A Neural Algorithm of Artistic Style

E4040.2016Fall.DCBA.report

Bo Jiang bj2344, Haixu Ying hy2489, Jiechao Gao jg3752

Columbia University

Abstract

This project aims at introducing a neural network based algorithm that is able to create artistic images with the style and content extracted from arbitrary images. The success of the algorithm hinges on the ability to capture the representation of style and content of an image. We reviewed methodology proposed in the original paper (Leon A. Gatys, Alexander S. Ecker, Matthias Bethge, 2015) and applied it with modification. Our results showed that the algorithm is capable of obtaining style and content representation from any pictures and mix them to product a new artistic picture of high quality.

1. Introduction

In fine art, master pieces are usually characterized by extraordinary combination of content and style. However, the generating process remains a mystery. On the other hand, researchers have successfully implemented algorithms that can match human performance with the biologically inspired vision models called Deep Neural Networks in many subfields of computer vision. The goal of this project is to develop an algorithm to understand how human percept style and content, and mimic the process of creating artistic images.

The technical challenge is to retrieve content and style of any input images. Early researcher have shown that feature responses in higher layers of Convolutional Neural Network, one of Deep Neural Networks, are able to capture content information. In addition, researchers have shown that a feature space, originally designed for texture information (Leon A. Gatys, Alexander S. Ecker, Matthias Bethge, 2015), is able to capture the style information.

2. Summary of the Original Paper

2.1 Methodology of Original Paper

The authors have presented different methods to capture content information and style information contained in an image.

2.2 Key results of the Original Paper

The overarching discovery the original paper is that style and content can be separated independently from neural network.

The similarity between the contents of two images can be characterized by squared-error loss between the two corresponding feature representations of the original image and the generated image:

$$L_{content} = \frac{1}{2}\sum_{i,j}(F_{ij}^{l} - P_{ij}^{l})^2$$

And the similarity between the styles of two images can be characterized by the weighted loss from each layer on a measure that will be discussed later:

$$L_{style} = \sum_{l=0}^{L} w_l E_l$$

where:

$F^l \in R^{N_l \times M_l}$ is the matrix that stores the responses in a layer l, which has N_l features map each of size M_l, for original image

$F^l \in R^{N_l \times M}$ is defined similarly.

$E_l = \frac{1}{4{N_l}^2 {M_l}^2}\sum_{i,j}(G_{ij}^{l} - A_{ij}^{l})^2$ is a measure of the difference between generated images and original images in layer l

G^l , $A^l \in R^{N_l \times N_l}$ are the Gram matrices that store the correlations between features maps of generated image and the original image

The papers shows that by minimizing a linear combination of content loss and style loss, the algorithm is able produce an image of the content and the style from respective images.

3. Methodology

3.1 Introduction

In this section, the methods in the original paper are presented. Namely, how content and style can be captured from the parameters of a neural network. Furthermore, intuitions of these methods will also be discussed.

3.1.1 Content representation

The methods used in the original paper are based on VGG-Network, a Convolutional Neural Network that is trained for objective recognition. In convolutional neural network, each layer is made up of a number of filters of certain height and width. A filter is designed to move across every possible region, with the same height and width of the filter, of the image and meanwhile, computing the dot product between the filter and respective entries of the image. The result is recorded at every possible region for every filter. The network will learn useful features, if present in the images, such as the edge of the object. It also can be shown that as we move through the hierarchy of the network, the layers of higher levels will focus more on the actual content of the image rather than the actual pixel values, compared to layers with lower levels (Leon A. Gatys, Alexander S. Ecker, Matthias Bethge, 2015). Intuitively, the complexity increases along the hierarchy: filters in subsequent layers are learned upon the response of input images to preceding convolutional layers. Thus they are able to capture more 'global' information than 'local' information.

The feature representation of the content of any input image at layer l can be stored in a matrix

$$F^l \in R^{N_l \times M_l}$$

where F_{ij}^l is the activation of ith filter at position j in layer l , N_l is the number of feature maps and M_l is the size (height by width) of the feature map.

