Artist Painting Style Transfer with Convolution Neural Network

A thesis submitted to the National University of Ireland, Cork
in partial fulfilment of the requirements for the degree of
Master of Science in Interactive Media
in the Faculty of Science

October 2018

Moyu Wang
Department of Computer Science

Contents

List of Figures

Abstract

In this report, I implement a recently developed method and algorithm [17] of painting style transfer which can be also viewed as texture transfer from one painting to the other picture. This method is achieved by using two main technologies, one is visual texture modelling, the other one is image reconstruction. Both of those two technologies are based on convolution neural network which is a powerful machine learning network for object recognized. The convolution neural network (CNN) contains many convolution layers with a set of convolution kernels which can extract different levels feature details from an image. A convolution kernel can be viewed as a feature filter, it can be activated by a certain feature contained in input image. When input one image into CNN, the output generated from different layers is a set of feature maps, these feature maps indicate that if some certain feature contained in input image has been activated by convolution kernel. Hence, all these feature responses can be regarded as a content representation. Furthermore, a summary statistic method can be used to compute the correlations between these feature responses extracted by CNN, these correlations reveal the distribution of different features in the original image. This distribution can be regarded as a representation of image style. Since we have the representation of image content and style, we use a backward method which called gradient descent to reconstruct a new image. This method is to find an optimized image which has same content representation with target content image and same style representation with target style image by minimizing the loss function which defined by the distances between representations of different images. Hence, we can obtain an image contains a content of one image and a style of the other image which achieves style transfer. In this report, I will introduce the background of these technologies and implement it by using a machine learning platform TensorFlow built on Python, and giving the evaluation and the discussion of the results.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institution of learning.

Signed:
Moyu Wang

Acknowledgements

I would like to express my gratitude to all the people who has helped me during the this project. Especially, I am deeply indebted to my project supervisor Dr.John O' Mullane, who gave me many valuable suggestions and insightful guidance. I would also like to say thanks to my family, without their support, I cannot finish the study in Cork. I am really grateful for every teacher and friend in Cork. Studying here with all of you will be an absolutely wonderful memory in my life.

Chapter 1

Introduction

Since the computer science and industry developed rapidly, there is a significant number of complex work that computer can now perform instead of human beings. Also, there is a growing number of human abilities achieved by computer systems. For example, computer vision system can help computer understand images by divide one image into pixels. Based on pixels, there are lots of algorithms to help computer learn to recognize object. Since it is possible for computer to learn how to recognize an object, the simulation of computer vision needs to have a further understanding about human vision system. As a symbol of human civilization, paintings of different artists usually have various styles. Recognizing painting style could be regarded as a high-level human vision ability. So, how to develop a computer algorithm to understand these styles could be extremely challenging.

In order to understand how to represent style, there are many further researches on studying texture which can be regarded as a complex image representation. The style of an artist painting can be simply described as the preference of some particular lines, shapes and colors appeared in the painting. There are some early approaches [1, 2] regarded this preference as a texture of image when they developed an algorithm to generate similar image from origin image. But these approaches can only simulate simple texture of nature or artificial image, artist paintings could contain extremely complex texture. Since human visual system is not precise when distinguishing extremely small details of an image, a style of artist painting can be also approximately viewed as the texture of this painting.

According to previously research about texture generation [1, 2, 16] two main methods have been used to generate image has the same texture with the original image. One is non-parametric method [1, 2], one approach [1] based on non-parametric method is with the help of a Markov random field model, given an initial texture image piece, considering neighbor pixels and querying original image to generate every pixel per time. The other approach [2] of non-parametric method is that cutting original texture image into several patches then selecting a proper patch and stitching it with transferred image by calculating local image information. Although this non-parametric texture transfer method achieves high quality results and processing efficiency when dealing with natural texture, it does not give a certain universal model for generating nature or artificial texture particularly. It much more like a mechanistic process

that randomly chooses a texture patch compare with the patch of applied image and select most similar patch to synthesize.

The second method is to find a parametric model which can statistically measure the features of one image [16]. A certain feature can be defined by a particular mathematic model. For example, Figure 1.1 shows a chestnut image, one feature of this chestnut image is that every chestnut has an open mouth, a simple mathematic model which contain two intersecting arcs with some particular angles can be used to describe this feature. This mathematic model which described the open mouth feature will be activated by the corresponding feature appeared in image. The activated response which giving a numerical representation of image feature is an output parameter of the model for a certain feature in image. Hence, the image texture can be represented by the statistic measurement of all the output parameters which reveals the correlations of different features contained in the image. Therefore, if a parametric model contains enough number of mathematic models, statistical measurements of the model that can be used to obtain coefficient of enough number of different features contained in image to describe the image texture more accuracy. When input one image into a parametric model, the outcome would be a set of activation responses of features (model parameters) and a statistic measurement of these parameter which represent the image texture, if the other image has the same outcome statistic measurement of the first image, both two images share the same texture. Back to the example, if we can find enough feature to describe chestnut, there will be a particular coefficient between these features, if the other image has the same coefficient between these features, then it is probably another chestnut image.

Figure 1.1: Chestnut image

The concept of a parametric model used to generate output indicates the texture of an input image has been supported and promoted by many researchers since it was put forward. Julesz [6] first assumed that the pixels Nth-order joint histograms of one image can indicate a visual texture. Later, researchers found that oriented band-pass filters can simulate mammalian early visual system which response linear properties [8, 13]. Hence, instead of measuring the feature information directly extracted from pixels, the texture model can statistically measure the filter responses. Furthermore, steerable Pyramid [9], a linear filter bank has been used to filter different feature information from input image and get different feature responses to describe image. Containing a set of handcrafted mathematical models, a parametric model proposed by Portilla and Simoncelli [13] can have enough parameters to describe different features. And the statistic coefficient of these parameters can have good performance on representing not complex textures. The model gave good results when synthesizing many nature or artificial

texture. But this model cannot contain all the texture features of one image because it is limited by the number of handcrafted statistical models.

With the development of deep learning technology, many other research fields have experienced a rapid development also. One of improvement is the object recognition based on convolution neural network. Traditional method to identify object is to cut image into small pieces, and then compare each piece with target features. If the number of similar features big enough, then the patch contains matched pixels can be viewed as identified object. But this method compares small piece of image which contains pixels, it could fail to identify object when changing the shape of the object. Convolution neural network contains all useful features, a pretrained convolution neural network by a labeled image dataset which contains a quite large number of images with a label to indicate the object contained in the image will automatically extract useful features by optimal methods. So, this could solve the problem of traditional object recognize model.

Object recognition model with convolution neural network usually contain many layers, the convolution layer of CNN can be viewed as an extractor to extract different features. With the layers increasing, the performance of CNN to extract useful features is better. Hence, CNN provide a powerful parameter model to extract features of an image. the output of convolution layer is a set of feature maps which indicate different feature responses of the input image. With the help of the CNN model, Gatys [17] used the Gram Matrix to statistic measure the coefficients between different feature response. The element of Gram Matrix is the result of inner product between the output feature maps of a CNN convolution layer. Since the output contains many feature maps which one feature map indicate one feature response, the size of Gram Matrix will be determined by the number of feature maps. This Gram Matrix contributed a good result when used to represent image texture. In fact, the representation of texture method using Gram Matrix is using a second order statistic to modeling texture [20]. By using a back-propagation method to generate an image has the same Gram Matrix on the output of CNN model layers, the result image could have a similar texture with the original image. Later, N. Y. Wang and etc. [21] demonstrate the theory of style transfer from Domain Adaption, and they also proved that to minimize the difference between two Gram Matrix is similar to minimize the statistic measurement of two image based on the second order kernel function of Maximum Mean Discrepancy (MMD). Hence, there could a new method to approach style transfer.

In this report, I will demonstrate all the referred technologies and implement the style transfer algorithm with a convolution neural network and try a different style defined method.

Chapter 2

Convolution Neural Network

2.1 Background

Researches of mammals visual mechanism based on Visual Neuroscience shows that human visual system could be layer separated and abstract when recognizing images [7]. When a scene captured by human eyes, the light signal of the scene will be converted to electric signal by retina, later the signal will transmit to the visual cortex which is visual signal processor of human brain. In 1959, David and Wiesel [12] put an electrode into the primary visual cortex of cat, then displayed light bands with different shapes, locations and angles before cats eyes and measured the electric signal released by cat brain neurons. The results indicated that the intensity of the electric signal was strongest when the light band was at some particular location and angle and different neurons show different preferences with different spatial locations and directions. Now, it has been shown that visual cortex has a multi-layer structure [7]. The signal from the retina first reaches the primary visual cortex, the V1 cortex. V1 cortical neurons are sensitive to some specific details of image signals. After the V1 cortex treatment, the signal is conducted to the V2 cortex. The V2 cortex represents the edge and contour information as a simple shape and is then processed by neurons in the V4 cortex, which is sensitive to colour information. Complex objects are ultimately represented in the inferior temporal cortex.

Shown in Figure 2.1, convolution neural network can be regarded as a simple simulation of the visual cortex mechanism [7]. It consists of multiple convolution layers, each containing multiple convolution kernels. All the pixels of input image will be scanned by these convolution kernels, the result output data is a set of matrices which called feature maps. The convolution layer in the front of the network captures the local and detailed information of the image, and there is a small receptive field, that is, each pixel of the output image uses only a small range of the input image. The receptive field of subsequent convolution layer is increased layer by layer to capture more complex and more abstract information. After the computations of multiple convolution layers, the abstract representations of the image at various scales are obtained. The first concept of convolution neural network was proposed by LeCun in 1989 [22]. The early success application of convolution neural network is handwritten characters image recognition [23 24]. Later, a deeper network with more layers AlexNet has been proposed and made a huge

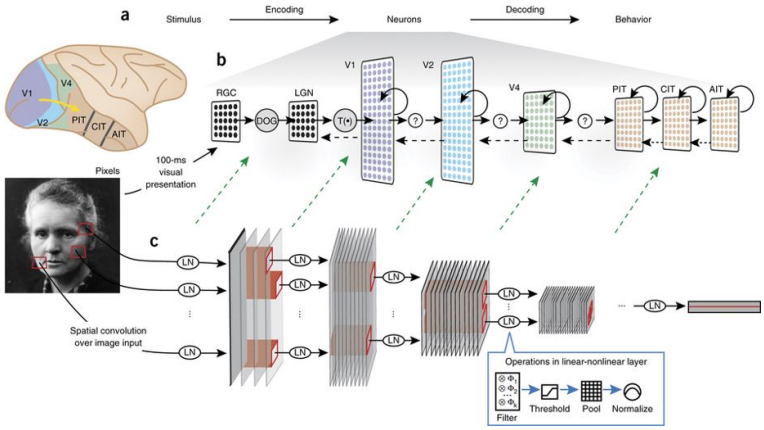

Figure 2.1: Simulation Image [7]

success in object recognizing [3]. Since then, the convolution neural network develops rapidly and makes a significant number of achievements.

2.2 Essence of Convolution

In order to demonstrate the function of convolution, some knowledge about Signal Processing is required. For example, a single sine wave always defined by two parameters, one is amplitude, the other one is frequency. A common way to describe a sine wave is to use a time-magnitude coordinate system. But it can also be transferred and described by a frequency-magnitude coordinate system which the two axes indicate the value of frequency and the intensity of amplitude. According to the Fourier Transform [25], a signal can be mapped from time domain to frequency domain shown in Figure 2.2. Any kind of complex signal can be described as an accumulation of a set of simple different frequency and amplitude sine signals.

