Artistic Style Transfer Using Neural Network

E4040.2016Fall.NASS.report

Guowei Xu gx2127, Huafeng Shi hs2917, Kexin Pei kp2699

Columbia University

***Abstract*—In this project, we focus on the works of transferring artistic style using the deep neural network. Specifically, we reproduce the result of the work "A Neural Algorithm of Artistic Style" by Gatys et al. [6], and its following-up work "Combining Markov Random Fields and Convolutional Neural Networks for Image Synthesis" by Li et al. [7] that uses Markov Random Field (MRF) prior.**

We present the introduction of these two papers and their methodology, the description of our implementation, and the detailed evaluation and result comparison in this report. We discover that there are both pros and cons of the first and second paper's technique. In general, the second paper outperforms by giving the better style-content transfer result while the first paper has much less runtime/memory overhead. On the other hand, we also discuss the limitations of the papers in our report.

I. Introduction

The area of artistic style transfer using the deep convolutional neural network (CNN) has recently received much attention [6], [7]. Gatys et al. [6] introduces an artificial system based on convolutional neural networks (CNN) to combine the content of an arbitrary photograph and the style of an artwork and renders new images in the style of the artwork while conserving the same content as the original photograph. The key finding here is that the representations of content and style in convolution neural network can be manipulated independently. That is, a strong emphasis on style or content will result in images that much resemble the artwork or the photograph. Consequently, by changing the ratio of the weightings of the content loss function and the style loss function, one can produce images that are more alike the artwork or the photograph.

However, their technique only leverages the Gram matrix to capture the global context constraints of the style image, which restricts the style sources to be non-photo-realistic art work. When deployed to transfer style information from the photographic image, the output appears to be unnatural. Li et al. [7] propose to use MRF regularizer that maintains local patterns of the "style" exemplar. As shown in Figure 1, the middle image is the style resources, which is a photographic image. The result from upper row (MRF prior) looks much better than the lower row (Gram matrix).

In this report, we review these two works: "A Neural Algorithm of Artistic Style" by Gatys et al. [6] and "Combining Markov Random Fields and Convolutional Neural Networks for Image Synthesis" by Li et al. [7] and reproduce the results of the paper. We implement our own version of CNN with the same structure VGG-Network [11] and create new mixtures of artworks and photographs in various content/style emphasis.

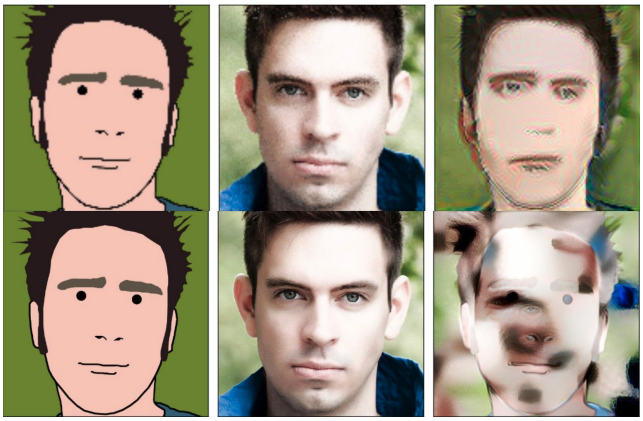

Fig. 1. Our result: A content image (left) is transferred into a painting (right) using the photographic image (middle) as the reference style. The upper one uses MRF constraints for style transfer while the lower one uses the Gram matrix.

The remainder of this report is organized as follows: Section II presents the summary of the methodology and key results of the original paper. Section III introduces the approach, methodology, and architecture used in our project. Section V shows our reproduced results and the several comparisons between our results and the original paper's, as well as our reproduced results between the first paper and the second paper. Section VI concludes our effort in this project. Section VII gives the online sources that we referred to when implementing the papers. Section A summarizes the contribution of each team member.

II. Summary of the Original Paper

A. Methodology of the Original Paper

We first give a brief explanation of how input is processed in each layer of the neural network. We then describe the general methodology of the first and the second paper, as they share the same idea at the high level.

