’, ‘conv3-1’, ‘conv4-1’ and ‘conv5-1’) as hyperparameters for the best model.

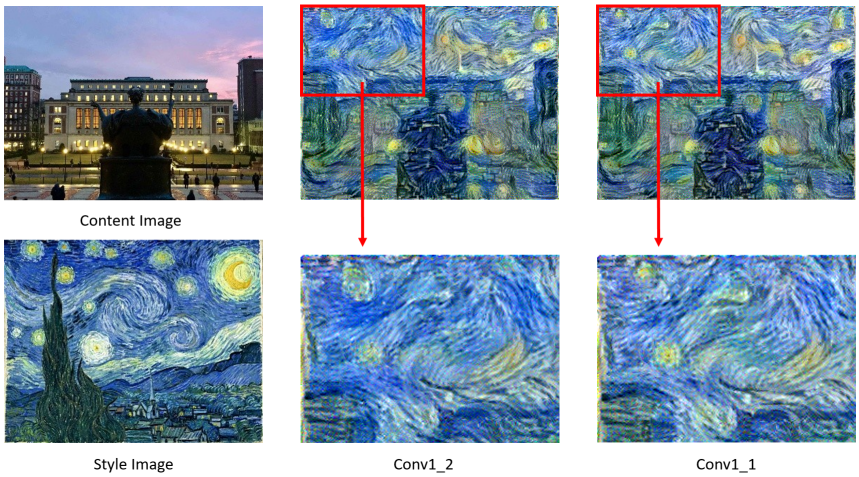


Figure 9: Comparison of different style layers set

3.7 Effect of using different pre-trained model weights

Using the pre-trained weights (vgg19_weights _tf_dim_ordering_tf_kernels) from the Github link, we wanted to see if different pre-trained parameters could lead to obvious difference in output image. Resizing has been done for both input image and output image for different model input shape requirement. The result is shown in Figure 10, these stored parameters might be trained on different datasets or used for other purpose, so the output can be quite different. We could hardly tell the shape of Alma Mater using the alternative pre-trained weights.

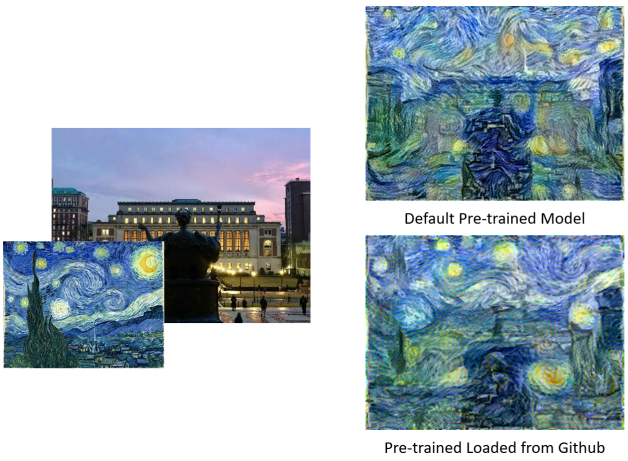
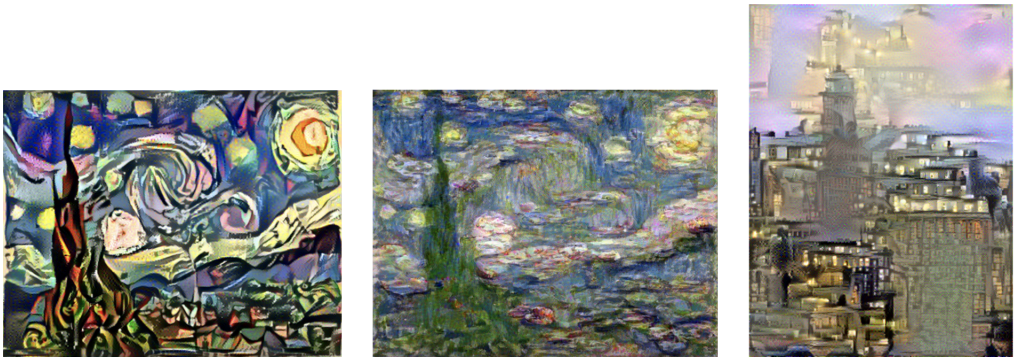