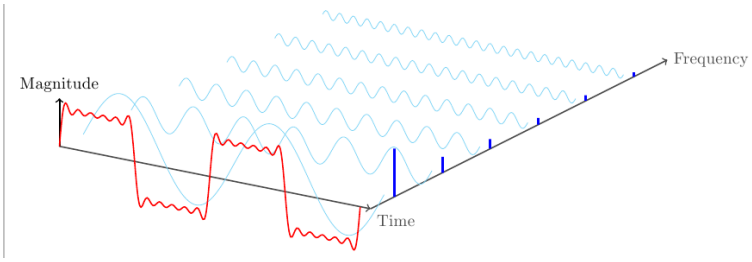

Figure 2.2: Fourier Transfer

The convolution between two functions is defined as:

$$x(t) * y(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau \tag{2.1}$$

Considering Fourier Transform, it can be shown that the convolution has a property which is:

$$x(t) * y(t) \leftrightarrow X(\omega)Y(\omega) \tag{2.2}$$

It indicates that when a convolution between two time-domain signals, the spectral representation of the result is the product of the spectral representations of two time-domain signals. In this case, a filter, containing some certain frequency component, convolves with a complex signal with some uncontained frequency component, the result signal will no longer contain this uncontained frequency component. Based on this filter theory, a low pass filter can filter high frequency signal while a high pass filter can filter low frequency signal.

Fourier Transform can be also used in two-dimensional function. It has the similar property with the one-dimensional Fourier Transform. In this case, a gray scale image can be described as a discrete 2D function $f(x, y)$, in which x, y determines the position of image pixels, $f(x, y)$ is the luminance value of pixel. Similarly, comparing with a 1D signal, the high frequency component of an image indicates that a pixel value changing rapidly area and the low frequency component of an image indicates that a pixel value changing slowly area. According to Figure 2.3, a high-pass filter makes a convolution with an image will save the luminance changing rapidly area. The convolution result of the image with low-pass filter will remove all the luminance changing rapidly area. Hence, when an image makes convolution with different filter, the result will be signals with different frequency component [26]. This can be regarded as the essence of convolution.

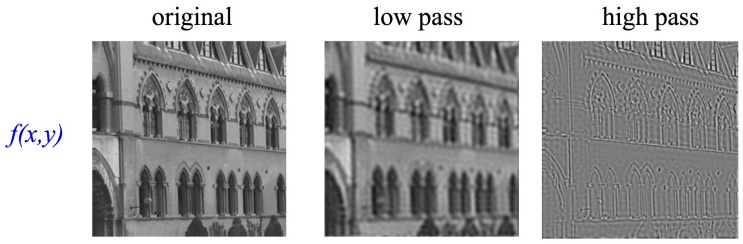

Figure 2.3: Different Pass Filter

2.3 Convolution layer in CNN

The convolution layers of a CNN model contain sets of convolution kernels which convolve with input image to extract different features. The convolution between the convolution kernel and the image shown in Figure 2.4, it can be described as a convolution kernels slides over the image matrix from top to bottom, from left to right, summing all the result of the elements of the convolution kernel matrix multiply the elements of the corresponding position elements of the matrix that the kernel covers on the image. The output matrix is the convolution result. After the convolution, the image size becomes smaller. Sometimes it is necessary to ensure the convolved result image has a same size compared to the original image, the original image will be padded, which adding zero to its periphery, before convolution. A simple pad which padding 1 is shown in Figure 2.5. In addition, in the process of sliding through the image pixels, the stride of sliding in the horizontal and vertical directions are usually set as 1, some other stride value can be also used. For the multi-channel inputs, one convolution kernel will convolve all the channels and sum the result matrices up to one matrix. Hence, each convolution kernel

will obtain a corresponding feature map. And the number of result matrices or feature maps is equal to the convolution kernels number.

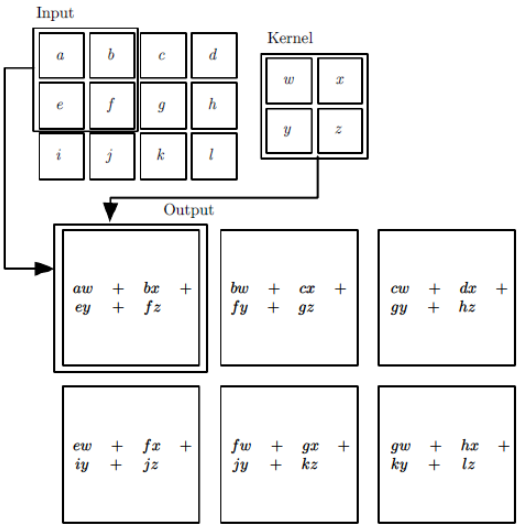

Figure 2.4: Convolution

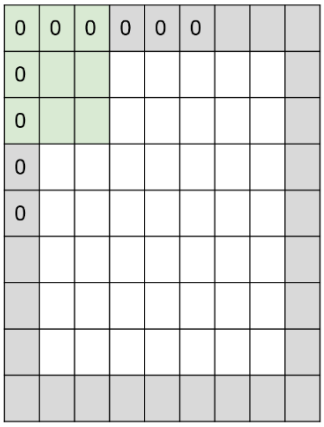

Figure 2.5: Padding

The convolution kernel matrix can be regarded as a two-dimensional filter. For example, the convolution kernel matrix of Sobel edge detection operator is:

$$\begin{bmatrix} -1 & 2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

This operator provides an information filter function to obtain edge information in one image. The convolution result of Sobel operator on an image is shown in Figure 2.6.

As can be seen from the Figure 2.6, the edge information of the image is highlighted by convolution. In addition to the Sobel operator, there are also Roberts, Prewitt operator which

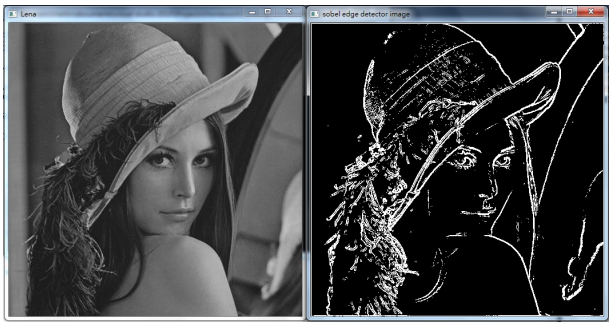

Figure 2.6: Sobel Operator

have same function and implement convolution in the same way, but with different convolution kernel matrices. With different convolution kernel, we can also extract more general image features. In image processing, the values of convolution kernel matrixes are usually designed by human. In CNN, a machine learning method involved to automatically generate convolution kernels to extract different image features. Usually, one convolution layer of CNN contains more than one convolution kernel to increase the accuracy and efficiency of feature extraction. Hence, the output of convolution layer in CNN is a set of matrices indicates the different activated feature responses of input image.

2.4 Activation Function

The convolution is a linear function of input image and output filter responses. A neural network contain only convolution layers can only have good performance when simulating linear functions. While the feature extracted process could be more complex that using linear function could hardly simulate. Because of the above reasons, when design a neural network, a convolution layer usually followed by a non-linear activation function. In the other words, the output of neural network will no longer be a linear combination of inputs, it can approximate any function. By adding more non-linear layers, the neural network can fit even more complex function. So that the depth of neural network is meaningful, for a linear neural network, the depth of linear layer will have little influence on the performance of network. Figure 2.7 shows some different activation functions.

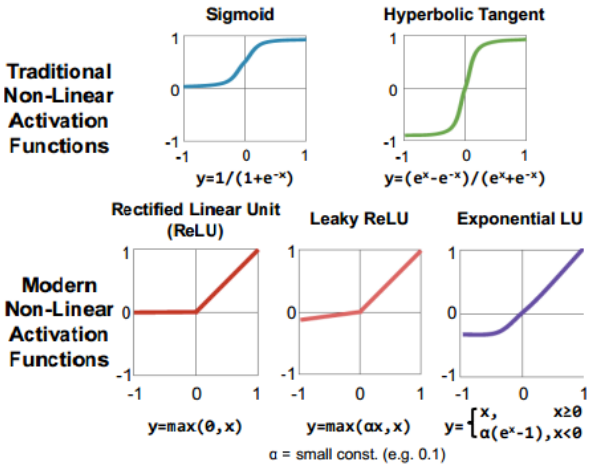

Figure 2.7: Activation Functions

2.5 Pooling Layer

Through the convolution operation, the dimensionality reduction and feature extraction of the input image has been completed. But the dimension of the feature image is still very high. High dimensionality not only causes computationally time consume, but also has more possibility to cause overfitting. Therefore, a down-sampling technique is introduced, also known as pooling.

Compared with the complexity of the convolutional layer, it could be easily found that the pooling layer has less computation. The input of pooling layer is usually the result matrices of feature maps pass through activated layer. The pooling process is to compress the sub-matrices contained in the input matrix. For example, for $2x2$ pooling, every $2 * 2$ sub-matrix which contain 4 parameters will turn to only one element. Considering 3x3 pooling process, then every nine parameters of $3x3$ size sub-matrix will turn to one element. Hence, the dimension of an input matrix will be reduced to a much smaller size. In addition to reducing the image size, another benefit of down-sampling is translational, rotational invariance, because the output value is calculated from a region of the image and is not sensitive to translation and rotation. With the increasing of the pooling size, more detail information could loss, hence, usually a CNN network selects a small pooling size but with many combinations of convolution layers and pooling layer.

In order to implement pooling process, usually there two types pooling, one is max pooling, the other one is average pooling. Max pooling is to select the maximum value of the pooling area as the pooling result. Average pooling is to compute the average value of whole pooling area and take the result as the pooling result.

The Figure 2.8 shows an example uses a pooling method that takes the maximum value. At the same time, the pooling of $2x2$is processing in other area of feature map which the pooling stride is 2. The pooling stride indicate that the distance between two pooling process.

First, the red $2x2$ area is pooled. Since the maximum value of the $2x2$ area is 6. The pooling result is 6 at the corresponding position of the output matrix. Since the stride is 2, the position

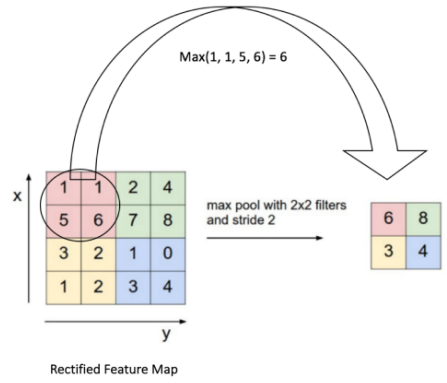

Figure 2.8: Pooling

is moved to the green position for pooling and output. The maximum value is 8. In the same way, the output values of the yellow and blue regions can be obtained. Eventually, our input $4x4$ matrix is compressed and becomes a $2x2$ matrix after pooling.

2.6 Fully Connected Layer

Typically, fully connected layers are the last structure of convolution neural network. This layer is used to connect all the feature responses of last convolution layer. The fully connected layer will sum all the parameters and allocate weight for these parameters to fit the training dataset. For example, the LeNet5 [16] shown in Figure 2.9 is designed to identify handwritten digits. After extracting the image features, fully connected layers will connect all the features and classify them into ten elements that indicate the handwritten digit of the image.