Given a photograph and an artwork, a 19-layer VGG network is implemented to extract content representation from the photograph and style representation from the artwork. Each layer has multiple filters to decode input into several feature maps. Further, by reconstructing the image only from the feature maps in each single layer, we can directly visualize

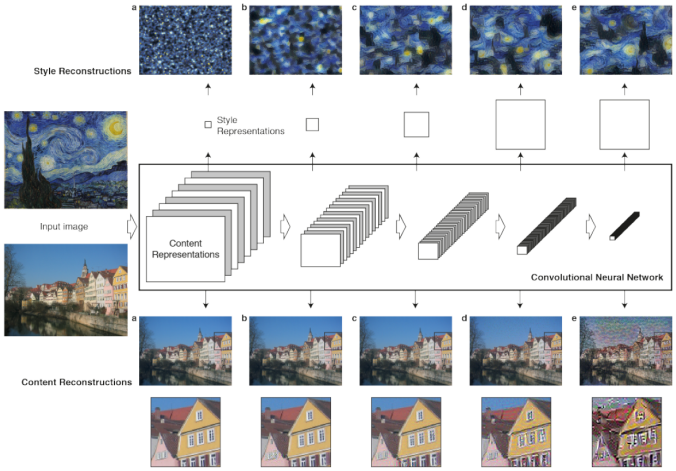

Fig. 2. **Convolutional Neural Network (CNN)**. A given input image is represented as a set of filtered images at each processing stage in the CNN. While the number of different filters increases along the processing hierarchy, the size of the filtered images is reduced by some downsampling mechanism (e.g., max-pooling) leading to a decrease in the total number of units per layer of the network. Content Reconstructions. We can visualize the information at different processing stages in the CNN by reconstructing the input image from only knowing the network's responses in a particular layer. On top of the original CNN representations, we built a new feature space that captures the style of an input image. The style representation computes correlations between the different features in different layers of the CNN. This creates images that match the style of a given image on an increasing scale while discarding information of the global arrangement of the scene.

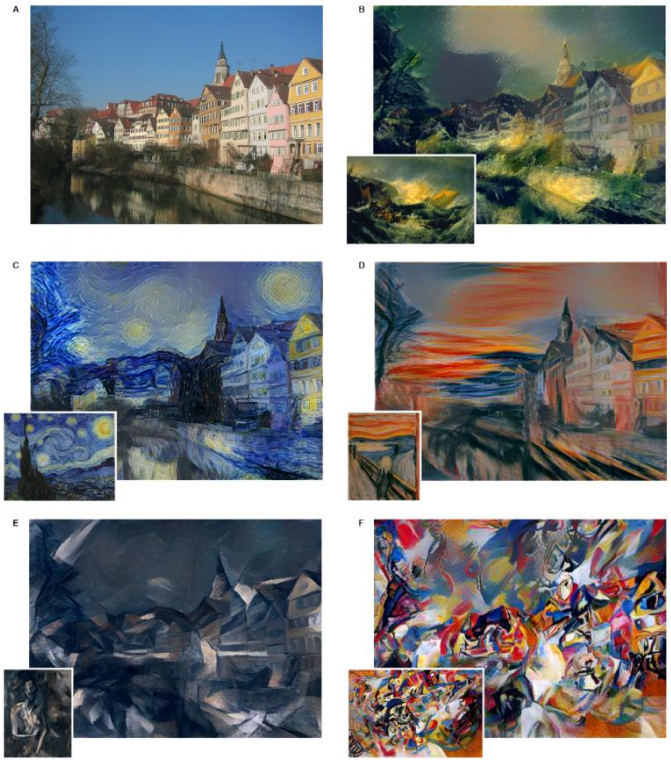

Fig. 3. Images that combine the content of a photo with the style of several well-known artworks.

the information of the input images. Figure 2 provides the flow of content/style image processed in each layer of the CNN.

To combine the artistic style and the content from different sources, the original papers (both the first and second paper) initialize the input I with random noise, and define the loss functions between (1) the feature map of the I and the content image, and (2) the feature map of the I and the style image. Then the two loss functions are jointly minimized by iteratively performing gradient descent on the I (fixing the parameters of the CNN while changing the pixel values of the I). Then I is modified to possess both the artistic style and the content from the two respective images.