Similarly, feature representation of the content of the original image can be store in a matrix F^l that is of the same dimension of F^l. By minimizing the square-error loss between the two with respect to the input image,

$$L_{content} = \frac{1}{2}\sum_{i,j}(F_{ij}^l - P_{ij}^l)^2$$

we are forcing the generated image(usually a white noise image at the beginning) to exhibit same features as the original picture. Thus, the content of the original image can be mimicked. The derivative of the loss with respect to the activation has simple formula:

$$\frac{\partial L_{content}}{\partial F_{ij}^l} = \begin{cases} (F^l - P^l)_{ij} & if\ F_{ij}^l \geq 0 \\ 0 & if\ F_{ij}^l \geq 0 \end{cases}$$

3.1.2 Style representation

In the original paper, the authors use the correlations between feature maps to represent the style of an image. To be specific, suppose layer l has N_l feature maps each with size$M_l = height \times width$, the Gram matrix $G^l \in R^{N_l \times N_l}$ stores the correlation information of all features maps. $G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l$, where F_i is the vectorized ith feature map in layer l. It is clear that G_{ij}^l is proportion to the correlation between the ith and jth vectorized feature map.

Suppose the Gram matrix, in layer l, for the original image and generated image (usually a white noise image at the beginning) are denoted by A^l, G^l respectively, the similarity between the styles of generated image and original image in layer l can be measure as the mean squared distance between the entries of A^l and G^l : $E_l = \frac{1}{4N_l^2M_l^2}\sum_{i,j}\left(G_{i,j}^l - A_{i,j}^l\right)^2$. Total loss is the weighted average of losses of all layers: $L_{style} = \sum_{l=0}^{L} w_l E_l$

The derivative of E_l with respective to activations in layer l is computed as follow:

$$\frac{\partial E_l}{\partial F_{ij}^l} = \begin{cases} \frac{1}{N_l^2 M_l^2}((F^l)^T(G^l - A^l))_{ji} & if \quad F_{ij}^l < 0 \\ 0 & if \quad F_{ij}^l > 0 \end{cases}$$

For a white noise image input, the generated image that possesses the content of one image and the style of another image can be created by minimizing the linear combination of content loss and style loss, namely: $L_{total} = \alpha L_{content} + \beta L_{style}$, where α, β represents the relative weight between content and style.

3.2 Objectives and Technical Challenges

In this project, we extracted the contents and the styles from two images, and generated a new image using the corresponding content and style. In other words, we produced new artistic images by mixing the characteristics from two original images. Also, we found that the algorithm has a wider application. And we could do more than mixing two images with the algorithm.

The result varies: it depends on how we get content and style. An effective method of representing the information to produce the new image is needed. The methods we used here to represent the characteristics are discussed above.

Another important procedure is the optimization loss function. There are several methods to optimize total loss with respect to activations. An efficient algorithm is the Quasi-Newton method, which has several variations itself. Stochastic Gradient Descent is another way that is widely used in deep learning. However, both methods target at finding the local minima rather than the global minima. Therefore, the initial state of the algorithm or the parameters has a large impact on final result.

After experimenting with both Quasi-Newton method and Stochastic Gradient Descent, we found both methods have their pros and cons. And we can also derive the difference in a mathematics way. Basically, the Quasi-Newton method will get a faster convergence while the Stochastic Gradient Descent might get a more satisfying result. It is worth noting that it is hard to say which is better since both methods have several parameters which also influence the result. The final method we used in this project is Stochastic Gradient Descent. We used momentum to prevent oscillation and get a faster convergence.

Moreover, another challenge is that there is no absolute way to determine whether one result is better than another. The results are artistic images and different people prefer different ones. Therefore, in our project, we adjusted the algorithms to get different results and presented them in the report. We explored the relations between the differences in the algorithm and their corresponding results.

4. Problem Formulation and Implementation

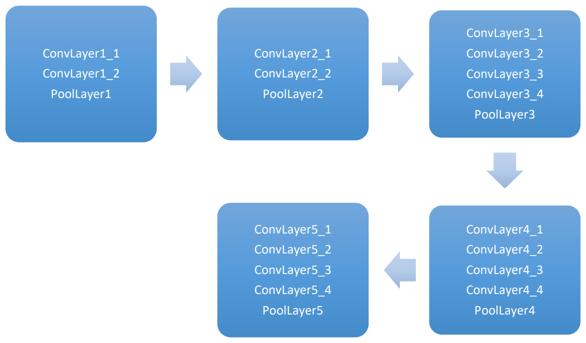
In this section, a brief introduction of the structure of the neural network used in the problem is given, followed by a description of the training algorithm.

