Deep Convolutional Neural Networks for Artistic Style Transfer

Xikai Chen, Samarth Tripathi, Yitong Xu
Columbia University
xc2358@columbia.edu, st3029@columbia.edu, yx2376@columbia.edu

***Abstract*—The Convolutional neural networks (CNNs) have been achieving great success in feature extraction of images, leading to wild applications on image processing. Recent research on style transfer introduces a new algorithm to transform the raw images into the works of arts using styles of masterpieces. In this project, we propose to build our CNNs to extract different features of images and decouple their contents and styles to reproduce the similar visual effect. In this project, we first implement the vanilla style transfer algorithm, then, we try out the state-of-art style transfer algorithm [6], annotating the input images with a semantic map.**

***Keywords*-CNNs, image process, style transfer;**

I. Introduction

The Neural style algorithm was first proposed by *Neural Algorithm of Artistic Style* (2015,Gatys et al) [1], has been very successful in mixing the main object of an image with astonishing style of famous paintings. It recomposes one image in the style of another. Because of its attractive artistic results, style transfer algorithm has received an increasing attention from both artists and scientists. However, sometimes we need to pay attention on the choice of picking the content image and style image, since we may get some undesirable results due to the "wrong combination" of style and content or bad choice of parameters. Recently, some research find ways to solve these problems. [2] utilizes the Markov random field, which has the ability to model the correlation of among all $k \times k$ image blocks, preserving the plausibility of the photo-realistic styles. [3] allows us to control the output images to meet our initial expectation, rather suffer from some image quality glitches. To further advance the exploration of the original paper [1], and get more reasonable result from the CNNs, we also try the state-of-art algorithms, which is introduced by *Semantic Style Transfer and Turning Two-Bit Doodles into Fine Artwork* (2016, Champandard et al) as well.

II. Summary of the Original Paper

The authors make use of the strong power of pre-trained CNNs architecture on classification task, VGG-16, to extract abstract features from higher layers of ConvNets. Higher-level and more abstract content features from deeper layers can obtain more spatial arrangement of main object, while higher-level features don't force the image pixels to be exact. On the other hand, style representations are obtained from Gram matrix, which will be elaborated in the following section. By jointly minimize the content loss and style loss, the style-transferred images will be created.

A. Methodology of the Original Paper

In this section, we will carefully introduce how Neural Style works. At the very beginning, we need to feed the network with two images, Content image matrix F, and Style image matrix S. Then we will create a white noise image matrix P, such that in the end, P image will both have the content in F and style in S. The loss is computed from the content loss and style loss. Before dive into the details on each of them, notice that CNNs activations are a series of features maps where each one captures a specific feature from the previous layers's output. Hence, the higher layers tend to capture more abstract and global features of the input images. The author employs the pre-trained VGG [5] network, with 16 Conv-layer and 5 pooling layer to extract features from image. Content features are obtained in the feedfoward network. The author only picks out the features from "Conv4_2", since he believes that those features contain sufficient information about the content image than lower layer. We call all features in this layer l. The context loss then is defined as

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

which measures the pixel-wise dissimilarity between the output image and content image. The style loss is computed from multiple layers of features, in each layer, the style features are measured by gram matrix. Which is defined by

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l$$

Assume that we have n filters at a particular layer l. Now we can unroll an activation volume of dimensions (n, w, h) into a matrix of (n, wh), and let's call this matrix V. The Gram matrix is found by performing VV^T . Since each row of matrix V represent a flattened feature map, hence by calculating VV^T, each element of the $n \times n$ gram matrix is the inner product of each feature map, can also be thought as a measure of correlation. Surprisingly, it turns out the Gram matrix keeps the style information of the image. Although exact pixel values at given locations are lost, Gram matrix retains the correlations between every pair of feature map. Finally, the loss function is defined

as $L_{total} = \alpha L_{content} + \beta L_{style}$, where α and β are hyperparameters, which control the relative importance of each term to us, e.g. setting alpha larger means we prefer to have smaller content loss, hence the image X will more tend to hold the original content information. We still perform gradient descent on training our model, but the main difference is that, in common ConvsNets, we iteratively update the weight vectors to minimize the loss function. But here, we want to minimize the pixel-wise squared $L2$ loss, adjust the pixel values in white noise image X, to minimize the loss function. To update the target image, we choose L-BFGS optimizer. The update process is in Figure 1.