The essential difference between the two papers that we implement is that the style loss function they define. The first paper leverages the Gram matrix to capture the style representation while the second paper makes use of the MRF prior to model the high-level image layout information, which they claimed to outperform the first paper in modeling the style representation.

B. Key Results of the Original Paper

1) First Paper: The key results of the first paper highlight that the representations of content and style in the CNN are separable. In particular, we can manipulate both representations independently to produce new, perceptually meaningful images. Figure 3 and 4, which mix the content and style representations from two different source images, illustrate this finding. In Figure 4, the rows show the result of matching the style representation of increasing subsets of the CNN layers (see Methods). Note that the local image structures captured by the style representation increase in size and complexity when including style features from higher layers of the network. This can be explained by the increasing receptive field sizes and feature complexity along the network's processing hierarchy. The columns show different relative weightings between the content and style reconstruction. The number above each column indicates the ratio α/β between the emphasis on matching the content of the photograph and the style of the artwork (see Methods).

2) Second Paper: Figure 5, 6, and 7 highlight some results of the second paper. As claimed by the author, for computing the style loss, they use a local constraint (MRF prior) that works for both non-photorealistic and photorealistic synthesis, while the authors of the first paper use the global constraint, which only works for non-photorealistic synthesis. Thus note that in Figure 6, the artistic image can be either a real face (the upper one) or an artistic face (the lower one).

III. Methodology

To perform the artistic transfer, the initial step is to pretrain a deep CNN that can effectively process different categories of images. We implement the VGG neural network [11], a deep CNN that rivals human performance on Imagenet — a common visual object recognition benchmark task [10]. Due to the extremely huge storage overhead to train the VGG by ourselves, we load the pretrained weight that is available

Fig. 4. Detailed results for the style of the painting

Fig. 5. A photo (left) is transferred into a painting (right) using Picasso's self portrait 1907 (middle) as the reference style. Notice important facial features, such as eyes and nose, are faithfully kept as those in the Picasso's painting.

online[1]. Then we implement the key loss function and L-BFGS optimizer — the second-order quasi-Newton methods [8], which outperforms the first-order such as stochastic gradient descent (SGD).

The main technical challenges are to implement the loss function defined by the paper. Given the neural network N with l layers, we define I_S as the style image input, I_G as the generated/transferred image initialized as random noise, and I_C as the content input. There are three types of loss functions:

(1) content loss, which defines the difference between the feature maps of N, given I_C and I_G as the input image.

(2) style loss, which defines the difference between the feature maps of N, given I_S and I_G as the input image. As noted in the previous section, the essential difference between

[1]https://s3.amazonaws.com/lasagne/recipes/pretrained/imagenet/vgg19_normalized.pkl

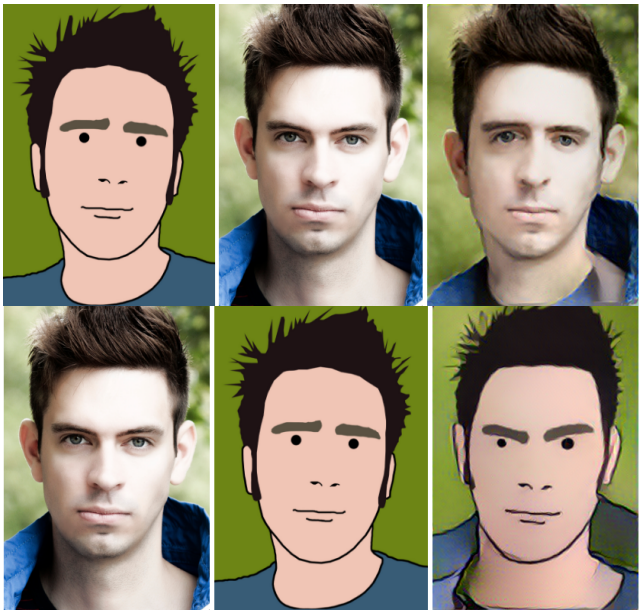

Fig. 6. In this example, we first transfer a cartoon into a photo. Then we swap the two inputs and transfer the photo into the cartoon.