4.1 Deep Learning Network

Convolutional Neural Networks (CNN) is a common method we usually use as deep learning network. CNN is comprised of one or more convolutional layers and then followed by one or more fully connected layers as in a standard multilayer neural network. The architecture of a CNN is designed to take advantage of the 2D structure of an input image. This is achieved with local connections and tied weights followed by some form of pooling which results in translation invariant features. Another benefit of CNN is that they are easier to train and have fewer parameters than fully connected networks with the same number of hidden units. In many ways we can build our CNN model by adding some convolutional layers or pooling layers. Sometimes we can also add layers such as Batch Normalization to make our CNN model more useful and effective. Also we can change the filter size and the number of filters. In our project, we choose to use VGG which is the same model as mentioned in the article.

There are two kinds of VGG: VGG16 and VGG19. We choose to use VGG19 to test the result. VGG is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition” [5]. The main contribution of VGG is a rigorous evaluation of networks of increasing depth, which shows that a significant improvement on the prior-art configurations can be achieved by increasing the depth to 16-19 weight layers, which is substantially deeper than what has been used in the prior art. To reduce the number of parameters in such very deep networks, we use very small 3×3 filters in all convolutional layers.

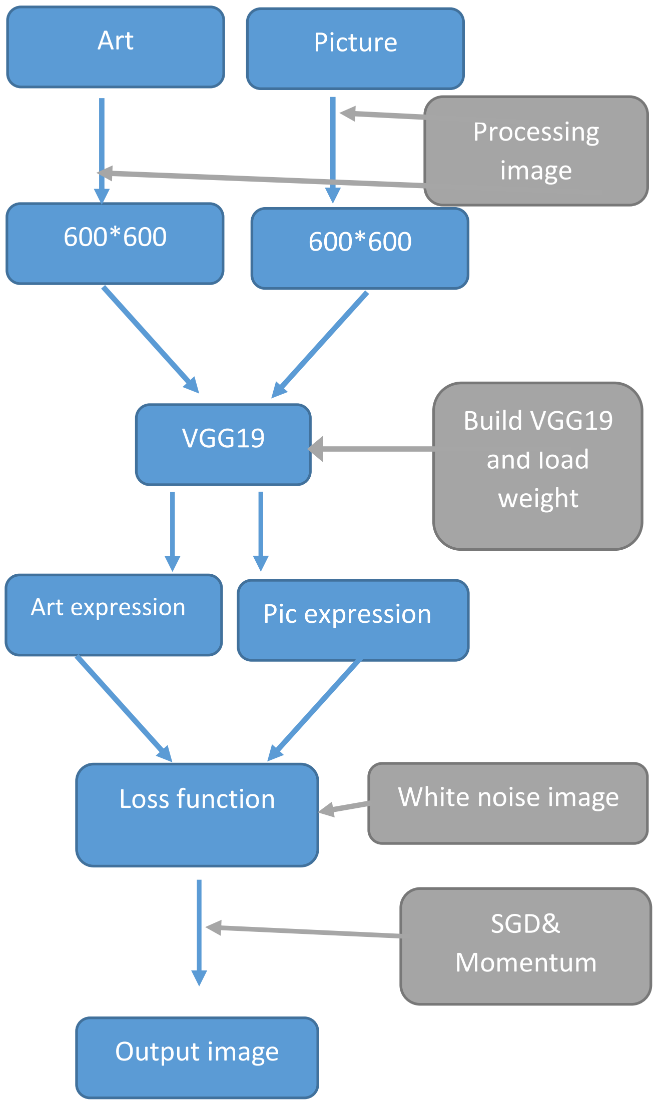
The construction of VGG19 model is given below:



After the network is build, pre-trained weights are loaded into the model.

4.2 Algorithm Design and Implementation

4.2.1 Algorithm Design



4.2.2 Optimization Algorithm

The optimization algorithm used in the project is briefly discussed in this section. Stochastic Gradient Descent (SGD) with momentum algorithm is used in the training process. After training, we reconstruct the images. To minimize the total loss, SGD with momentum is a considerable way to achieve the objective.