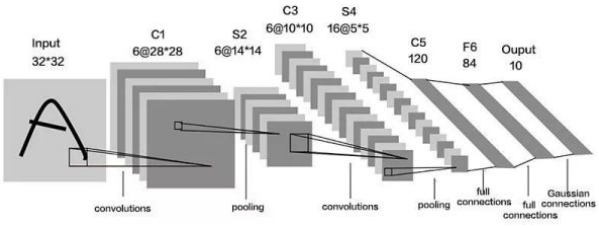

Figure 2.9: LeNet5 [16]

2.7 Overview of CNN

A Convolution Neural Network usually contains several multi-layers convolution layers, every convolution sub-layer will be followed by an activation function. At the end of convolution layer there is usually a pooling layer. The last layer could be one or more fully connected layer. The purpose of multi-layer convolution is that the features learned by one layer of convolution are often local, and the higher the number of layers, the learned features are more global.

Here, we can give a summary of the process of CNN. An image input into a CNN model, the early convolution layer will extract small features from the input image. In the next convolution layer, the size of the extract features increased. After feature extracted, a pooling layer will be used to reduce the size of feature responses to remove unnecessary information. With the increasing number of convolution layers, more complex features will be extracted. In the end fully connected layer, all these complex features will be combined together to give the object recognition result.

2.8 VGG-19 network

VGGNet is a deep convolution neural network which proposed by researchers from the Visual Geometry Group and Google DeepMind [15]. This network test different depth of network to compare the accuracy of neural network. By construct convolution layer with 3*3 convolution kernel and followed by a 2*2 size max pooling layers repeatedly, VGGNet gives several depth of convolution neural networks contained 16 19 convolution layers. Compared with the previous good performance network architecture, VGGNet has obviously decreased the error rate and won a second place in classification task and the first place in the localization task in the ILSVRC 2014 competition [15]. Furthermore, VGGNet provides a very scalable feature and it has a very good performance when migrate to other image dataset, VGGNet is designed by a very simple structure, and the whole network uses a same size of convolution kernel which is 3*3 and same maximum pool size (2*2). So far, VGGNet is still a widely used convolution neural network model to extract image features. Model parameters after VGGNet training are open sourced on their official website and can be used to retrain on specific image classification tasks (equivalent to providing very good initialization weights), so they are used in many places.

Since the convolution neural network with more convolution layers may have good performance on extracting features, in this project, a VGG-19 model will be used to extract image features. The structure of VGG-19 is shown in Figure 2.10:

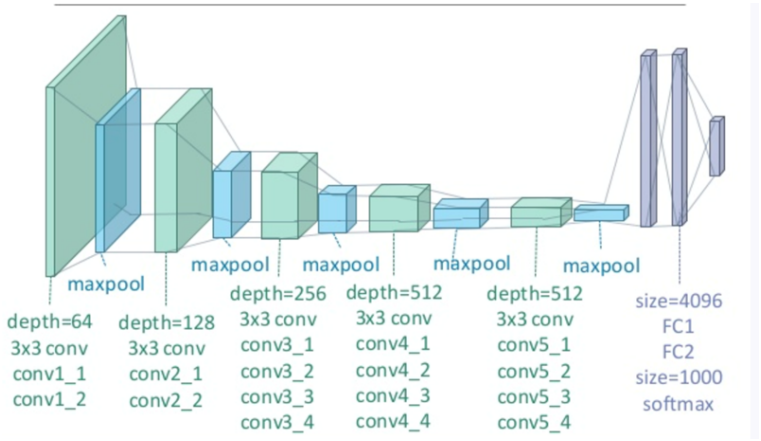

Figure 2.10: VGG-19 [15]

Chapter 3

Content and Style Generations

3.1 Gradient Descent

Given a defined loss function $F_{(\vec{x})}$, the gradient descent method is used to modifying the parameter vector $\vec{x}$ to find a minimized result of $F_{(\vec{x})}$. Theoretically, the gradient $\bigtriangledown_{(\vec{x})} F_{(\vec{x})}$ is defined as the derivative of loss function $F_{(\vec{x})}$ for vector $\vec{x}$, this can be viewed as slope of function $F_{(\vec{x})}$ at the point $\vec{x}$ under a multi-dimension coordinate system. Hence, we can modify $\vec{x}$ by small number determined by the gradient until the gradient is near zero (critical point). At this point the value of loss function cannot change which means it reaches its minimum. Then the vector $\vec{x}$ is the optimized result of gradient descent.

The modify step is defined and update by:

$$\vec{x} := \vec{x} - \epsilon \bigtriangledown_{(\vec{x})} F_{(\vec{x})}$$

In which ϵ is the learning rate to determine learning speed. For a particular loss function, when the learning rate is set an over large value, then the $\vec{x}$ would be modified too much to always pass the critical point. And when the learning rate is selected a very small value. The convergence speed of gradient descent could be too slow to reach the critical point in a reasonable time. In practice, there are many methods to choose a suitable learning rate, some of them are contained in machine learning framework which can be automatically selected.

3.2 Content Generation

3.2.1 Overview

Style transfer contains two parts, one is content generations to build a content representation and use a gradient method to generate an image that has the same content representation to the target content image. The other one is style generations to find a style representation and use the same method with content generation to find an image has the same style representation with the target style image. With the assistance of VGG-19 model, the necessary features of image can be identified and extracted by the convolution layers. And the results of extracted

features in VGG model are different feature maps. These feature maps can be regarded as the content representations. With the increasing of the convolution layers, more explicit information of image would be extracted [11]. In the other words, the early convolution layers only extract very precisely small features from image, when the input image passes more filters, the output feature maps will be the responses of more complex features with bigger size of image but also lost many small details. Therefore, based on different convolution layers, the content representation can be different. Using a gradient descent method, it can be achieved to find an image has the same content representation to the target content image which means they have same content.

3.2.2 Method

An initialized image with random noise x is input into VGG-19 model, the output feature maps of a convolution layer l has a corresponding mathematic expression which is a matrix we name it as F_l. The number of convolution kernels contained in layer l is an integer N_l which equals to the number of output feature maps in this layer. Here we resize the feature map to a one row matrix for easy computing. These feature maps share a same size M_l because in VGG network, all convolution kernel has a same size of $3 * 3$, hence the size of convolution results is same, which equals to the width multiplies the height of the feature map. But the size of feature maps will be different in different convolution layers because there is no padding in VGGnet, after convolution the size of input will be smaller. Hence the size of matrix F_l can be easily computed which is $N_l \times M_l$, and the element F_{ij}^l of the matrix F_l means the convolved result with i_{th} convolution kernel at the position j of layer l. Similarly, the matrix contains output of target content image p in VGG model layer l can be defined as P_l. In order to generate a result image p which has a same content representation compared to the target image p, a gradient descent can be applied to minimize the distance between the elements of the two output matrices F_l and P_l. The gradient descent loss function of two filter responses can be defined by squared-error [17]:

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

Using gradient descent method modifies elements of image x to reduce the value of loss functions which means reduce the distance between the elements F_{ij}^l and P_{ij}^l at same position of two different filter response matrices F_l and P_l. The result x will have the same filter response compared to target content image p in the same layer L. Hence, an optimized image x is the content generation result from content representation of layer L.

3.3 Style Generation

3.3.1 Overview

The output of convolution layer is a set of feature maps which is the convolved result of a set of convolution kernels. Hence, a statistic measurement of correlations between different

feature maps can be computed and stored in a feature space which built to indicate the texture representation. The element of the feature map usually saves the location information because the convolution process is orderly sliding from the first element to the last element. But the image texture should be a global feature representation of image. So, a summary statistic measurement which sums all the coefficient of different feature responses at some position of different feature maps will be used to remove the localized information. As it has been noted, the receptive field size increases with the number of convolution layers that the input image passes through. Every new element of the result matrix is a presentation of the upper $3 * 3$ pixels (In VGG network every convolution kernel has a same size $3 * 3$), with the number of convolution layer increasing, one simple element will contain more pixel information of the original image. Therefore, if the feature spaces contain different summary statistic measurement of coefficient between feature maps from different convolution layers, a multi-scale and global texture representation will be obtained. Similar to content generation, using a gradient descent method, it can be achieved to find an image has the same style representation to the target style image which means they have same style.

3.3.2 Method

Similar to content generation, a gradient descent method can be also used to generate an optimized stylized image x. Here we give the definition of Gram Matrix, an initialized image with random noise x has a corresponding element F^l_{ik} at k_{th} position of i_{th} feature map in layer l. For the other feature response at the same position, the corresponding element is F^l_{jk} at k_{th} position of j_{th} feature map in layer l. The coefficient of two feature responses i, j at position k is defined as the product $F^l_{ik}F^l_{jk}$, the summary statistic of two feature responses which is feature map i, j is computed by summing all the coefficient of two feature responses at different position. And the Gram Matrix contains all these summary statistics between all the feature responses. The element of Gram Matrix of input image x is defined as [17]:

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}$$

Similarly, the target style image a has a corresponding Gram Matrix, the element is defined as:

$$A^l_{ij} = \sum_k S^l_{ik} S^l_{jk}$$

Since the feature maps have been resized into one row matrix which length is M_l equals to width multiplies the height of the feature map in layer L. And the number of feature maps is N_l, all the output feature maps are stored in a $N_l \times M_l$ matrix F_l. Hence, the Gram Matrix G_l can be simply computed by the product of matrix F_l and the transpose matrix F^T_l:

$$G_l = F_l F^T_l$$

The size of Gram Matrix is $N_l \times N_l$. In convolution layer l, a Gram Matrix will compute the statistic coefficient between feature maps which can remove the localized information. The

different compared with content generation is that we no longer compute the loss between the filter response of two input images. Instead, we computer the distance between elements of two Gram Matrix corresponding to initialized image x and target style image a. The loss between the gram matrices of two input images in l layer is defined by mean-squared distance [17]:

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

In order to save multi-scale texture information, several gram matrices of different layers will be considered in loss functions of gradient descent. Hence, the total loss can be defined as [17]:

$$L_{style}(a, x) = \sum_{l=0}^{L} \omega_l E_l$$

And ω_l is the weight determines that the weight of each layers Gram Matrix loss.

3.4 Style Transfer

In order to implement style transfer, an image must be generated that has the same output feature maps compared with the target content image at certain layer and also has the same gram matrices calculated by feature maps of involved layers. A joint gradient descent method is applied to reduce not only style loss but also content loss. In this method, an initial image x, a target content image p, and a target style image a have been defined. And the loss function that needs to be minimized is:

$$L_{total}(x, p, a) = \omega_{content} L_{content}(p, x) + \omega_{style} L_{style}(a, x)$$

Which $\omega_{content}$, ω_{style} determine the weight of style and content in generated image.

