Image Style Transfer Using Convolutional Neural Network

Qing Mu, Wen Sun

Abstract

Generating image with desired style can be a difficult image processing task. Traditional multimedia techniques require users to go through certain procedures to transfer a content image into a desired art style. Here we use A Neural Algorithm of Artistic Style [1] to separate and recombine the image content and style of natural images. Through this algorithm, we are able to use image representation derived from Convolutional Neural Network optimized from object recognition to explicitly represent high level image information. With these information, we are able to generate images combining numerous art style with desired content with high perceptual quality. This project provides a solution for image processing and art composition.

Introduction

Motivation

Nowadays, people use mobile applications to modify their photography and post their compositions on social medias. This is when aesthetic value of the images becomes more and more important. With great contribution of dozens of amazing artists, using classic art pieces styles on our own photos would be a much easier way for us to recreate art pieces of our own. That is, we want to generate an image with the content of a content image and the style of an artwork. To achieve this goal, we investigated recent studies and choose to apply Convolutional Neural Network to help establish the image generating process, which has achieved satisfying results.

Background

Generally speaking, transferring style can be considered as transferring texture. In this way we need to synthesis texture from style image while constraining the target image with the content image. In former studies, there is a large range of algorithms that can synthesis images through re-sampling pixels which are limited to low-level image features.

However, the advance of Convolutional Neural Network provides us with powerful tools to extract high-level semantic information. In this project, we implemented the transfer of styles from one(or multiple) images to one content image using *Neural Algorithm of Artistic Style* [1] put forward by L. Gatys et al in 2016. This algorithm uses high-level Convolutional Neural Network to extract high level image features to synthesis image with desired style and high perceptual quality.

In our work, we have generated amazing artistic images using VGG-19 pre-trained network by ImageNet. ImageNet is a large visual database covering over 14 million URLs of images and more than 20 thousand ambiguous categories which helps to cover more diverse features when it comes to natural images.

Our Algorithms

The main idea of our project is to extract features from certain layers of VGG-19 [2] to generate the target image. To get result of higher perceptual quality, we choose relu4_2 and relu5_2 to extract content while using relu1_1, relu2_1, relu3_1, relu4_1 and relu5_1 to extract texture and color from the style image.

The cost function is defined over the output of content layers and style layers. Afterwards, we use Adam algorithm to optimize the network and back-propagation to generate the target image.

Methods

There are three steps to generate a target image. First, extract feature map from content image and a random generated image through VGG-19 and try to minimize the difference. Second, evaluate the correlation between two patterns and to transform it. Here we use dot product of to represent the correlation. Third, define the total loss and use Adam algorithm to optimize the result. The process is described in detail in this chapter. We also took experiments with groups of different hyperparameters and compared the results, which is further discussed in Result chapter.

1. Environment

Operating System: Windows 10
Python: v3.6.4
Tensorflow: v1.6.0
CUDA: v9.0 (optional)
GPU: NVIDIA GeForce GTX 1060 (optional)

2. Network

Network Structure

Generally each layer in the network defines a non-linear filter bank whose complexity increases with the position of the layer in the network. Hence a given input image $\vec{x}$ is encoded in each layer of the Convolutional Neural Network by the filter responses to that image. In this part, we use ImageNet pre-trained VGG-19 network weights. ImageNet is an image dataset organized according to the WordNet hierarchy. It is available publicly. We are only using the feature space of 16 convolutional layers and 5 pooling layers of the VGG-19 network, which is a 19-layer VGG network.

The structure of the network is shown in image 2-1. This model is available publicly.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

2-1 VGG-19 network consists of 16 convolutional layers, 5 pooling layers and 3 fully connected layers.

Network Analysis

Take image classification for example. For each layer, every hidden unit is looking for the image patches that maximize the unit's activation function. This means that they are looking for the image patches that correspond most to the pattern, or feature, of the units. The deeper the layer is, the larger region of the image it looks at, and the more complex feature it looks for. Therefore, along the processing hierarchy of the network, the input image is transformed into representations that are increasingly sensitive to the actual content of the image, but become relatively insensitive to its precise appearance. Image 2-2 shows a set of hidden units in different layers.