If the objective function has the form of a long shallow ravine leading to the optimum and steep walls on the sides, standard SGD will tend to oscillate across the narrow ravine since the negative gradient will point down one of the steep sides rather than along the ravine towards the optimum. The objectives of deep architectures have this form near local optima and thus standard SGD can lead to very slow convergence particularly after the initial steep gains [6]. Momentum is one method for pushing the objective more quickly along the shallow ravine. The algorithm is as below,

```
for i in range(iterations):
    set image value
    grad = gradient(total_loss, image)
    velocity = gamma * velocity + learning_rate * grad
    image = image – velocity
```

5. Result

5.1 Description

In our experiment, we used the same images as the original paper, namely, *Composition VII* by Wassily Kandinsky as the style image and the photograph of Neckarfront in Tubingen as the content image. Each row shows the training result of matching the style representation of CNN layers:

{‘conv1_1’},

{‘conv1_1’, ‘conv2_1’},

……

{‘conv1_1’, ‘conv2_1’… ‘conv5_1’}.

Each column corresponds to a certain relative weight of content and style, $\frac{\alpha}{\beta}$, which ranges from 10^{-9} to 10^{-6}.

5.2 Discussion

5.2.1 Fixed relative weight:

It is clear from Figure 1 that, with fixed relative weight of content and style (a particular column), the style of the generated image is closer to that of the original image when the subset of CNN layers from which style representation is constructed is increasing in size. This result is expected. Since correlation between all vectorized feature maps in a layer is used as proxy of the style information contained in that layer, the algorithm proposed is only able to obtain style representation contained in the subset of the CNN layers. As more CNN layers are included to reconstruct the style, the algorithm produces a picture that resembles more like the style image.

5.2.2 Fixed subset of CNN layers:

When a subset of CNN layers is fixed, it is clear from Figure 1 that, as the relative weight of content and style varies, the generated image exhibits extremely different characteristics. The total loss function consists of a linear combination of content loss and style loss. When α, the weight for content in the total loss function, increases, content loss has a larger influence on total loss. In other words, a reduction in content loss will contribute more to the total loss reduction than before. Consequently, the minimization algorithm emphasis on finding an image that has similar content information as the content image, resulting in an image that exhibits more content characteristics and less style characteristics.

5.2.3 Content representation in each layer

An assertion is made in Methodology part: the layers in higher hierarchy levels in CNN network contain more ‘global’ content information, while the layers in lower hierarch levels in CNN network contain more ‘local’ content information. In the light of Figure 2, the generated images from lower level layers are exact. The reason is that lower layers almost preserve the pixel values around the area. On the contrary, the images generated from higher level layers losses some ‘local’ information.

5.2.4 Style representation

As a confirmation, we also experiment on including an increasing number of CNN layers for style reconstruction. The results are shown in Figure 3. It is clear that as more and more CNN layers are included, the generated image becomes more similar to the original image in style.

5.3 Comparison of Results

Generally speaking, our algorithm is able to produce images of similar quality as the original paper. And results are consistent to the theory in the original paper. In particular, we have shown that

1) The influence of the size of selected subset of CNN layers on the style of the generated image.
2) The influence of relative weight of content and style on the characteristics of the generated image.

However, some differences remain:

1) We are unable to get similar pictures with$\frac{\alpha}{\beta} = 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}$. Instead, the algorithm experiments on $\frac{\alpha}{\beta} = 10^{-9}, 10^{-8}, 10^{-7}, 10^{-6}$.
2) Compared with the original pictures, our results seem to loss some brightness.
3) Training time also differs. For generating a particular image, our algorithm takes several hours using Stochastic Gradient Descent, or less than an hour using Quasi-Newton method. Although the training time of original algorithm is not mentioned in the paper, it is reasonable to assume that their training time will also vary when using different optimizing methods.

6. Conclusion

In this project, our group have explored the techniques of synthesizing the content of one image and style of another image to produce artistic images. We followed methodologies of the original paper, and experimented the algorithms in different settings. Our results are consistent to the results in the original paper.

However, our algorithm can be improved in at least the following two perspectives:

1) Modifying the style reconstruction method to capture the relationship between style representations in different layers.
2) Researching for more efficient optimization algorithms to reduce training time.

7. References

[1]https://bitbucket.org/e_4040_ta/e4040_project_dcba

[2] Leon A. Gatys, Alexander S. Ecker, Matthias Bethge. (2015). A Neural Algorithm of Artistic Style.