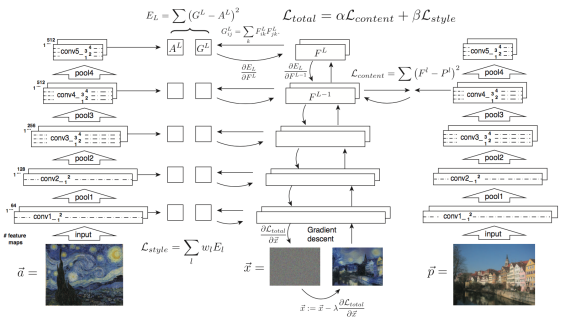

Figure 1: Style transfer algorithm. The style image $\vec{a}$, The content image $\vec{p}$ and the random white noise image $\vec{x}$ are be put in a VGG-16 network together.

B. Key Results of the Original Paper

The key finding of *Neural Algorithm of Artistic Style* (2015,Gatys et al) is that, it first proposed that image could be decoupled with style and content independently. Moreover, it shows that the low-level layers can almost reproduce the exact style image, because the lower layers mainly focus on the image texture manipulation, while the higher layer extract more abstract feature, with the risk of losing details (2015, Shiqing He).

III. Methodology

In our project, we basically replicate the VGG-16 [5] network architecture, with fully connected (FC) layers and softmax layer being ignored. We use VGG-16 architecture to extract the features. In the section 3.1, we will elaborate more on our model. In section 3.2, we will discuss the main challenges.

A. Model Architecture

We train an image classification model, inspired from VGG network. It uses Convolution layers(Conv), Max-Pooling layers(Pool), Hidden Layers and Softmax Layer, and has the following architecture - [(Conv, Conv, Pool), (Conv, Conv, Pool), (Conv, Conv, Conv, Pool), (Hidden, Hidden), (Softmax)]. The model is implemented in Theano, The convolution layers use kernel sizes as [3,3], Relu as Activation function and keep the output size same as input size using Zero Padding. The number of kernels for each Convolution Layer are [(64,64), (128, 128), (256, 256, 256)] for each of the Conv layers in the above model. The Pool layer performs Max-Pooling using pool sizes of [2,2] and reduces the output size to half of the input size. The Hidden Layers has dimensions 4096 each, the Sofmax has the output dimension of 10. We train the model on the Cifar-10 image dataset, using Logistic Regression as the loss function and Adam as the Optimizer. We train the model untill we reach 18% classification error, and save the weights of the Convolution layers. We also train a more Complex model with this architecture [(Conv, Conv, Pool), (Conv, Conv, Pool), (Conv, Conv, Conv, Pool), (Conv, Conv, Conv, Pool), (Conv, Conv, Conv, Pool) (Hidden, Hidden), (Softmax)], where the kernel and pool sizes remain the same, and the number of kernels are [(64, 64), (128,128), (256,256,256), (512, 512, 512), (512, 512, 512)] for the Convolution layers. The Pool Layers perform Max Pooling keeping the output size same as the input. The whole structure can be see Figure 2, The Convolution Layers do not use Zero Padding, and hence reduce the size of the image by (kernel size-1). The rest of the details remain the same. However we still found that it is very difficult to identify style of such a small image.

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**
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**
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**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 2: The whole structure of VGG-16

B. Technical Challenges

There are two main challenges. First, our own trained model's the input size is fixed as [7, 8] image and cannot be changed. It is very difficult to identify style of such a small image, with 3 Convolution-Pool Layers. Second, we trained our network on Amazon Web Service, with GPU 2.2x large, while the hardware available to the original

authors, TITAN Black, was significantly more powerful than ours. It might take us for months to train the network, but we believe the ultimate goal of the project is to think deeply about the paper, so we choose to further use the existing VGG16 parameters online. In Figure 3. On the very top are the original images, below follow the last layer features extracted by our network. It shows that our trained network doesn't extract the features well.

Another difficult is that to achieve good results using neural style transfer in practice today, users must pay particular attention to the composition and/or style image selection, or risk seeking unpredictable and incorrect patterns. For portraits, facial features can be ruined by incursions of background colors or clothing texture, and for landscapes pieces of vegetation may be found in the sky or other incoherent places. There's certainly a place for this kind of glitch art, but many users become discouraged not being able to get results they want.