Fig. 7. It is possible to balance the amount of content and the style in the result: pictures in the second column take more content, and pictures in the third column take more style.

the first and the second paper lies in the style loss function definition.

(3) A smooth loss function that regularizes the I_G, which we use the total variation loss [9].

Two papers both select some convolutional layers activation feature map as the input to the loss function.

Formally, given a CNN, let layer l has N_l distinct filters, the response is a matrix $F^l \in R^{N_l \times M_l}$ where F^l_{ij} is the activation of the i^{th} filter at position j in layer l. Gradient descent is then performed on a white noise image to find another image that matches the feature responses of the original image. Note that L_C and L_G are the content and the generated images and P and F their respective feature representation in layer l. The content loss is defined as:

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

The gradient with respect to the input image I_G can be computed using standard error back-propagation. Thus we can change the initial random image I_G until it reaches the identical response in a certain layer of the CNN as the original image p.

On top of the CNN responses in each layer, a style representation is built to compute the correlations between the different filter responses. In the first paper, these style representation are given by Gram matrix $G_{i,j}^l \in R^{N_l \times N_l}$, where $G_{i,j}^l$ is the inner product between the vectorised feature map i and j in layer l:

$$G_{i,j}^l = \sum_k F_{i,k}^l F_{j,k}^l \quad (2)$$

Let A_l and G_l be the corresponding style representation response of of the input I_G and the I_S, the style loss function is thus defined as:

$$L_{style} = \frac{1}{4N_l^2 M_l^2}\sum_{i,j}(G_{ij}^l - A_{ij}^l)^2 \quad (3)$$

On the other hand, the second paper defines the different style loss function by using the MRF regularizer, where the loss function is an energy function:

$$\begin{aligned} L_{style}(I_S, I_G) &= E_s(\Phi(I_S), \Phi(I_G)) \\ &= \sum_{i=1}^{m} \|\Psi_i(\Phi(I_G)) - \Psi_{NN(i)}(\Phi(I_S))\|^2 \end{aligned} \quad (4)$$

where $\Psi(\Phi(I_G))$ denote the list of all local patches extracted from $\Phi(I_G)$ — a specified set of feature maps of I_G. Each "neural patch" is indexed as $\Psi_i(\Phi(I_G))$ of size $C \times k \times k$, where k is the width and the height of the patch, and C is the number of channel of the patch. Note that the index $NN(i)$ denotes the best match patches in the style patches, which "the best" is obtained by computing the normalized cross-correlation between $\Psi(\Phi(I_G))$ and $\Psi(\Phi(I_S))$. We omit the formula here and let the interesting reader referred to the original paper [7].

To generate the images that mix the content of a photograph with the style of a painting, we jointly minimize the distance of a white noise image from the content representation of the photograph in one layer of the network and the style representation of the painting in a number of layers of the CNN:

$$L_{total}(I_S, I_C, I_G) = \alpha L_{content} + \beta L_{style} + \gamma \Upsilon(I_G) \quad (5)$$

where $\Upsilon(I_G)$ is the total variation loss [9], α, β, and γ are the weight variables that can be tuned to balance between different factors in the total loss.

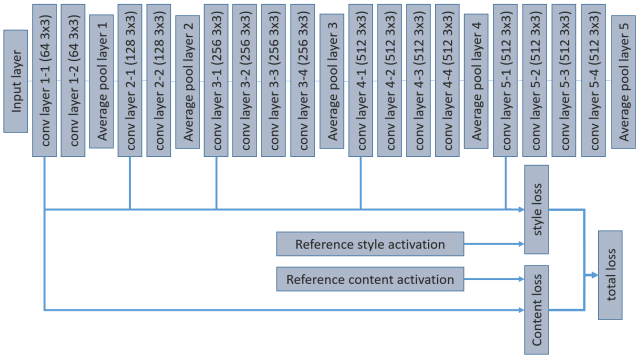

Fig. 8. network structure

IV. Implementation

We give our implementation details in this section. Most of our implementation of the first and the second paper share the similar architecture and the training algorithm, except the definition and implementation of the style loss function.