[3] Leon A. Gatys, Alexander S. Ecker, Matthias Bethge. (2015). Texture synthesis and the controlled generation of natural stimuli using convolutional neural networks.

[4] Leon A. Gatys, Alexander S. Ecker, Matthias Bethge. (2015, 05 27). Texture Synthesis Using Convolutional Neural Networks.

[5] K. Simonyan and A. Zisserman. University of Oxford. (2015, 04 10). Very Deep Convolutional Networks for Large-Scale Image Recognition.

[6] UFLDL Tutorial http://ufldl.stanford.edu/tutorial/

8. Appendix

8.1 Result Figures

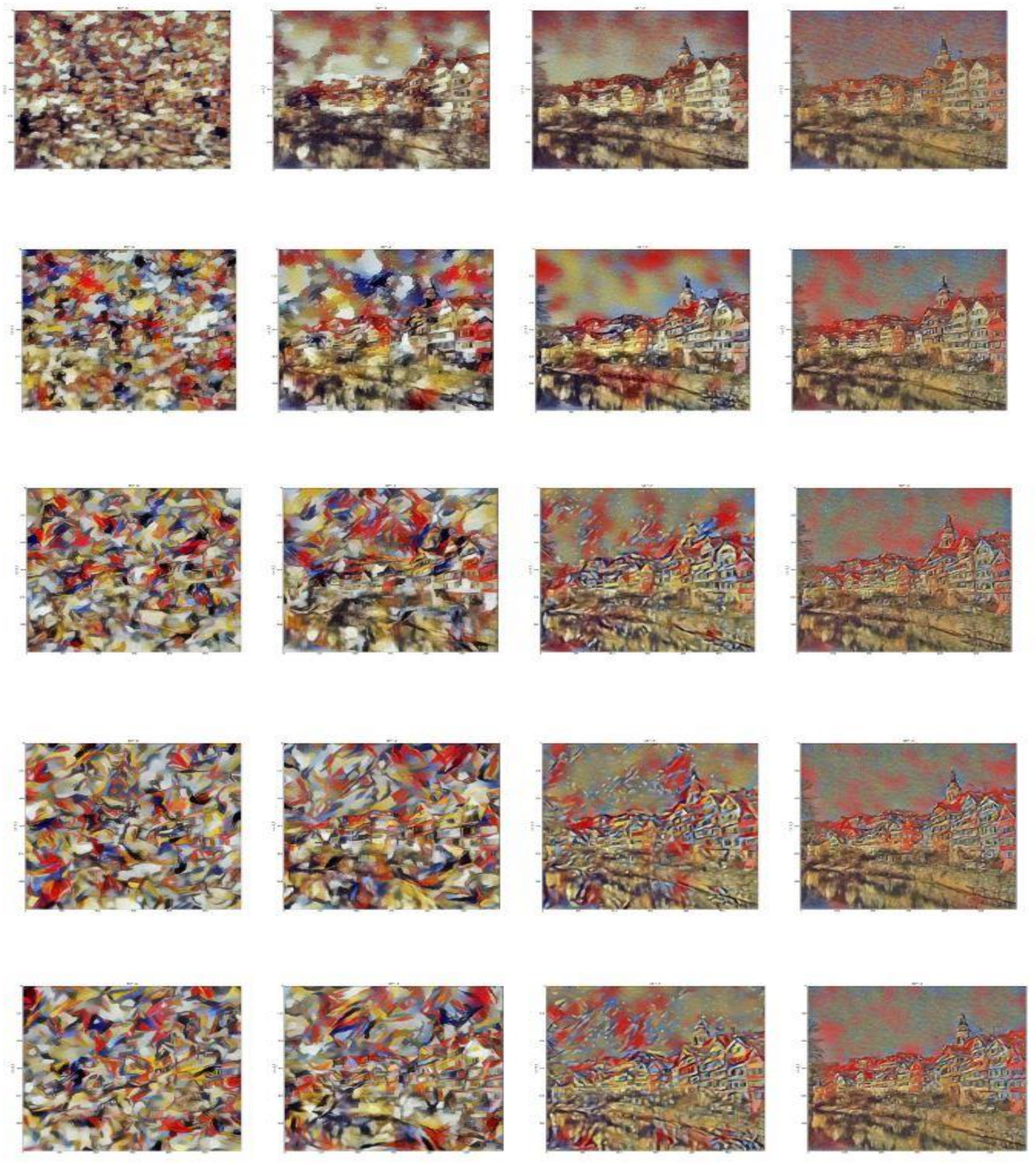


Figure 1: Reproducing Figure 3 in the original paper.