3.5 Gram Matrix

3.5.1 Summary Statistic

The element of Gram Matrix is computed by the inner product of two vectorized filter responses or two feature maps [17]. An element of feature map is indicated the filter response of a corresponding convolved 3×3 matrix in the last feature map, so this feature response can be viewed as containing location information. In a certain layer, different filters generate different responses with a same size, while elements with the same position in different feature maps contains the different feature information of the same 3×3 matrix in last feature map. So, the product of two elements can indicate the localized coefficient between two feature responses at a certain position, while the element of Gram Matrix is a summary statistic computed by the correlations between different filter responses [11]. The Gram Matrix will not contain spatial information because the element is the summation of all the coefficient of filter responses at different positions in the feature map while the position of element in feature map indicate the

location information. According to the magnitude of element in Gram Matrix, the possibilities of different feature responses appears in feature maps and the relations of different features appeared in the image can be measured. In the other word, the Gram Matrix indicates the distribution of features in image which represents the texture of image. While minimizing the distance between two Gram Matrices of two image can lead to a style transfer process.

3.5.2 Domain Adaptation and Maximum Mean Discrepancy

Domain adaptation is an aspect of transfer learning which is sub-research area of machine learning [19]. Typically, domain adaptation is used to apply a model trained by a source domain usually contains a label with a certain data to a different target dataset without containing any labels [21]. Usually a machine learning model is training by a labeled domain, this will help the model to understand the learning target. But this model could fail when testing an unlabeled different target domain, even if the intrinsic qualities of two domains are similar. For example, a face recognition model training from an Asian people dataset may have problem when recognizing European faces. The goal of domain adaptation is to adjust the distinguish of the source trained domain and the target test domain to make the original model perform well when used in another domain. Generally, domain adaptation is achieved by measuring and minimizing the distribution difference of two adaptation dataset. Here we introduce a popular choice for domain adaptation techniques which is the Maximum Mean Discrepancy (MMD) [4], a common discrepancy metric, which measure the mean difference of domain samples in a Reproducing Kernel Hilbert Space [18,21].

Given two sample datasets $X = \{x_i\}_{(i=1)}^{n}$ and $Y = \{y_j\}_{(j=1)}^{n}$ with two different distribution p and q, when input these two datasets into a machine learning model, the output of each sample can be simply present as $\phi(x_i)$ and $\phi(y_i)$, if the mean discrepancy between two output equals to zero, the distributions of two dataset can be regarded as the same or $p = q$. here the squared MMD is given by:

$$
\begin{aligned}
& \mathrm{MMD}^2[X, Y] \\
= \; & \|\mathbf{E}_x[\phi(\mathbf{x})] - \mathbf{E}_y[\phi(\mathbf{y})]\|^2 \\
= \; & \|\frac{1}{n}\sum_{i=1}^{n}\phi(\mathbf{x}_i) - \frac{1}{m}\sum_{j=1}^{m}\phi(\mathbf{y}_j)\|^2 \\
= \; & \frac{1}{n^2}\sum_{i=1}^{n}\sum_{i'=1}^{n}\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_{i'}) + \frac{1}{m^2}\sum_{j=1}^{m}\sum_{j'=1}^{m}\phi(\mathbf{y}_j)^T\phi(\mathbf{y}_{j'}) \\
& - \frac{2}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m}\phi(\mathbf{x}_i)^T\phi(\mathbf{y}_j),
\end{aligned}
$$

And using an associated kernel function $k(x, y)$ to replace the inner product between vectorized output, the result formula will be:

$$\begin{aligned}&\mathrm{MMD}^2[X,Y]\\ =\ &\frac{1}{n^2}\sum_{i=1}^{n}\sum_{i'=1}^{n}k(\mathbf{x}_i,\mathbf{x}_{i'})+\frac{1}{m^2}\sum_{j=1}^{m}\sum_{j'=1}^{m}k(\mathbf{y}_j,\mathbf{y}_{j'})\\ &-\frac{2}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m}k(\mathbf{x}_i,\mathbf{y}_j).\end{aligned}$$

The kernel function intrinsically defines mapping the original two samples to a higher multi-dimensional feature space [21].

According to our style loss function at l_th layer [17]:

$$L^l_{style}=\frac{1}{4N_l^2M_l^2}\sum_{i,j}\left(G^l_{ij}-A^l_{ij}\right)^2$$

The element in the Gram Matrix of target style image is defined as the inner product between two vectorized feature maps in layer l [17]:

$$A^l_{ij}=\sum_k S^l_{ik}S^l_{jk}$$

Similar to the optimized image:

$$G^l_{ij}=\sum_k F^l_{ik}F^l_{jk}$$

Hence, we can transform the loss function [21]:

$$L^l_{style}=\frac{1}{4N_l^2M_l^2}\sum_{i=1}^{N_l}\sum_{j=1}^{N_l}(\sum_{k=1}^{M_l}F^l_{ik}F^l_{jk}-\sum_{k=1}^{M_l}S^l_{ik}S^l_{jk})^2$$

$$=\frac{1}{4N_l^2M_l^2}\sum_{i=1}^{N_l}\sum_{j=1}^{N_l}((\sum_{k=1}^{M_l}F^l_{ik}F^l_{jk})^2+(\sum_{k=1}^{M_l}S^l_{ik}S^l_{jk})^2-2(\sum_{k=1}^{M_l}F^l_{ik}F^l_{jk})(\sum_{k=1}^{M_l}S^l_{ik}S^l_{jk}))$$

$$\sim$$

$$=\frac{1}{4N_l^2M_l^2}\sum_{k_1=1}^{M_l}\sum_{k_2=1}^{M_l}((\sum_{i=1}^{N_l}F^l_{ik_1}F^l_{jk_2})^2+(\sum_{i=1}^{N_l}S^l_{ik_1}S^l_{jk_2})^2-2(\sum_{i=1}^{N_l}F^l_{ik_1}F^l_{jk_2})^2)$$

$$=\frac{1}{4N_l^2M_l^2}\sum_{k_1=1}^{M_l}\sum_{k_2=1}^{M_l}((f^l_{.k_1}f^{lT}_{.k_2})^2+(S^l_{.k_1}S^{lT}_{.k_2})^2-2(f^l_{.k_1}S^{lT}_{.k_2})^2)$$

This transformation means we change the order to compute the loss function. Instead of computing the inner product between different feature maps which are the row of feature maps stored matrix, the new formula computes the inner product between the columns of feature maps stored matrix. Then, we use a second order degree polynomial kernel $k(x,y)=(x^Ty)^2$ to replace the components in the formula, the style loss function can transfer to a square MMD representation [21]:

$$\begin{aligned}\mathcal{L}_{style}^{l} =& \frac{1}{4N_l^2 M_l^2} \sum_{k_1=1}^{M_l} \sum_{k_2=1}^{M_l} \Big(k(\mathbf{f}_{\cdot k_1}^{l}, \mathbf{f}_{\cdot k_2}^{l}) \\ &+ k(\mathbf{s}_{\cdot k_1}^{l}, \mathbf{s}_{\cdot k_2}^{l}) - 2k(\mathbf{f}_{\cdot k_1}^{l}, \mathbf{s}_{\cdot k_2}^{l}) \Big) \\ =& \frac{1}{4N_l^2} \mathrm{MMD}^2[\mathcal{F}^l, \mathcal{S}^l],\end{aligned}$$

In this representation, for a particular input image, f, s is the different feature responses at same position of different feature maps which contains local information. F, S means the summary statistic measurement of correlations between same filter response at different position. Using domain adaptation to demonstrate this formula, F, S can be viewed as the domain output from the CNN model. After summing all the local feature information and taking average, the features difference is removed, hence the local distribution information of whole features is obtained.

In summary, refer to the definition of Gram Matrix, the element computation comes from two steps, first it computes the coefficient of different filter responses at same position in feature map which contains location information, then sum all of them to remove location information. Hence, the final Gram Matrix can be used to indicate the correlations between different features in the image which is the distribution of features in the whole image. While according to the demonstrate by domain adaptation, we first compute the correlations of same filter response at different positions. And then sum them up, the result matrix will indicate the correlations between total activation of all convolution kernels at different position in feature maps which could be also achieved style representation by shown the distribution of whole feature responses in a pixel of image. Although the demonstrations of style representation from Gram Matrix and domain adaptation are not equal, the style loss function minimizing process can still be simply concluded as an align process to modify the distribution of content image and the style image [21].

Chapter 4

Implementation

4.1 TensorFlow in Python

Nowadays, a growing trend can be witnessed in the research field of deep learning. There are various open source deep learning frameworks come into being, such as TensorFlow, Caffe, Keras, CNTK, Torch7, MXNet, Leaf, Theano, DeepLearning4, Lasagne, Neon and so on [28]. However, among these different frameworks, TensorFlow has made a great achievement on attracting public attentions and users. The good performance when using TensorFlow handling different practical problem could be the reason for its popular. Actually, TensorFlow has many advantages such as the simplicity of code designing neural network structures, the efficiency of distributed deep learning algorithms, and the ease of deployment.

TensorFlow is an efficient high-level machine learning framework that provide users a easily design experience when building their own neural network without having to learn and write C++ or CUDA code to achieve a higher efficient implementation. And TensorFlow contains many advantages of other framework. It supports automatic derivation as well as Theano, and the user does not need to solve the gradient by back-propagation. The core code is written by C++ like Caffe. Using C++ simplifies the complexity of online deployment, and allows devices with tight memory and CPU resources to run complex models which would waste source and implement with lower efficiency if using Python. This allows users to deploy models in C++ on a machine with better hardware configuration. SWIG supports interfaces of various languages for C/C++ code, so the interfaces of other scripting languages can be easily accessed into TensorFlow through SWIG in the future. The most widely used scripting language for TensorFlow could be Python. However, there is a problem with efficiency when using Python. Every small batch need to be fed into the network from Python. This process may be brought a bad influence relatively large delay when the amount of data in mini-batch is small or the operation time is short.

The major mainstream frameworks basically support Python, which currently dominates the field of scientific computing and data mining. Although there is competition pressure from R, Julia and other languages, but various libraries of Python are too perfect and hardly

to be beat. Python has a widely usage in Web development, data visualization, data pre-processing, database connection, crawler and so on omnipotent, and it has a perfect ecological environment. Only refer to the data mining tools chain, there are NumPy, SciPy, Pandas, Scikit-learn, XGBoost and other components in Python. With these components, data acquisition and pre-processing by Python are very convenient, and the subsequent model training stage can be perfectly linked with TensorFlow and other Python-based deep learning framework. Furthermore, compared to R, Julia and other coding language, Python provide an obviously convenient developing environment. It could be the easiest learning language for beginners.

TensorFlow has high-quality code at the product level, with the strong development and maintenance capabilities of Google, and the overall architecture design is also excellent [26]. TensorFlow is more mature and complete than the same Python-based rival Theano. Furthermore, the documentation of the framework could be extremely completed and helpful. Since the TensorFlow has attracted a widely range of focus and a significant number of users, the developing community could also provide an efficient online question answer ability for users. And it could be easy for new users to find a related tutorial on the internet.