2-2 Convolutional Network Neuron Example

To visualize the image information that is encoded at different layers of the hierarchy one can perform gradient descent or other optimizers on a white noise image to find another image that matches the feature responses of the original image.

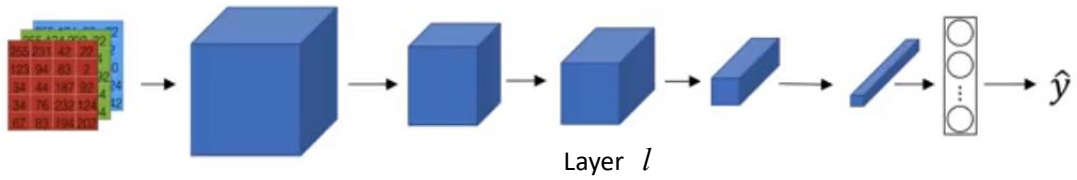
3. Content representation

When we do the content representation, we need a criteria to measure how much information the representation image loses from the content image, and optimize the representation according to it. Here, we introduce the cost function.

Consider only the loss of content image. Use C to represent content image, and use G to represent generated image. Let $a^{[l](C)}$ and $a^{[l](G)}$ be the activation of layer l on the images. When $a^{[l](C)}$ and $a^{[l](G)}$ are similar, it means that both images have similar content. Thus, a content cost function measures the difference between the content of generated image and the content image. So for each layer, $\mathrm{J}_{content}(C,G)=c\left\|a^{[l](C)}-a^{[l](G)}\right\|^2$, in which c is a normalization coefficient. This function is a part of the total cost function.

4. Style representation

Assume we are using activations of layer l to measure the style. Define style as correlation between activations across channels.



2-3 Layers of a network

Use S to represent style image, and use G to represent generated image. We can see the activations of layer l as a cube with n_H height, n_{W} width and n_{C} channels, which is shown in image 2-4. The correlations of the activations across different channels indicates which of the high level texture components of the neurons tend to occur together. The higher the correlation is between two channels, the more likely the textures of these two channels occur together. This representation works for both S and G. When compare the correlations of S and the correlations of G, one is able to know how close G is to S. With this feature, the cost function of style part is introduced.

2-4 Activation of layer l can be seen as a cube with n_H height, n_{W} width and n_{C} channels

Let $\mathrm{a}_{i,j,k}^{[l]}$ be the activation at (i,j,k). Because there are n_C channels, we need a $n_C \times n_C$

matrix $G^{[l]}$ to save the correlations across all channel pairs. Given two channels k and k', define gram matrix $G^{[l]}_{k,k'}$ as the correlation matrix between these two channels. Thus, for layer l, the correlation across channels k and k' is $G^{[l]}_{k,k'} = \sum_{i=1}^{n_H} \sum_{j=1}^{n_W} a^{[l]}_{i,j,k} a^{[l]}_{i,j,k'}$. Apply this formula on both style image and generated image, one is able to measure the style difference between S and G, or in other words, the loss of the generated image from the style image.

Let $G^{[l](S)}$ be the style matrix of style image on layer l, and let $G^{[l](G)}$ be the style matrix of generated image on layer l. So the style cost function of layer l is $J^{[l]}_{Style}(C,G) = c\left\|G^{[l](S)} - G^{[l](G)}\right\|^2_F = c\sum_{k=1}^{n_C}\sum_{k'}^{n_C}\left(G^{[l](S)}_{kk'} - G^{[l](G)}_{kk'}\right)^2$, in which c is a normalization coefficient. The overall style cost function is the sum of style cost functions across all the layers: $J_{Style}(S,G) = \sum_l \lambda^{[l]} J^{[l]}_{Style}(S,G)$, where λ is the hyperparameter defining which layer counts more in the style cost function.