Each row corresponds to the results of matching the style of a certain subset of the CNN layers with different relative weights for content and style. Each column corresponds to the results of matching the style of an increasing number of subset of the CNN layers with fixed relative weight for content and style. Relative weight $\frac{\alpha}{\beta}$ ranges from 10^{-9} to 10^{-6}.

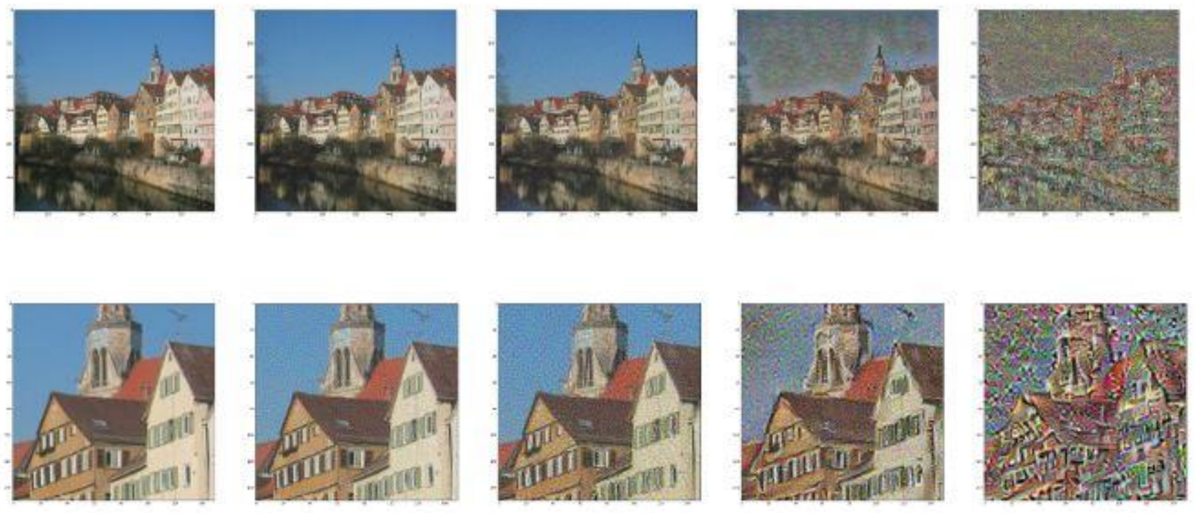


Figure 2 Content Reconstruction

The images are constructed with only content information from CNN layers 'conv1_1', 'conv2_1', 'conv3_1', 'conv4_1' and 'conv5_1'.

The first row are the generated pictures as a whole.

The second row only shows a particular region of the generated pictures shown in the first row for comparison.