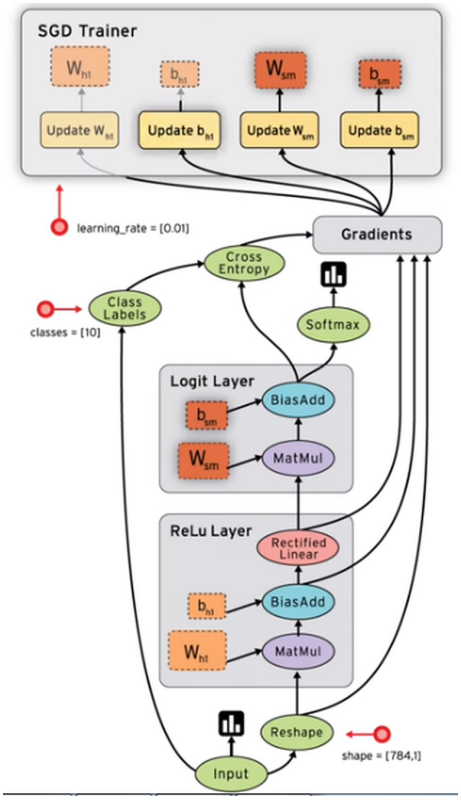

Figure 4.1: Architecture of TensorFlow [28]

4.2 Code Process Construction of TensorFlow

TensorFlow is a computing system based on graph (Graph). The operation points of a graph are composed of operations, and the operation points of the graph are linked by tensors as edges. So, the computation process of TensorFlow is a Tensor flow graph. When an operation added, the operation will not be executed immediately. TensorFlow will wait until all operations are added, and then optimizes the graph to determine how to perform the calculation. Tensor is a variable or a constant in the code. All variables need to be initialized before starting to perform graph calculations.

The graph of TensorFlow must be calculated in a session (Session). Session provides an environment for Operation execution and Tensor evaluation. Considering the low performance of Python, we try to use non-Python code when we execute numerical computing, such as pre-compiled C code like NumPy for matrix operations. The time it takes to switch between a Python internal computing environment and an external computing environment, such as NumPy, is called overhead cost. For a simple operation, such as a matrix operation, switching from Python to Numpy, Numpy gets the result, and then cuts back to Python from Numpy, the cost is much lower than the cost of doing the same kind of operation purely inside Python. However, a complex numerical operation is composed of several basic operations. If each basic operation switches the environment once, overhead cost cannot be ignored. To reduce round-trip environment switching, TensorFlow does this by defining the entire Graph in Python and then running the entire Graph outside of Python. Therefore, the code process structure of TensorFlow is separated by two stages [28].

4.3 Architectural of Implementation

This implementation is operated under Mac OS by TensorFlow with Python. The implementation of this paper is based on TensorFlow, Python with Mac OS. Analyzing the style transfer process, the architectural of the implementation should be an optimized loop. The goal of this loop is to minimize the total loss function which contains style and content generations. And since the TensorFlow does not contain VGG-19 net, a preload method will be included to save all the useful information from the VGG-19 official network file to build our convolution network in TensorFlow. After building our VGG-19 model, now the initial, content and style images can be input and extract the output matrices from the model.

In order to have a convenient operate environment, an ArgumentParser will be required to change the arguments easily when run the program under a Linux operate system. Normally, TensorFlow will be activated in a virtually environment under Linux. Hence, in the main loop file, an ArgumentParser will need to be define with several necessary arguments such as change the input and output file, change the weights between content and style generation. So, the Python program can be operated in terminal (Mac OS).

After all the needed parameters have been defined, the main loop function which including read network, read input image, resize and preprocess image for the requirement of VGG-19 network, style transfer process and save result image. Here, the most important function should be the style transfer function, which based on the project analysis in the previous chapters.

First, the output of initial, style and content images will be computed by the built network, and the output feature maps of a certain layer will be saved. Second the content loss will be computed which will only compute the distance between the elements in feature maps of initial image and content image. Third, a feature space will be created to save the correlations between feature maps which compute the Gram Matrix. And based on this feature space, the style loss function will be defined. Later, a joint function will be defined which contain content loss and style loss and their weights. In the end, an optimized method needed to be chosen to minimized the total loss function.

4.4 VGG pretrained-network

47x1 Layer array with layers:

1	'input'	Image Input	224x224x3 images with 'zerocenter' normalization
2	'conv1_1'	Convolution	64 3x3x3 convolutions with stride [1 1] and padding [1 1 1 1]
3	'relu1_1'	ReLU	ReLU
4	'conv1_2'	Convolution	64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
5	'relu1_2'	ReLU	ReLU
6	'pool1'	Max Pooling	2x2 max pooling with stride [2 2] and padding [0 0 0 0]
7	'conv2_1'	Convolution	128 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
8	'relu2_1'	ReLU	ReLU
9	'conv2_2'	Convolution	128 3x3x128 convolutions with stride [1 1] and padding [1 1 1 1]
10	'relu2_2'	ReLU	ReLU
11	'pool2'	Max Pooling	2x2 max pooling with stride [2 2] and padding [0 0 0 0]
12	'conv3_1'	Convolution	256 3x3x128 convolutions with stride [1 1] and padding [1 1 1 1]
13	'relu3_1'	ReLU	ReLU
14	'conv3_2'	Convolution	256 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
15	'relu3_2'	ReLU	ReLU
16	'conv3_3'	Convolution	256 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
17	'relu3_3'	ReLU	ReLU
18	'conv3_4'	Convolution	256 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
19	'relu3_4'	ReLU	ReLU
20	'pool3'	Max Pooling	2x2 max pooling with stride [2 2] and padding [0 0 0 0]
21	'conv4_1'	Convolution	512 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
22	'relu4_1'	ReLU	ReLU
23	'conv4_2'	Convolution	512 3x3x512 convolutions with stride [1 1] and padding [1 1 1 1]
24	'relu4_2'	ReLU	ReLU
25	'conv4_3'	Convolution	512 3x3x512 convolutions with stride [1 1] and padding [1 1 1 1]
26	'relu4_3'	ReLU	ReLU
27	'conv4_4'	Convolution	512 3x3x512 convolutions with stride [1 1] and padding [1 1 1 1]
28	'relu4_4'	ReLU	ReLU
29	'pool4'	Max Pooling	2x2 max pooling with stride [2 2] and padding [0 0 0 0]
30	'conv5_1'	Convolution	512 3x3x512 convolutions with stride [1 1] and padding [1 1 1 1]
31	'relu5_1'	ReLU	ReLU
32	'conv5_2'	Convolution	512 3x3x512 convolutions with stride [1 1] and padding [1 1 1 1]
33	'relu5_2'	ReLU	ReLU
34	'conv5_3'	Convolution	512 3x3x512 convolutions with stride [1 1] and padding [1 1 1 1]
35	'relu5_3'	ReLU	ReLU
36	'conv5_4'	Convolution	512 3x3x512 convolutions with stride [1 1] and padding [1 1 1 1]
37	'relu5_4'	ReLU	ReLU
38	'pool5'	Max Pooling	2x2 max pooling with stride [2 2] and padding [0 0 0 0]
39	'fc6'	Fully Connected	4096 fully connected layer
40	'relu6'	ReLU	ReLU
41	'drop6'	Dropout	50% dropout
42	'fc7'	Fully Connected	4096 fully connected layer
43	'relu7'	ReLU	ReLU
44	'drop7'	Dropout	50% dropout
45	'fc8'	Fully Connected	1000 fully connected layer
46	'prob'	Softmax	softmax
47	'output'	Classification Output	crossentropyex with 'tench' and 999 other classes

Figure 4.2: VGG-19 layers [30]

Figure 4.2 shows all the layers in VGG-19 model, and also indicates the details and types of each layer. The fully connected layer usually used to connect all the extracted features and give the possibility of the recognized result. So, in this work, the fully connected layer will no longer be used cause only the feature maps will be needed.

The *relu* layer is the activation function in Convolution Neural Network, this layer usually provides a nonlinear character to the whole network. In VGG-19, the activation function is ReLU, the advantages of ReLU is that it can reduce the number of parameters to save computing time and avoid overfitting. Furthermore, the ReLU function is easy to compute derivatives. When training model, it will accelerate the training speed. In this project, the value of the initial image will be renewed by compute the derivatives of the loss function. Hence, the ReLU can also be viewed as a good choice.

The VGG-19 pretrained model is downloaded from the official website [27] which is a Matlab file format, typically, a component *scipy.io* can be used to read Matlab file in Python. Since the full connected layers are not necessary, the built network will only include layers from $conv1_1$ to $relu5_4$. For the purpose to build VGG-19 network in TensorFlow, the convolution kernels and bias in necessary layer are needed which will be saved after loading the Matlab VGG-19 file. In addition, the image which will input into VGG network need to be preprocessed. The preprocess of input image is each pixel minus the average pixel value of training dataset of VGG network. Hence, all the useful information has been obtained, the TensorFlow will provide all the necessary function for build a convolution network.

```
#scipy.io to read matlab file
data = scipy.io.loadmat(data_path)
```

According to the structure of Matlab VGG-19 file, the parameters in convolution kernel are saved in different position, therefore, when building network, the kernel matrices need to be transposed. The convolution, pooling and activation layers defined in TensorFlow is as following:

```
#define a cnovolution layer which return the convolution result
tf.nn.conv2d(input, filter, strides, padding, use_cudnn_on_gpu = None,
                                          data_format = None, name = None)

#define a pooling layer which return pooling result
tf.nn.avg/max_pool(input, ksize=(1, 2, 2, 1), strides=(1, 2, 2, 1),padding='SAME
                                          ')

#define a RELU activation layer
output = tf.nn.relu(input)
```

[28]

The shape of input image should be a tensor with four parameters, *batch* which indicates the number of images will be input at once, in_height which indicates the height of input image, in_weight indicates the width of input image and $in_channel$ is the channels of image. $filter$ is the convolution kernel which also contains four parameter which indicate the height, weight input image channels and number of convolution kernel. The output of this convolution layer is a tensor with four parameters, *batch*, *height*, *weight* and $out_channels$ which equal to the number of convolution kernel.

The target net is a Python dictionary which contains all the output feature maps which related to its particular layer.

4.5 Operate Environment

Since TensorFlow is running in a virtual environment which will be activated by Linux, most TensorFlow with Python project need to be operated from terminal (Mac OS). Hence, a tool provided by Python to receive command line information should be included in the program. In Python, build an ArgumentParser is a good choice when handling the requirement to running Python program under Linux. ArgumentParser provided several useful functions to give enough information for users. By adding new argument for parser, users can change the argument in the Linux operation environment before running Python program. Here is a example for adding argument to parser.

```
#create a new argument parser
parser = ArgumentParser()
#add a new argument
Parser.add_argument(
#option parameters
   --name
#return parameter name to Python program
dest =   name
#help information
help =  information
#parameter shown in help information
metavar =   name in information
#if this parameter must be assigned
required = True
#type of
type = int/ float
```

In this project, the beginning assumes of the arguments contained in argument parser should include input images file path, output file path and the weights of content and style generations. But during the implement process, more arguments would be added to the parser. With the argument parser, it could be easily achieved to assign statement to argument conveniently. Users can change the content and style image without understanding the original Python code. And it could be necessary for its future application development. Figure 4.3 shows the operation interface of this project, after installing TensorFlow framework and activating the virtual environment, the operating environment is under terminal of Mac OS.