5. Style transfer

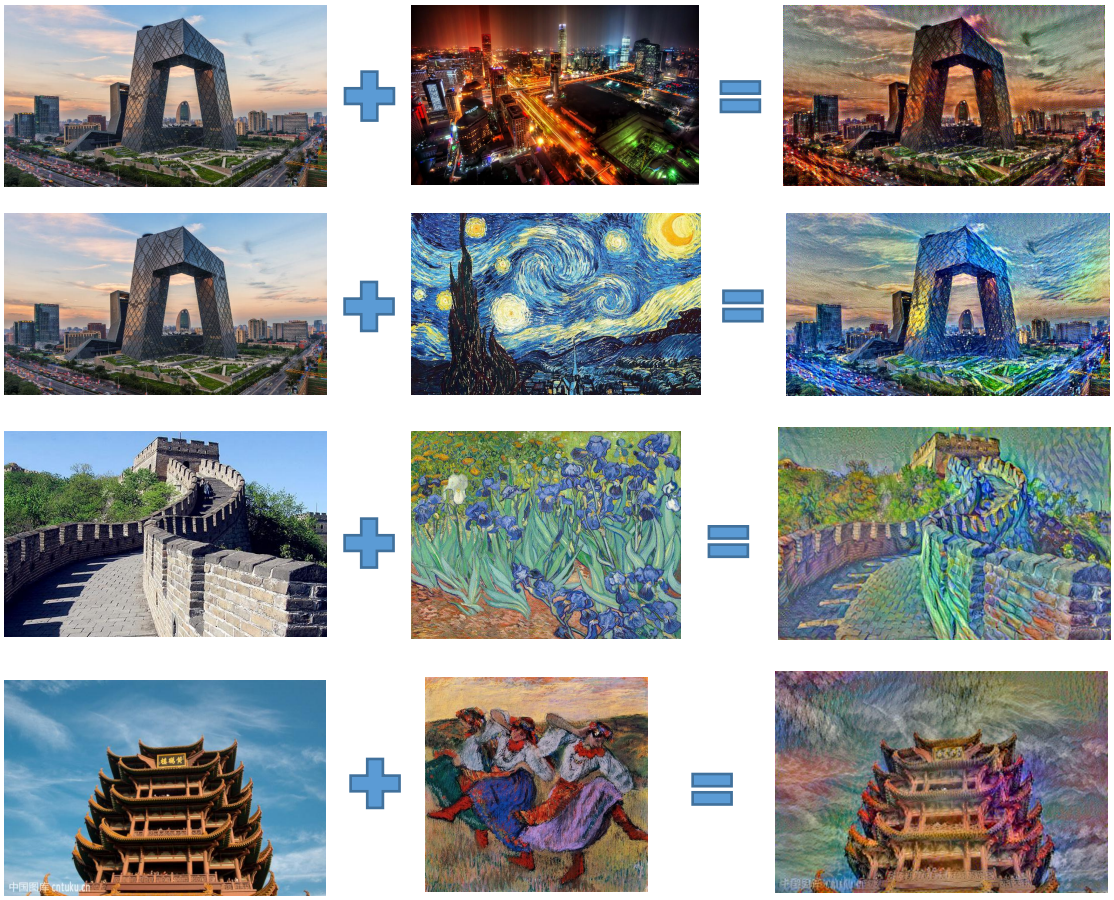
Now, assume that given a content image C and a style image S, we are going to generate an image G. Thus, the total loss of image G is the weighted sum of the lost of image C and the lost of image S, which is $J(G) = \alpha J_{content}(C,G) + \beta J_{Style}(S,G)$. Hyperparameters α and β represent the weight taken by the cost of content image and the cost of style image.

When we do the transfer, we first initialize the image G as a white noise image with every pixel value generated randomly. Then, use optimizer, such as gradient descent, to minimize the value of the cost function. In this part, we took experiments on three factors which has influence on the generated image. First, we set the optimizing iteration times to several different numbers and compared the result. Second, we changed the weights of style layers and content layers and compared the results. Third, we adjusted the hyperparameters α and β and got several groups of outputs. All the results are shown in Result chapter.

Our Results

1. General results on different images

We did experiments on many content images and style images, and we found that the content images with clear edges and contrast, and style images with smooth textures and bright colors can generate better results.