IV. Datasets

We use CIFAR-10 to train our VGG-16 network. For pre-trained VGG-16, the weights are from ImageNet. And all images are from google image.

V. Results

A summary of our model performances can be divided to two parts according to two different papers.

A. Neural Style

Intuitively, we believe that a robust and powerful CNNs architecture is the key starting point of style transfer. Therefore, in order to get a strong feature extractor, we constructed a deep network. The details are provided as above. Then, we trained our model on CIFAR10 for 12 hours, applied it to extract style features from each layer, and content feature from only the third layer. Since CIFAR10 only provides 32-by-32 images, so only 32-by-32 images can be integrated to our framework. We tried a few images, it turned out that quality of the original images were not good enough. The result in Figure 3 was not clear enough. What's more, the loss in Figure 4 didn't decrease after 10 iterations. We suspected that it may be caused by poor quality of our image. Therefore, we want to load weights provided by Imagenet. However, there were no existing parameters in ImageNet provided to Theano model. For this reason, we decide to choose VGG-16 from Keras in our model, and implement other thing with Theano.

We decide to use the pre-trained model in keras. In general, the accuracy of the models were higher than our own VGG. Here we use L-BFGS, which is worked best for image synthesis. The famous Adam optimization algorithm is not quite for this task. Since at each iteration, the loss function decreases much slower than L-BFGS. For example change the rate of $\frac{style\ weight}{content\ weight}$. We get different images. Just as

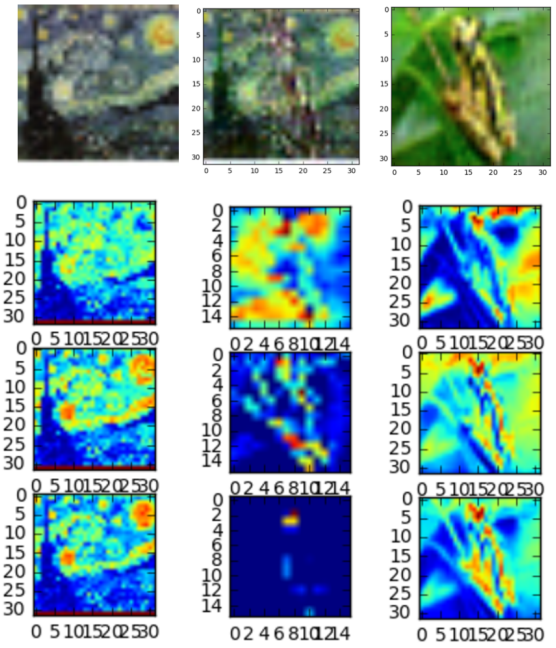

Figure 3: Result from self-trained VGG-16 net work. We trained our own model 12 hours, but because the limitation of image size and quality, the ability to extract image feature is weak. This is the last layer's feature. The middle is a combined image. The feature is not clear enough

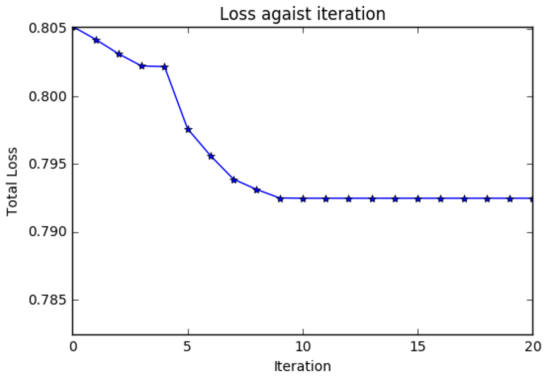

Figure 4: Total loss against the iteration. The loss stop decreasing at 10-th iteration. Because the quality of image

mentioned in original paper, the images shown in Figure 5 and Figure 6 were synthesized by matching the content representation on layer conv4-2 and the style representation on layers Conv1-1, Conv2-1, Conv3-1, Conv4-1 and Conv5-1.

The result shows that a pre-trained VGG-16 has a very strong ability to extract feature of image. We then turn to study it's ability to extract feature. We study the feature from conv1-1, conv2-1, conv3-1, and conv4-1. As showed in Figure 7, Figure 8. The deeper the layer, the more abstract the feature is, but in the first two block VGG-16 can extract

Figure 5: Images that combine the content of a photograph with the style of several well-known artworks. The images were created by finding an image that simultaneously matches the content representation of the photograph and the style representation of the artwork with transformed rate $\frac{style\ weight}{content\ weight} = 1000$.