Input help command to see all the available arguments.

```
python style_transfer.py --help
```

The return arguments and descriptions are shown in Figure 4.4 The required arguments are path of input images and save path of output image. Other arguments have a default value which will give a proper result of style transferred image with a middle weight between content and style representation. Figure 4.5 shows when only changing iterations without changing other default arguments, the program running process. Which the running command is:

```
python style_transfer.py --content examples/content2.jpg --styles examples/
                                        starry-night.jpg --output result.jpg --
                                        iterations 500
```

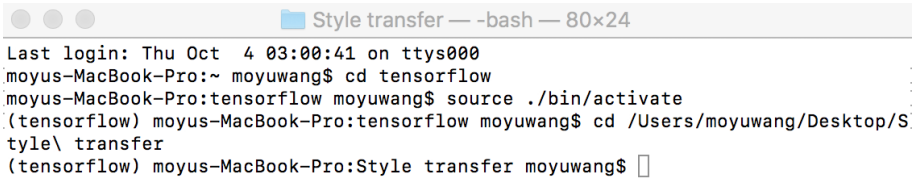

Figure 4.3: Activated Environment

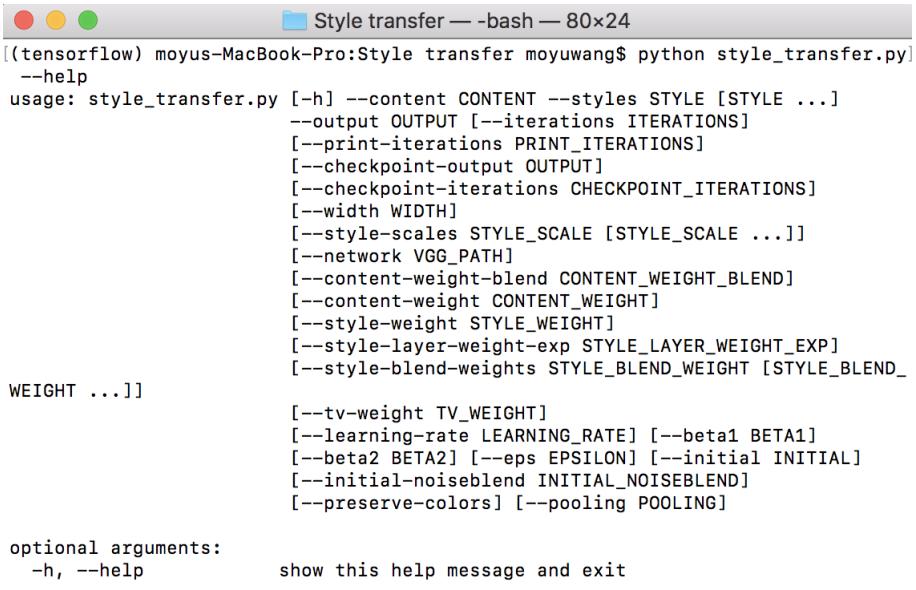

Figure 4.4: Arguments description

```
Style transfer — python style_transfer.py --content examples/content2.jpg --st...
(tensorflow) moyus-MacBook-Pro:Style transfer moyuwang$ python style_transfer.py
 --content examples/content2.jpg --styles examples/starry-night.jpg --output res
ult.jpg --iterations 500
/System/Library/Frameworks/Python.framework/Versions/2.7/Extras/lib/python/scipy
/misc/pilutil.py:413: FutureWarning: Conversion of the second argument of issubd
type from `int` to `np.signedinteger` is deprecated. In future, it will be treat
ed as `np.int64 == np.dtype(int).type`.
  if issubdtype(ts,int):
/System/Library/Frameworks/Python.framework/Versions/2.7/Extras/lib/python/scipy
/misc/pilutil.py:415: FutureWarning: Conversion of the second argument of issubd
type from `float` to `np.floating` is deprecated. In future, it will be treated
as `np.float64 == np.dtype(float).type`.
  elif issubdtype(type(size),float):
2018-10-04 14:26:25.832821: I tensorflow/core/platform/cpu_feature_guard.cc:140]
 Your CPU supports instructions that this TensorFlow binary was not compiled to
use: AVX2 FMA
Optimization started...
Iteration    1/ 500
Iteration    2/ 500
Iteration    3/ 500
Iteration    4/ 500
Iteration    5/ 500
Iteration    6/ 500
```

Figure 4.5: Simple Approach

After the optimizing process ended, the terminal returns the result loss, and the result image is saved in the input path.

4.6 Feed-Forward Model

As early defined the initial image x, the target content image p, and the target style image a, the loss function needs to be minimized is:

$$L_{total}(x, p, a) = \omega_{content} L_{content}(p, x) + \omega_{style} L_{style}(a, x)$$

Which $\omega_{content}$, ω_{style} determine the weight of style and content in generated image.

In order to build this loss function, first the content feature responses and style feature responses will be obtained from the convolution neural network which has been built. The input image will go through all the convolution layers and the output feature maps will be saved in a Python dictionary. Hence, feature maps of any layers can be obtained from this dictionary. Which later the feature maps of different layers contained in the computation of minimize the loss function will be changed to evaluate the different optimized results. The process of getting output feature maps form first convolution layer to the final convolution layer is called feed-forward. So, a graph and session created to compute the net process with input style and content image. The output feature maps will be saved in an array. Cause we may use different layers to compute the result, hence, in this step, layers are changeable.

```
#build a graph of tensorflow
g = tf.Graph()
#create a session to compute
    with g.as_default(), g.device('/cpu:0'), tf.Session() as sess:
#create a placeholder which is formally argument
        img = tf.placeholder('float', shape=shape)
```

```
#get results from pre built CNN network
        network = vgg.net_preloaded(weights, img, pooling)
#preprocess to subtract mean pixel value of vgg training dataset
        content/style = np.array([vgg.preprocess(content/style, mean_pixel)])
#get result of computed convolution layers
        for layer in CONTENT_LAYERS/STYLE_LAYERS:
            content/style_featuremaps[layer] = net[layer].eval(feed_dict={image:
                                                    content/style})
```

4.7 Back-Propagation Method

Generally, in training a convolution neural network, the training target are the convolution kernels. Firstly, according to the experience and early testing results, a suitable structure will be designed. This structure contains the size of convolution kernel, the number of convolution layers, the chosen of activation function and so on. The structure could play a dominant role in the performance of CNN model. Typically, early approach shows that the deeper network model is, the better performance it will bring. The reason is when using small size convolution kernel, more useful features with different size would be covered. But this will cost many compute time and space in computer.

After all the components of network structure are conformed, a dataset will be used to training the model which used to tell the model what kind of convolution kernel is useful. Hence, secondly, we input a set of labeled images into the model. The labeled image will give the result of the recognized images, for example, an image contains a cat will have a label cat. The output of the model will be the possibilities of what kind object the image contains. Hence, a loss function will be defined to sum all the dataset result compared to the labels which are the right answers. Therefore, the model training becomes a gradient descent process. The target is to minimize the loss function, and the modify parameters in this function is the convolution kernels. But the model contains many different layers, and all the convolution kernels contained in these different layers, the parameters modification could be a little complex. Here, a back-propagation method will be used to compute the descent of loss function. In the final output layer, the loss between labeled result and model output can be obtained and contained in loss function, a mathematic method, chain rule, propagates the final losses to the early convolution layer, hence, the early parameters can be modified. In this way, the best model that can fit the labeled dataset can be trained.

Thirdly, a test labeled dataset will be used to test the accuracy of the model. According to the test result, several methods can be used to improve the performance of the model.

In this project, we no longer need to training the convolution kernels, a set of pretrained convolution kernels form VGGnet will be obtained to build our convolution neural network. And the loss function is defined by the sum of the loss between the content representations of content image and result image and the loss between the style representations of style image and result image. In this loss function, the modified parameters are the pixels of result image. So, the TensorFlow variable need to be defined in program is an image. In TensorFlow, variable is a changeable value in the whole computation process. With TensorFlow, the mathematic result

of back-propagation is not necessary given by hand computation, TensorFlow provides many functions to automatically implement the back-propagation algorithm. And there are many gradient descent optimizers can be used to achieve a good optimized result of loss function.

Since the result image is generated by an initial random noise image. the intensity of pixel would change rapidly compared with its neighbor. Hence, the result image could be seemed not smooth enough. Usually a technology called total variation denoising will be used to minimize this sharp change between neighbor pixels in image processing [29]. The total variation is defined by the sum of all the variation between signals. In this project, the total variation will be defined as the square distance between neighbor pixels by two directions, one is X-axis, the other one is Y-axis. After defining total variation loss, it will be contained in the total loss function. Hence, the total loss function will consider three parts, content loss, style loss and total variation. While the total variation will not have a significant influence on the weight between style and content representation cause its only affect the value between pixels not between features. But the weight of the total variation should be considered not to be set a very large value because it could influence the minimize process of gradient descent.

TensorFlow provides many methods to perform gradient descent algorithm without define the computation by ourselves. Compare with traditional gradient descent methods, there are some improved method to accelerate the minimize process and avoid local minimized result [5]. Among these methods, Adam-optimizer is a widely used optimize method. The algorithm dynamically adjusts the learning rate for each parameter according to the first-order moment estimation and second-order moment estimation of each parameter gradient of the loss function [14]. Adam is also based on the gradient descent method, but each iteration parameter learning step has a certain range, the situation that the large gradient leads to a large learning step will not happen, the value of the parameters is relatively stable. Here, we define an Adam-optimizer to automatically minimize the total loss function. Hence, the main part of implementation has been finished. In next chapter, I will discuss the result of my implementation and evaluate different result from different content and style layers contained in the computation.

```
#style representation
#different with theory, the feature maps are reshape to a size
# of Ml * Nl, hence the computation of gram matrix is different
# to the definition
#reshape feature maps matrix
featuremaps = np.reshape(featuremaps, (-1, featuremaps.shape[3]))
#compute gram matrix
gram = np.matmul(featuremaps.T, featuremaps) / featuremaps.size
#mmd method
#mmd = np.matmul(features, features.T) / features.size
```

```
#define the adam optimizer
train_step = tf.train.AdamOptimizer(learning_rate, beta1, beta2, epsilon).
                                        minimize(loss)
```

Chapter 5

Results and Evaluation

5.1 Pooling Method

There are two kinds of pooling methods in convolution neural network, one is max-pooling which select the biggest filter responses among pooling area, the other one is average-pooling which compute the average value of filter response in the pooling area. It can be easily understood that, max-pooling has an advantage on selecting important features when used to recognize object. In the other words, the very small details of image content are not needed to recognize the whole object. Using max-pooling could reduce unnecessary parameters to accelerate the process speed. But when represent image style, we are trying to measure all the feature correlations to have a good performance on style simulation. Here a comparison result images between max-pooling and average pooling is given.