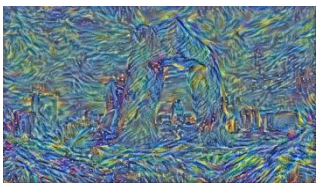


2. Results of different iteration numbers

Content: 

Style: 

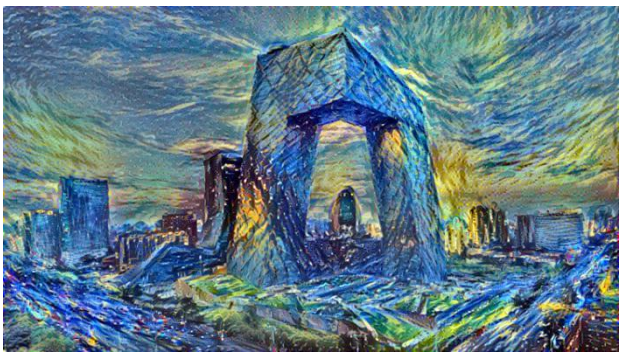
ITERATION = 1



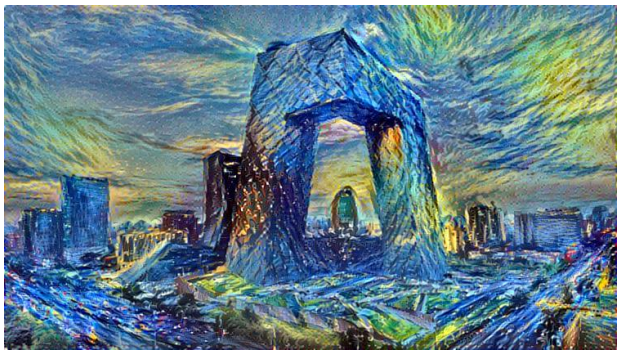
ITERATION = 100

ITERATION = 300

ITERATION = 500



ITERATION = 750



ITERATION = 1000

3. Combining multiple style images

We are able to add more than one style images to generate the result image. The texture and color of the style images will be combined together.

A) Group A

Content: 

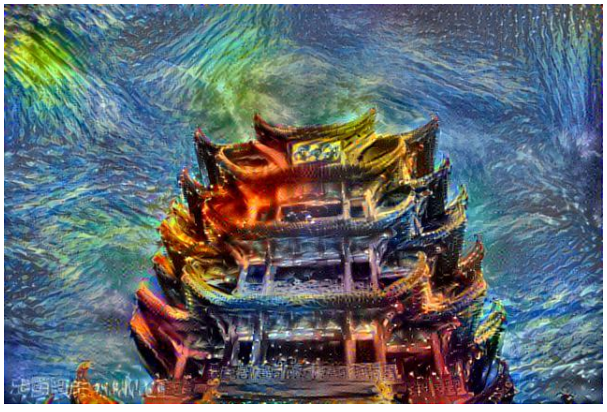
Style: 

Generated Image

B) Group B

Content: 

Style: 



Generated Image

4. Changing the weights of content and style

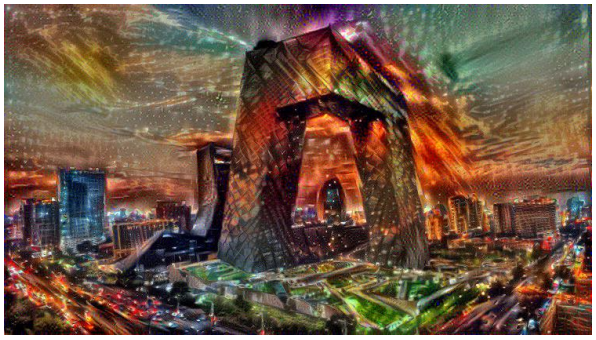
We did experiments on several groups of content images and style images, and we found that when style weight is between 10 times and 100 times of content weight, the result is better.