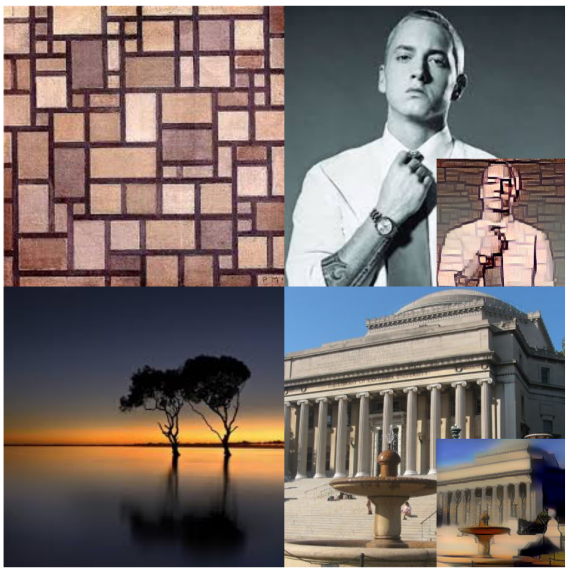

Figure 6: Images that combine the content of a photograph with the style of several well-known artworks. The images were created by finding an image that simultaneously matches the content representation of the photograph and the style representation of the artwork with transformed rate $\frac{style\ weight}{content\ weight} = 5000$.

the feature from image very accurate. On the other hand, because this model can manipulate images 400×400 size. The quality of images is higher.

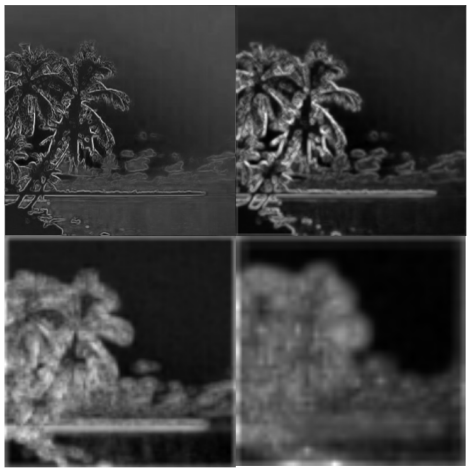

Figure 7: Feature coco tree in conv1-1, conv2-1, conv3-1 and conv4-1.

Figure 8: Feature of Columbia University in conv1-1, conv2-1, conv3-1, and conv4-1.

B. Neural Doodle

Another interesting result is from neural doodle which is a novel methods created by Alex J. Champandard. Instead using a Gram-based model, They choose a patch-based approach, by doing this, they algorithm doesn't struggle to provide reliable user controls to help address glitches. In their algorithm, Patches of k k are extracted from the semantic layers and denoted by the function Φ, For any patch i in the current image and layer l, K-means algorithm is adopted to choose the match patch, its nearest neighbor

$NN(i)$ is computed using normalized cross-correlation by taking into account weighted semantic map:

$$NN(i) = \arg\ \min \frac{\Phi_i(s)\Phi_j(s_s)}{|\Phi_i(s)||\Phi_j(s_s|}$$

Then The style error E_s between all the patches i of layer l in the current image to the closest style patch is defined as the sum of the Euclidean distances:

$$E_s(s, s_s) = \sum ||\Phi_i(s) - \Phi_{NN(i)}(s_s)||^2$$

To implement this mode we also using a pre-trained VGG-16 from keras and update the white noise image by L-BFGS algorithm, and implement a specific mode with 18 layers maxpooling to manipulate the mask matrix. The result is quite surprising, Figure 9, we can see that after only 5 iteration, we get a ideal images. Figure 10 show our final result. top left is the original image, top right is the origin image mask. We use bottom left mask as the content mask, and after 20 iteration, we get a ideal image as bottom right.