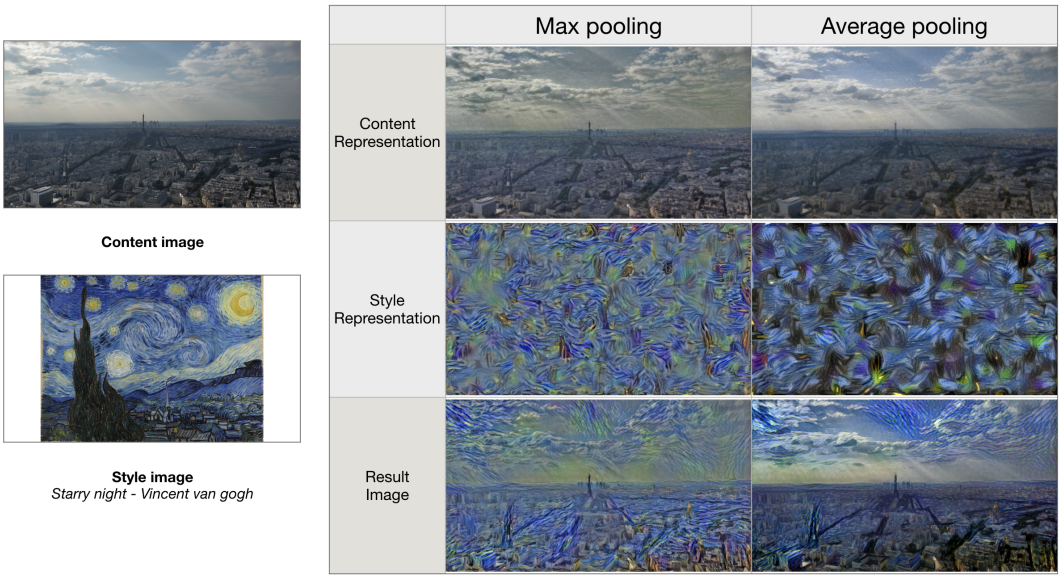

Figure 5.1: Pooling Comparison

Compare with the target style image in Figure 5.1, it can be easily found that the average pooling has a better performance. The most obviously reason is the color, the max poling result gives an unsatisfied color style transfer result which the blue looks not dark enough. Therefore, the pooling method will be average in later testing implement.

5.2 Content Generation with Different Layers

Since the implementation has been finished, a set of results can be obtained to be used to evaluate. The convolution neural network contained in this project has five convolution layers. When choosing feature maps from different convolution layers to represent the content and style of image. The results could be different because the feature responses captured by different convolution layer are not same. Theoretically, each element of feature map contains more information of original image pixels with the increasing number of the image passing through the network. When a feature map pass through next convolution layer, the new feature map will be the convolution result of the feature map and a 3*3 convolution kernel. Hence, one element of new feature map will be the convolution result of nine elements in the last feature map and the convolution kernel. In the other word, the receptive field increased. Although there could be one more sub-layer in one convolution layer, the result test will not contain comparing the difference among all the sub-layers. Because there are few differences between this convolution sub-layers, the receptive field changed not extremely and without pooling and activated function, the convolution is a linear process and does not remove any useless information.

Here I give several results which indicate the content representation of different convolution layer. First, the difference between content representations of sub-layers in a certain convolution layer is compared. The style weight is set as zero, the content layer used to computation is different, in the comparison, four convolution layers will be used, the two front sub-layers in these convolution layers contained in the comparison. These layers are $conv2-1$, $conv2-2$, $conv3-1$, $conv3-2$, $conv4-1$, $conv4-2$, $conv5-1$, $conv5-2$.

According to the comparison result images in Figure 5.2, it can be observed that the difference between sub-layer content representations is not obviously, while the difference between content representations of different convolution layers is significant large. The reason could be the convolution layer after a convolution layer could have few influences on the feature representation cause the receptive field size changes not rapidly. On the contrary, the pooling layer at the end of a convolution layer usually reduce the size of feature maps vary sharply. Hence, the feature representation will lose more details. Hence, later, we will use a certain sub-layer in the convolution layers to compare results, the difference between sub-layers will not be considered. From the comparison among results, it can be observed that with the increasing number of input image passes through the convolution layer, the content representation loses more detail features, on the other hand, the content representation becomes more abstract. This can be also explained by the increased receptive field size of the network and the pooling layer conditionally selecting useful information. Hence, a common choice of the content layer used for representing content is a middle convolution layer for having a balance between abstract result and saving

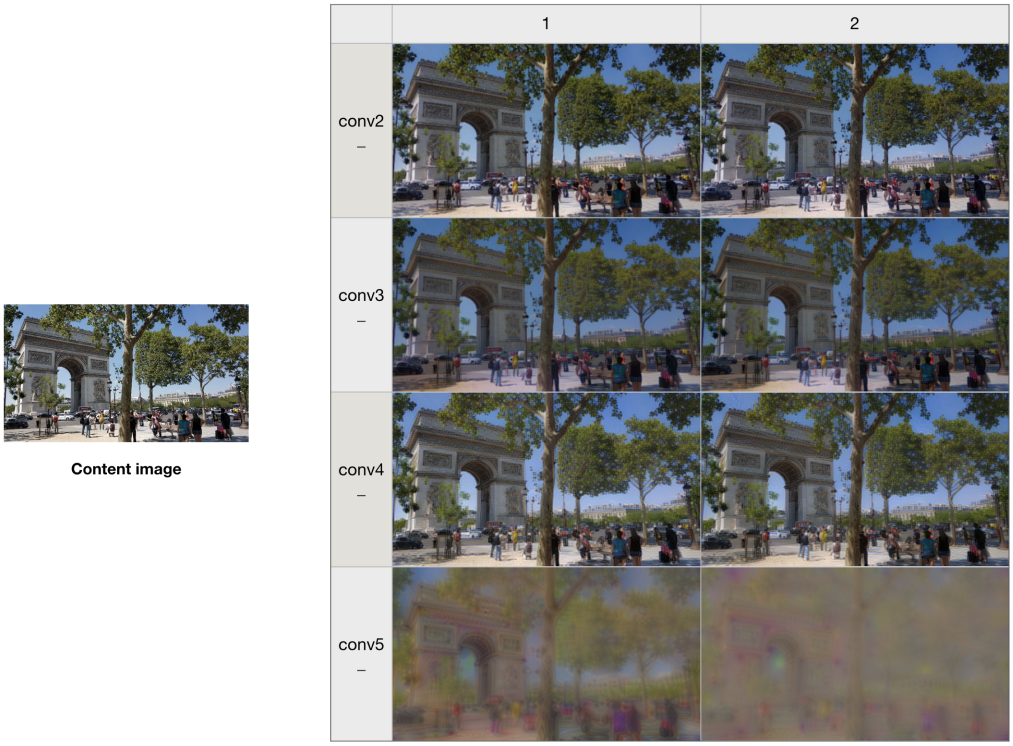

Figure 5.2: Content Generation Comparison

enough details.

5.3 Style Generation with Different Layers

According to the content representation comparison results, the small features will be abandoned which leads to a vaguer result image when choosing higher convolution layer to represent image. Since the style representation is a coefficient between different features. Here we test different combinations of convolution layers to represent image style. The test combination will be from first layer to fifth layer contain all the combined layers. The reason for do not chose any single convolution layer in further test is that only represent a certain size of features is useless. The goal is to achieve a global and fully described image style. Hence, every coefficient of features is necessary to represent the image style and compare the results of from the top layer from lower to higher.

The results shown in Figure 5.3indicate that with the top layer increased, the style image shows a growing scale size of style representation. Hence the receptive field size theory has been also proved in style generation. Also, by observing style representation images, it can be found that these results contain none local information. It proves that the using Gram Matrix to represent image texture is a global statistic method which gives a full vision of image style. The first test result with convolution layer $conv1-1$ gives an unsatisfied style representation. From the result image, all the style features are aligned at the edge of image. The reason is

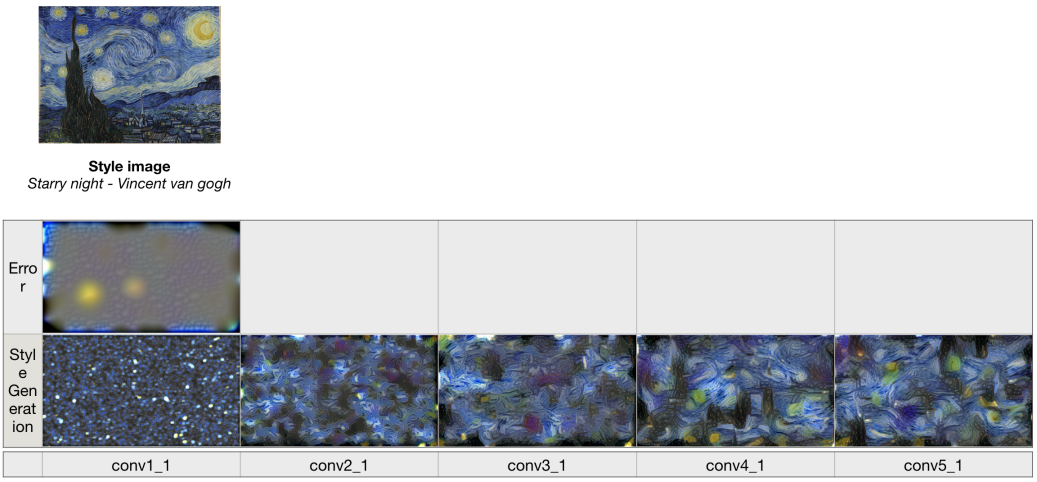

Figure 5.3: Style Generation Comparison

supposed to be since the $conv1-1$ represent style with a very small features response, the various between pixels could be extremely large. Hence the total variation loss takes an important role in the whole image generation. Therefore, we remove the total variation loss and generate a new style representation. The new result gives a more reasonable style representation. In later style transfer work, the style loss computation will contain all the five convolution layers to save all size of style representations to achieve a best result.

5.4 Style Transfer with Different Weights

Here a set of result images shown in Figure 5.4 is given which comes from same content represent layer and combination of style content represent layer. The style and content representation weight are different in this result generations. Hence, with a certain receptive field size, it can also achieve a limited flexible weight change to obtain more abstract stylized image. Furthermore, it can be observed that the content details contained in different result images are nearly same. Although the content representation becomes less obvious with the style weight grows, the fineness of content representation will not change. This reveals that the content fineness is only determined by the convolution layer used to generate content representation.

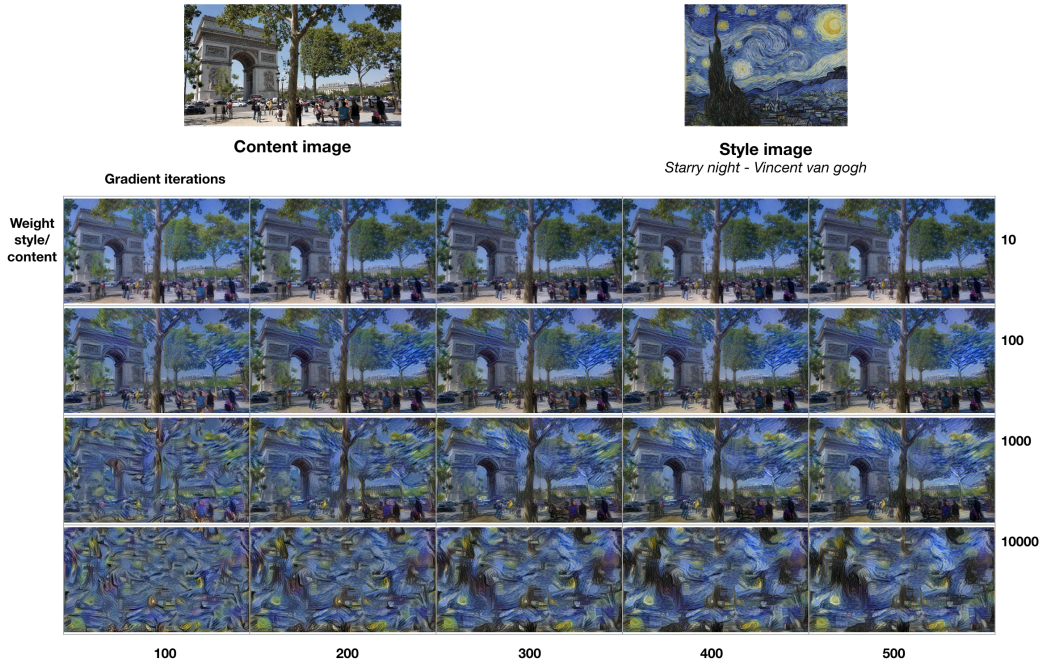

Figure 5.4: Different Weights Between Content and Style Generation

5.5 Style Generation with MMD Function

Here I used an MMD function to generate style representation. Compared to Gram Matrix, the MMD corresponding Metrix is computed by changing the direction of inner product in feature maps contained matrix, hence we can just replace the order of feature maps matrix and its transpose in the Gram Matrix compute formula.