A) Group A

Content: 

Style: 

STYLE_WEIGHT = 10*CONTENT_WEIGHT

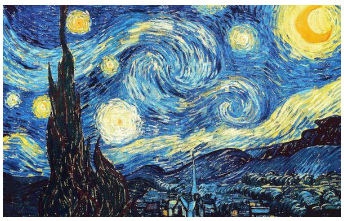


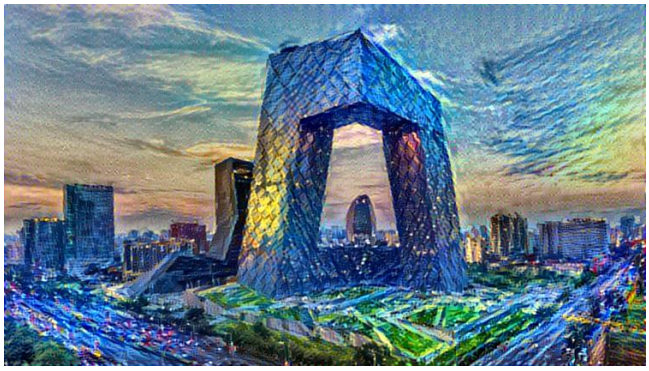
STYLE_WEIGHT = 100*CONTENT_WEIGHT

STYLE_WEIGHT = 1000*CONTENT_WEIGHT

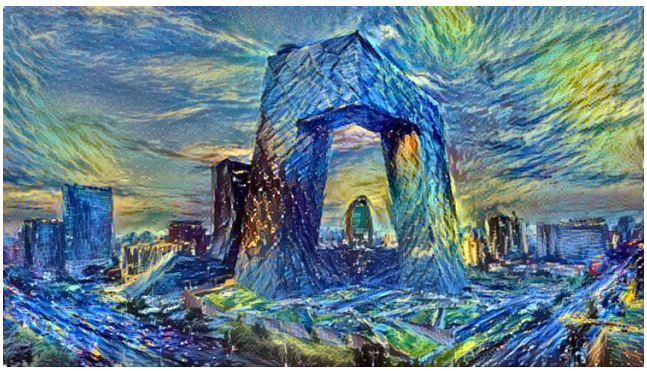
B) Group B

Content: 

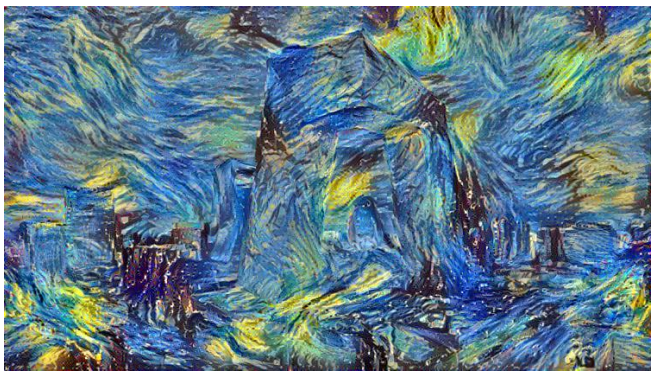
Style: 