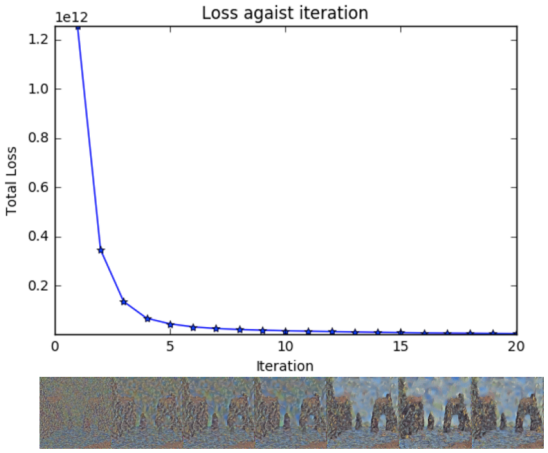

Figure 9: The loss and image of the generated process. After 10 iteration the loss becomes very small, and the white noise on generated image become negligible

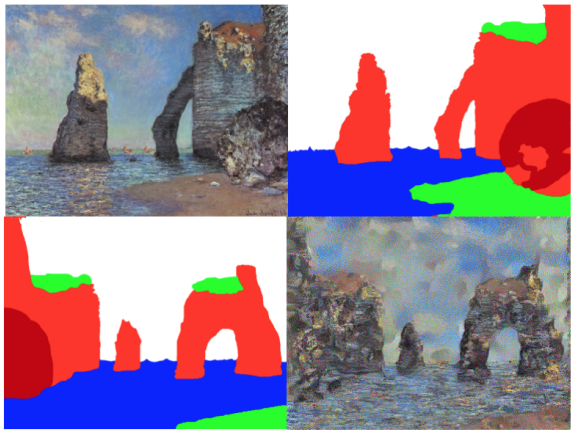

Figure 10: Deep image analogy for a Monet painting based on a doodle. Top left is the original image, the bottom right is generated image

VI. Conclusions and Future Work

In this project, we offer a comprehensive review of the principle neural style transfer algorithm, and carefully explained the idea of style loss and how it is computed. Then, we show the results from both the original paper and the Semantic Style Transfer and Turning Two-Bit Doodles into Fine Artwork (2016, Champandard et al). In the Result section, we showed the roadmap of our project, and compare the results with the original paper. By utilizing the VGG16 network, we successfully reproduce the results in both papers. The current style transfer algorithm mostly runs on images, because reproduce a style-transferred image may take a long time, and it is hard to implement on video. Although some newly published papers attempt to transfer one image style to the whole sequence of video [8], the only one style greatly limits the potential applications of style transfer. So in the future, we will further study on this.

VII. Contribution of each group

1) **Xikai Chen(xc2358):** Participation of two papers model building, lead the general code building and participate in final report. 1/3
2) **Yitong Xu(yx2376):** Participation of two papers model building, integrate all models and lead the final report and model selection. 1/3
3) **Samarth Tripathi(st3029):** Participation of two papers model building, build VGG-16 and Maxpooling model, lead in architect building. 1/3

REFERENCES

[1] Our bitbuket https://bitbucket.org/e_4040_ta/e4040_project_ggwp

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

[3] Alex J. Champandard, Semantic Style Transfer and Turning Two-Bit Doodles into Fine Artworks, *arXiv:1603.01768 [cs.CV]*, 2015.

[4] Manuel Ruder,Alexey Dosovitskiy,Thomas Brox, Artistic style transfer for videos, *Computer Vision and Pattern Recognition*, 2016.

[5] K. Simonyan and A. Zisserman. Very Deep Con- volutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556*, Sept. 2014.

[6] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs. *arXiv:1412.7062 [cs]*, Dec. 2014. arXiv: 1412.7062.

[7] M. Cimpoi, S. Maji, and A. Vedaldi. Deep filter banks for texture recognition and segmentation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recogni- tion*, pages 3828-3836, 2015.

[8] A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference*, volume 2, pages 1033?1038. IEEE, 1999.

[9] Long, E. Shelhamer, and T. Darrell. *Fully Convolutional Networks for Semantic Segmentation*. pages 3431?3440, 2015.

[10] N.Ashikhmin.Fasttexturetransfer. *IEEEComputerGraph- ics and Applications*, 23(4):38-43, July 2003.

[11] M. Berning, K. M. Boergens, and M. Helmstaedter. SegEM: Efficient Image Analysis for High-Resolution Connec- tomics.*Neuron*, 87(6):1193-1206, Sept. 2015.

[12] Fast neural doodle https://github.com/DmitryUlyanov/fast-neural-doodle

[13] Neural style transfer https://github.com/jcjohnson/neural-style