Figure 10: Comparison of different pre-trained model weights

3.8 Creative results

We tried something creatively, that is, we applied the style of one artist to the drawing of another artist. Since the styles between different artists are quite different,we were wondering that how about integrating the styles of different artists into one "drawing". Then we used the *Starry Night* as the content image, and the previous drawing from Picasso and *Water Lilies* from Claude Monet as style images. We got the new "drawing" as (a), (b) in the Figure 11. Surprisingly, we could easily find out the styles of artists from the new images.

Another creative experiment is that we applied the style of one real photo to another one. As the camera in our smartphone becomes more and more efficient, we are used to taking photos to record our lives. Even if for the photos, different people may produce different styles of photos. Thus, we also considered using the photo as style image. The *Nightfall at Columbia* was applied to the *Good morning Manhattan* as the style image. The result image is quite fancy, the model changed the morning sky at Manhattan to pink and purple. Also the windows of buildings are 'lighted' by the lights in front of the Butler library. The *Good morning Manhattan* was perfectly changed to 'Good night Manhattan'.



(a) Apply Picasso style to The Starry Night

(b) Apply Monet style to The Starry Night.

(c) Apply Columbia nightfall to Manhattan Building Views

Figure 11: Creative results

These attempts are surprisingly success and we could create more interesting images by different combination.

4 Disscussion

For us, this project helps us better understand how Neural Networks models extract different features of input (like content image and style image in this project) in different layers. When using the pre-trained model to do style transfer, it was found that lower layers simply taken the exact pixel values of the original image as features and then optimized. On the contrary, higher layers might capture high-level features of images and thus, give a better result when reconstructing the mixed output.

Similar inference can be applied to other applications of Neural Networks. In tasks like image recognition and localization, models with more layers usually have better results. This may lead to the classic discussion of how many layers are proper for a model.

However, we do think there is a limitation of the current algorithm. It's hard to tell which model or set of hyperparameters are better when tuning, for the loss may always have a great value and is not a good metric. Standard by people can be introduced, in our opinion, to decide the best model. We can invite people to choose the best one of the outputs from different models for one style, or to rate them. The best model can be chosen based on these results. This might not be the perfect metric for it is all about personal aesthetic standards, but still, it's another way to think about further work.

Last but not least, further research can focus more on real-time transfer mentioned in Johnson's paper, or using content video instead of content image to be the base. Every frame of the video can be seen as a content image, to which we can apply the algorithm. In addition, we can build a user-interface website which allows user to upload their own image. The algorithm can be used to help users to apply style filters to the uploaded photo.

References

[1] Gatys, Leon A., Alexander S. Ecker, and Matthias Bethge. "Image style transfer using convolutional neural networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[2] Gatys, Leon, Alexander Ecker, and Matthias Bethge. "A Neural Algorithm of Artistic Style." Journal of Vision 16.12 (2016): 326. Crossref. Web.

[3] Johnson, Justin, Alexandre Alahi, and Li Fei-Fei. "Perceptual losses for real-time style transfer and super-resolution." European conference on computer vision. Springer, Cham, 2016.

[4] Chen, Dongdong, et al. "Stylebank: An explicit representation for neural image style transfer." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

[5]"Neural Style Transfer nbsp;: nbsp; TensorFlow Core." TensorFlow, `www.tensorflow.org/tutorials/generative/style_transfer`.

[6]Basu, Victor. "Style Transfer Deep Learning Algorithm." Kaggle, Kaggle, 26 Apr. 2019, `www.kaggle.com/basu369victor/style-transfer-deep-learning-algorithm/notebook`.