STYLE_WEIGHT = 10*CONTENT_WEIGHT



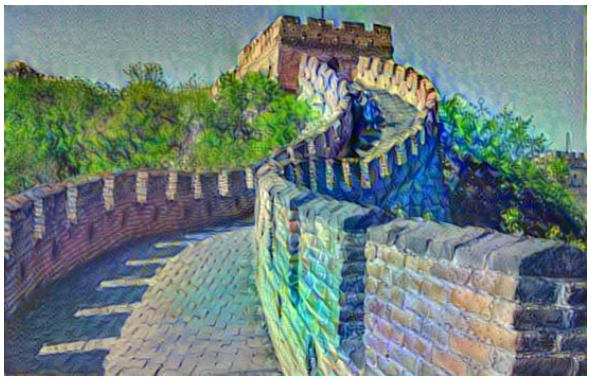
STYLE_WEIGHT = 100*CONTENT_WEIGHT



STYLE_WEIGHT = 1000*CONTENT_WEIGHT

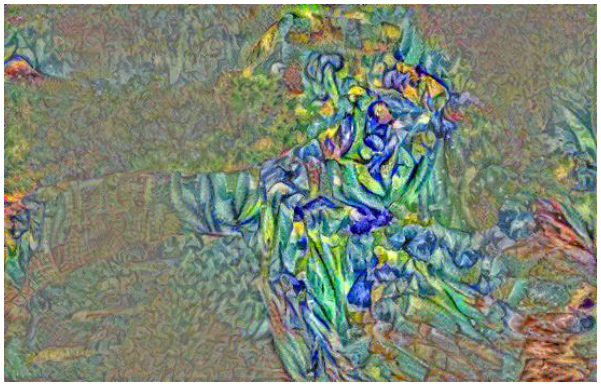
C) Group C

Content:  Style: 



STYLE_WEIGHT = 10*CONTENT_WEIGHT

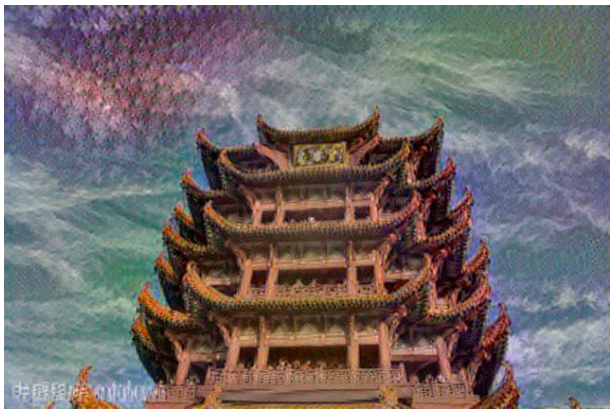
STYLE_WEIGHT = 100*CONTENT_WEIGHT



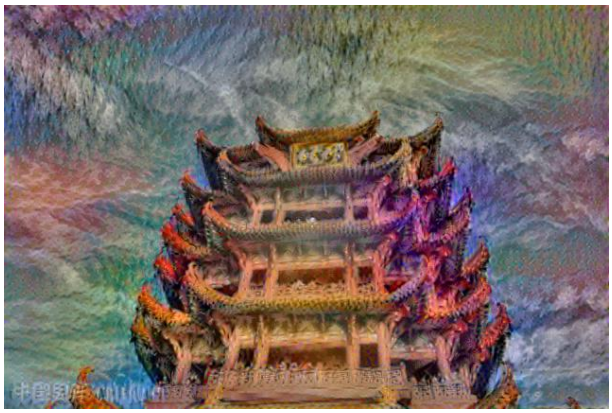
STYLE_WEIGHT = 1000*CONTENT_WEIGHT

D) Group D

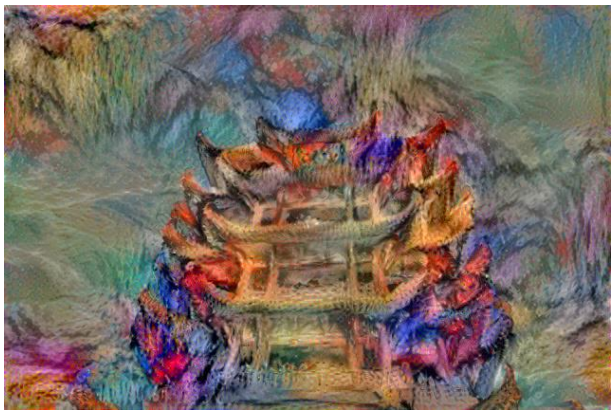
Content:  Style: 