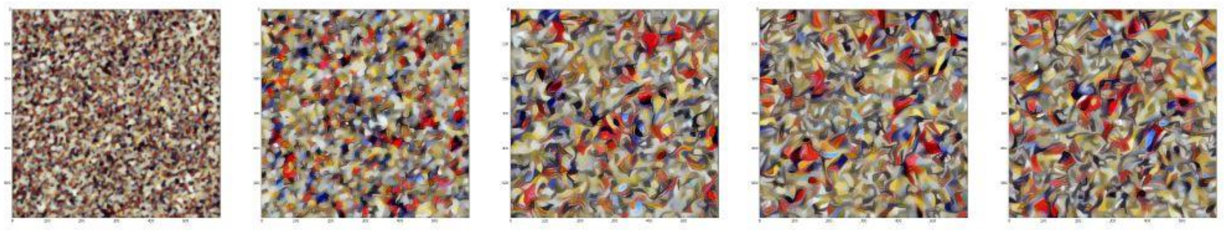
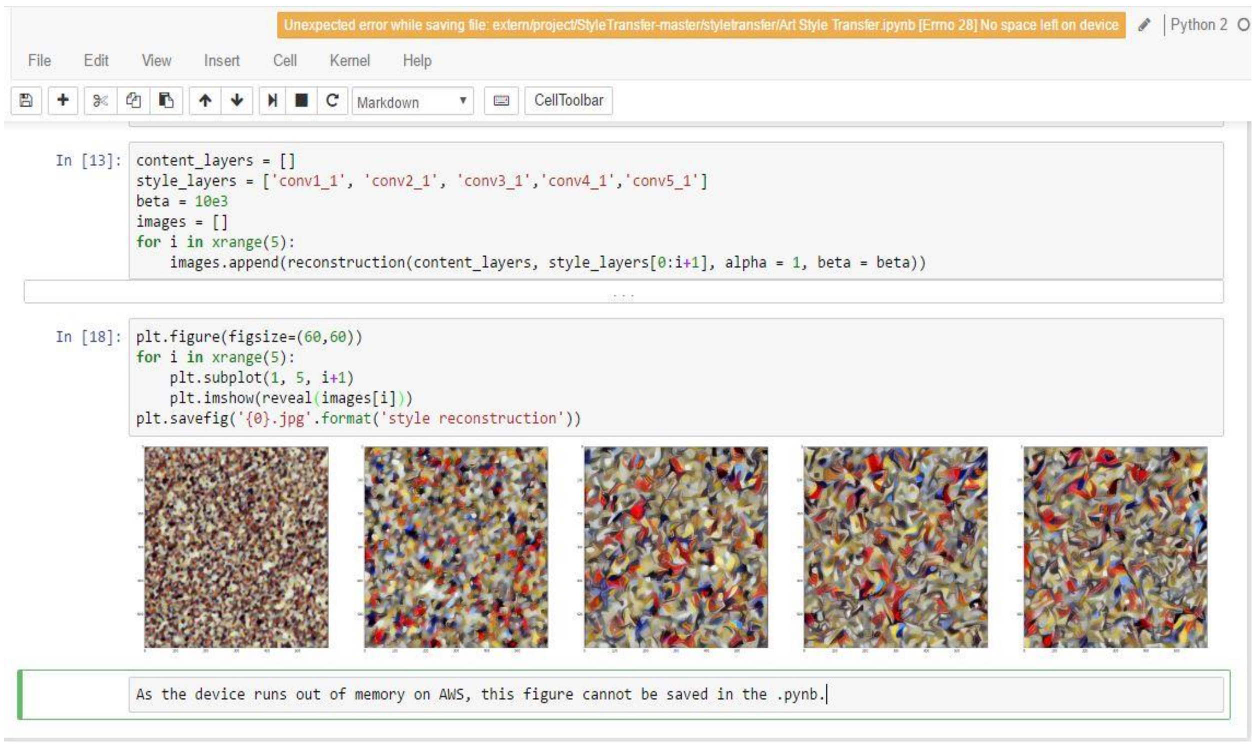


Figure 3 Style Reconstruction

Retrieving style representation from an increasing subset of CNN layers ‘conv1_1’, ‘conv2_1’… ‘conv5_1’.


Unexpected error while saving file: extern/project/StyleTransfer-master/styletransfer/Art Style Transfer.ipynb [Errno 28] No space left on device
Python 2
File
Edit
View
Insert
Cell
Kernel
Help
Markdown
CellToolbar

```
In [13]: content_layers = []
         style_layers = ['conv1_1', 'conv2_1', 'conv3_1','conv4_1','conv5_1']
         beta = 10e3
         images = []
         for i in xrange(5):
             images.append(reconstruction(content_layers, style_layers[0:i+1], alpha = 1, beta = beta))
```

```
In [18]: plt.figure(figsize=(60,60))
         for i in xrange(5):
             plt.subplot(1, 5, i+1)
             plt.imshow(reveal(images[i]))
         plt.savefig('{0}.jpg'.format('style reconstruction'))
```

As the device runs out of memory on AWS, this figure cannot be saved in the .pynb.

Figure 4

Error message when saving the style reconstruction images.

8.2 Individual student contributions

	bj2344	hy2489	jg3752
Last Name	Jiang	Ying	Gao
Fraction of (useful) total contribution	1/3	1/3	1/3
What I did 1	Researching methodology and designing algorithm.	Implementing the optimization algorithms.	Writing dlp_layers.py, reconstruct the training model functions.
What I did 2	Collaborating on the codes.	Building the running environment, testing	Drawing the Algorithm Design flow chart.
What I did 3	Drafting report.	Contributing to the report.	Contributing to the report.