Since the size of MMD corresponding Metrix is $M_l \times M_l$ which could be extremely large, compared to the size of Gram Matrix $\mathrm{N}_l \times N_l$ which the largest number of feature maps in VGGnet is 512. But the largest size of feature maps is the size of input image which is width multiples height of input image which is much bigger than 512. During implement, the computation cost so many storages that my computer cannot perform. Hence, I cut the input image size, and only test the style generation result with the style representation of first convolution layer.

The result shown in Figure 5.5 that the difference between these two methods. The Gram Matrix gives a global texture representation with the colours distribute separately, while the MMD gives a result that colour features distribute in a similar location of the original image. In the future, more complex style of image could be test with these two methods by a high-performance computer.

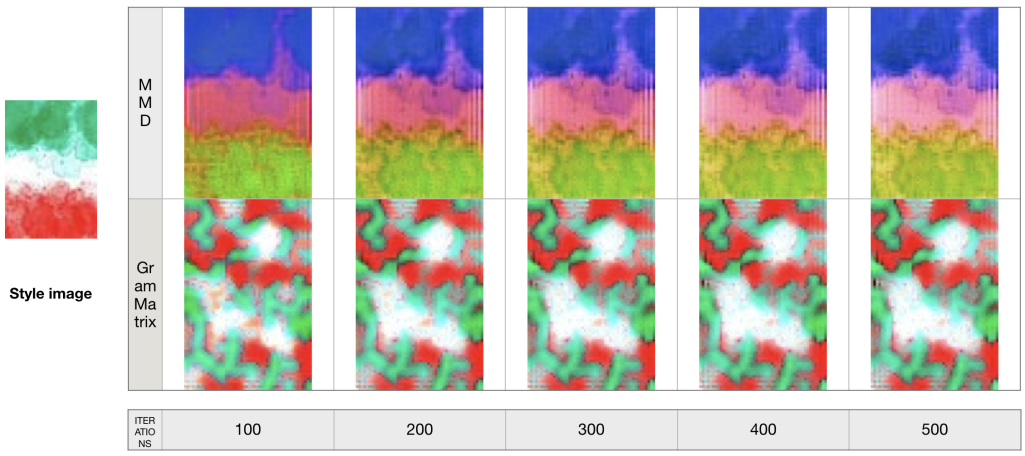

Figure 5.5: Style Generation by Gram Matrix and MMD Method

Chapter 6

Summary and Discussion

In this report, a style transfer method based on convolution neural network has been introduced [17]. And the implement shows it has a good performance when dealing with some artist paintings. Also, the implement program provides a component change flexibility which can adjust content image, style image, content and style represent weight and representation layers conveniently. This could bring an easy operate experience for user to select a best style transferred result image.

The convolution neural network is a powerful model which used to recognize object. The theory of the recognition process is based on regard target object as several separated features, then combine all the coefficients of features to give a possibility about a particular object. Hence, the convolution neural network is designed to have a good performance on extract useful features. The multiple layers architecture can provide a various size of feature extract which can avoid the miss of some certain necessary features. Recently, there are many different kinds of pre-trained convolution neural network which performing a quite accuracy on object recognition testing. In this project, a pretrained network called VGGnet is used to extract features. It provides the convolution kernel and the construction of convolution layers to build our feature extracted model.

The main algorithm of style transfer is we represent content and style of an image by the output feature filter responses of convolution layers in our CNN model, hence, we build the content and style representation, an image has the same content representation with the content target image and the same style representation with the style target image can be viewed as the style transfer result. And the method used to generate result image is gradient descent which used to minimize the distance between these representations.

The implementation of this report is based on TensorFlow with Python. TensorFlow is a popular deep learning framework which can provide a significant assistance to users. With TensorFlow, it can be easily achieved to build a convolution neural network and implement gradient descent method.

In order to evaluate the results and demonstrate the theory of style transfer. Several comparation results have been generated with different components. First, we compare the two types of pooling. According to the result, we found that max-pooling may lead to a features loss

while average pooling gives a better style representation. Second, we used different convolution layer to generate content image. By analyzing the different result images, we proved a theoretical conclusion that the receptive field size of model increased with the convolution layers increasing. Later this conclusion will have a similar performance in the style generations which the result image shown a scaled trend with the input image passing through more convolution layers. Based on this finding, we usually use one middle convolution layer to generate content representation to have a suitable balance on the saving content detail and give a abstract result. And the style generation will cover all the convolution layers in the model to achieve a complete artist painting style. In this way we will keep all the correlations between image features with different sizes. Later approaches are change the content and style generate weight while the layers to generate content and style representations are determined to achieve different style transfer results in a certain range. All the result shows the implement provides a convenient way to obtain different degrees of stylized images.

Since the input style and content image can change flexibly in this implement, it can be considered to use this project to develop an application for users. Users can input any content image and style image to have a stylized result. And the application can also provide a weight changeable for users to obtain their favorite stylized image. Although this method implements a little slow in personal computer, in my MacBook, it takes nearly half hour to finish a 300kb image style transfer, it could have a fast-enough processing speed in professional server. Besides, there is another method to accelerate style transfer process, a feed-forward model can be trained to perform a stylized process on input content image. This model could modify content image pixels directly to minimize the distance of style representation between content image and style image. But this feed-forward model can only be trained to deal with one certain style. Although the process speed could be fast enough, it still needs to pretrain many models to satisfy different needs of users.

This project may reveal the mechanism of human visual to recognize a high-level image information. Compare with normal painting content, style indicates a more complex expression of artist. Although the reason of how can an artist creates their own style and uses style to explain their feelings is still unknown, this project could be a first step for researchers understand how style works. Furthermore, we cannot really synthesis content of one image and style of other image, all we can do is find a balance between content representation and style representation. Every pixel value contains content information and style information. But human visual system can be cheated easily, so find an appropriate weight between content and style can also achieve a good result. This theory could also transfer to other research field, for music, every song has a harmonious between its lyrics and melody. It could be absolutely weird to synthesis lyric of one song with the melody of the other song. But it could be success to modify them and achieve a good balance to cheat our ears. In a conclusion, this work may give a significant spirit to the researches about human brain system.

References

[1] A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. In Computer Vision, 1999. *The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 10331038. *IEEE*, **1999**.

[2] A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. *In Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 341346. ACM, **2001**

[3] Alex Krizhevsky, Ilya Sutskever, Geoffrey E.Hinton. ImageNet Classification with Deep Convolutional Neural Networks.

[4] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schlkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723773, **2012**.

[5] Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro and Benjamin Recht. The Marginal Value of Adaptive Gradient Methods in Machine Learning. *arXiv*: 1705.08292v2 [stat.ML] **22 May 2018**. URL https://arxiv.org/pdf/1705.08292.pdf. *arXiv*: 1705.08292v2

[6] B. Julesz. Visual Pattern Discrimination. *IRE Transactions on Information Theory*, 8(2), **February 1962**.

[7] Daniel Yamins, James J Dicarlo. Using goal-driven deep learning models to understand sensory cortex. **2016**, *Nature Neuroscience*.

[8] D. J. Heeger and J. R. Bergen. Pyramid-based Texture Analysis/Synthesis. *In Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH 95, pages 229238, New York, NY, USA, **1995**. *ACM*.

[9] E. P. Simoncelli and W. T. Freeman. The steerable pyramid: A exible architecture for multi-scale derivative computation. *In Image Processing, International Conference on*, volume 3, pages 34443444. *IEEE Computer Society*, **1995**.

[10] Eric Tzeng, Judy Hoffman, NingZhang, Kate Saenko, and Trevor Darrell. Deep domain confusion: Maximizing for domain invariance. *arXiv* preprint *arXiv*:1412.3474, **2014**.

[11] Gatys, L. A., Ecker, A. S. & Bethge, M. Texture synthesis and the controlled generation of natural stimuli using convolutional neural networks. *arXiv*:1505.07376 [cs, q-bio] (**2015**). URL http://*arxiv*.org/abs/1505.07376. *arXiv*: 1505.07376.

[12] Hubel D. H, T. N. Wiesel. Receptive Fields of Single Neurones in The Cat's Striate Cortex. *Journal of Physiology*, (**1959**) 148, 574-591.

[13] J. Portilla and E. P. Simoncelli. A Parametric Texture Model Based on Joint Statistics of

Complex Wavelet Coefficients. *International Journal of Computer Vision*, 40(1):4970, **October 2000**.

[14] Kingma, D. P., & Ba, J. L. (2015). Adam: a Method for Stochastic Optimization. *International Conference on Learning Representations*, 113. [15] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv*:1409.1556 [cs], **September 2014**. *arXiv*: 1409.1556.

[16] Leon A. Gatys, Alexander S. Ecker, Matthias Bethge. Texture Synthesis Using Convolutional Neural Networks. *arXiv*: 1505.07376v3 [cs.CV], **Nov 2015**. *arXiv*: 1505.07376v3.

[17] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. *In CVPR*, **2016**.

[18] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael I Jordan. Learning transferable features with deep adaptation networks. *In ICML*, **2015**.

[19] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):13451359, **2010**.

[20] Y. C. Jing, Y. Z. Yang, Z. L. Feng, J. W. Ye, Y. Z. Yu and M. L. Song, Neural Style Transfer: A Review. *arXiv*: 1705.04058 [cs.CV], **May 2017**. *arXiv*: 1705.04058

[21] Yanghao Li, Naiyan Wang, Jiaying Liu, Xiaodi Hou. Demystifying Neural Style Transfer.*arXiv*: 1701.01036 [cs.CV], **Jan 2017**. *arXiv*: 1701.01036

[22] Y.LeCun, B.Boser, J.S.Denker, D.Henderson, R.E.Howard, W.Hubbard, and L.D.Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, **1989**.

[23] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In David Touretzky, editor, *Advances in Neural Information Processing Systems 2 (NIPS*89)*, Denver, CO, **1990**, Morgan Kaufman.

[24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, **november 1998**.

[25] https://www.princeton.edu/ cuff/ele201/kulkarni_text/frequency.pdf

[26] http://www.robots.ox.ac.uk/ az/lectures/ia/lect2.pdf

[27] http://www.vlfeat.org/matconvnet/pretrained/#downloading-the-pre-trained-models

[28] https://www.tensorflow.org/api_docs/python/tf/nn/conv2d

[29] http://eeweb.poly.edu/iselesni/lecture_notes/TVDmm/TVDmm.pdf

[30] http://www.mathworks.com/help/deeplearning/ref/vgg19.html