STYLE_WEIGHT = 10*CONTENT_WEIGHT



STYLE_WEIGHT = 100*CONTENT_WEIGHT

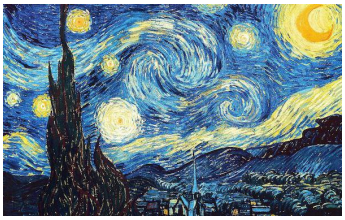


STYLE_WEIGHT = 1000*CONTENT_WEIGHT

5. Changing the weights of network layers

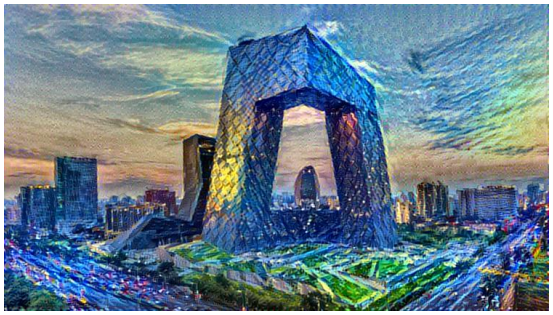
Since we are using the feature map of several layers of style image and content image for representation, we have weights for those layers in the cost function. In default, the weights for each layer are the same, but we are able to change the weights. Here, we did groups of experiments to see how these weights can influence the generated results in visual effect.

Content: 

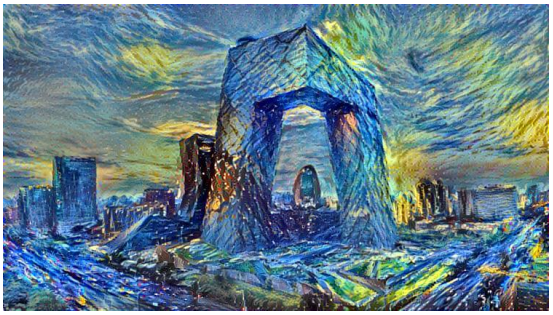
Style: 

A) Group A - Content Layer Weights

When computing the content loss function, we use feature maps of two layers of the content image and generated image. The value of parameter CONTENT_WEIGHT_BLEND indicates the ratio of the first layer. We can see that the more weights of the relu4_2 layer is, the clearer the extracted content feature is in the result image.



CONTENT_WEIGHT_BLEND = 1.0



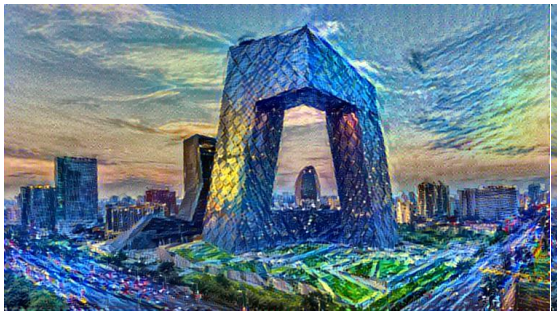
CONTENT_WEIGHT_BLEND = 0.8

CONTENT_WEIGHT_BLEND = 0.5

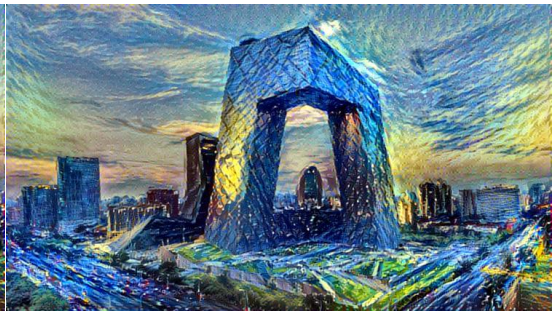
CONTENT_WEIGHT_BLEND = 0.1

B) Group B - Style Layer Weights

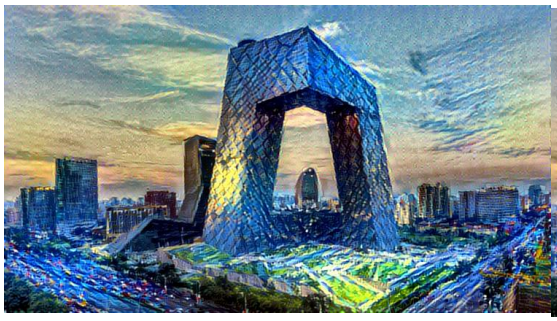
When computing the style loss function, we use feature maps of five layers of the style image and generated image. The value of parameter STYLE_LAYER_WEIGHT_EXP indicates the ratio of every layer to the previous layer. We can see that the more weights of the relu1_1 layer is, the less stylized the result is.



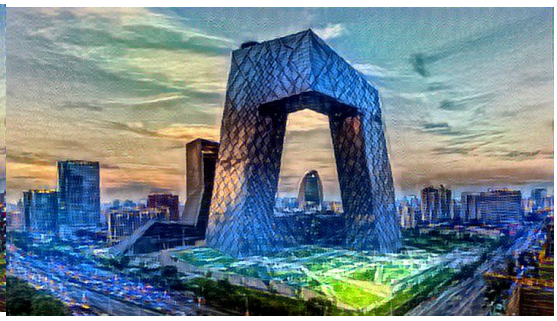
STYLE_LAYER_WEIGHT_EXP = 1.0



STYLE_LAYER_WEIGHT_EXP = 0.5



STYLE_LAYER_WEIGHT_EXP = 0.2



STYLE_LAYER_WEIGHT_EXP = 0.01

6. Preserving the color of the content image

We also added the feature that we can preserve the color of the content image and only use the texture style of the style image.

Content: 

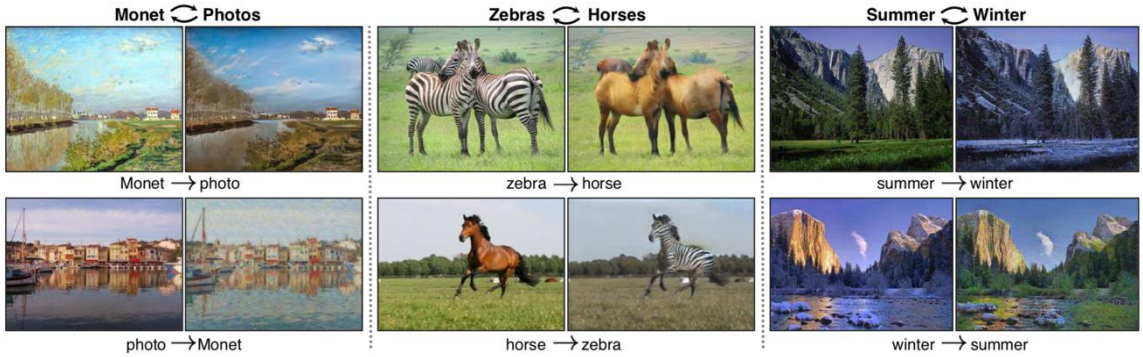
Style: 

STYLE_WEIGHT = 100*CONTENT_WEIGHT, PRESERVE_COLOR = TRUE

Discussion

To make generated image more smooth and continuous when transferring the texture, we can try to add denoising procedure into each iteration steps. We have performed simple image processing using Tensorflow. There were also plenty of algorithms which can handle image denoising in a satisfying manner. Viren Jain et al. [3] have put forward algorithm to denoise natural images with convolutional network in the earlier years.

With more and more detailed studies in convolutional neural network, there are handful of style transfer algorithms with better visual effect and less constrains on images. J. Zhu et al. [4] have put forward algorithm to transfer style from an style domain using GAN which helps computer to imagine what an image looks like under other conditions.



4-1 style transfer using GAN

Philip Isola et al. [5] have applied condition GAN to generate images with a sketch.

4-2 style transfer using condition GAN

References

[1] L. Gatys, A. Ecker, and M. Bethge. A neural algorithm of artistic style. arXiv preprint arXiv:1508.06576, 2015.

[2] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556 [cs], Sept. 2014. arXiv: 1409.1556. 3

[3] Jain and S.H. Seung. Natural image denoising with convolutional networks. In Daphne Koller, Dale Schuurmans, Yoshua Bengio, and Leon Bottou, editors, Advances in Neural Information Processing Systems 21 (NIPS’08), 2008.

[4] J. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent ad- versarial networks. In International Conference on Computer Vision (ICCV), to appear, 2017.

[5] P.Isola,J.-Y.Zhu,T.Zhou,andA.A.Efros.Image-to-image translation with conditional adversarial networks. arxiv, 2016. 4, 8

[6] Coursera: Convolutional Neural Network by Andrew Ng:
https://www.coursera.org/learn/convolutional-neural-networks

[7] Neural Style by Anish Athalye:
https://github.com/anishathalye/neural-style