A. Deep Learning Network

1) Network Architecture: The network used to synthesis and combine two different pictures (content and style) is based on the VGG-19 network, which is pretrained for image recognition. It contains 16 convolution layers and 5 pooling layers. Our whole implementation is based on the Theano framework [2]. Figure 8 shows our implemented network architectures. Note that we choose average pooling instead of max pooling as the average pooling does not lose much information in the intermediate feature maps during the feed-forward process.

2) Style Loss Function: The implementation of style loss function of the first paper is trivial, as computing the gram matrix includes only one line in Theano by using the "theano.tensor.tensordot". For the second paper, as it needs to compute every generated image's best-cross-correlated neural style patch, we leverage the Theano's "conv2d" function with "filter_flip=False", which acts as an extra convolution layer. While the author claims this process can be "efficiently" implemented with an extra convolution layer, our result shows the overhead is still non-trivial. According to our result, the average running time of the second paper is 10 times longer than that of the first paper.

3) Training Algorithm: To minimize the loss functions, the generated image is iteratively modified by adjusting its pixel values according to the gradient of the loss on each pixel value. We test the two optimizers, namely RMSprop [12] and L-BFGS [8] in the first paper to show that the second-order L-BFGS outperforms first-order RMSprop w.r.t the runtime overhead. We thus use only L-BFGS in the second paper as the optimizer. We implement our own version of RMSprop

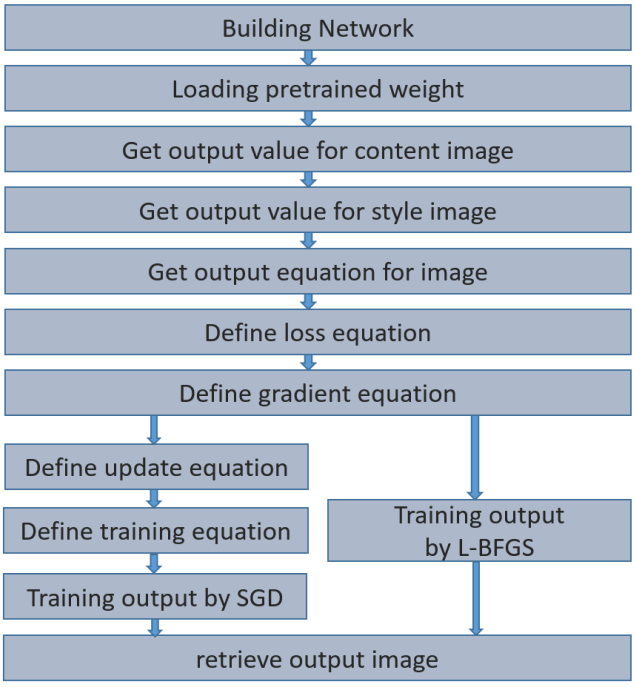

Fig. 9. Our software design working

optimizer in Theano while use the L-BFGS in Scipy[2] as our L-BFGS optimizer.

4) Dataset: Our dataset is based on the Imagenet's images. However, as mentioned in the previous section, due to the storage limitation in processing Imagenet dataset, we use the pretrained weights from VGG-19.

B. Software Design

The flowchart of our software design is shown in Figure 9. First, the network should be built for later processing. We implement three different layer classes (input layer, convolution layer, and pooling layer) based on Theano. Then the weight of pretrained weight is loaded into the network. The file contains the network weight is read and transformed into the numpy data format. Since the weight is shared variables in Theano, it can be set to given value by using "set_value" function. After that, the output values for both content image and style image is calculated. They are also the Theano shared variables, which is mapped to the memory space of GPU so as to access them efficiently. They are reference values, which take part in the calculation of total loss. Then the input variable is feed to the network to build the output. The loss function can be derived by output equation and referenced content and style value. The gradient can be easily computed by "Theano.tensor.grad". As discussed above, we compare two training algorithms using stochastic gradient descent and the L-BFGS algorithm. We use the RMSprop to accelerate the SGD training process. It should be noted that for implementing the second paper, we omit using SGD training as it is shown to have much higher runtime overhead than using L-BFGS when implementing the first paper.

[2]https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fmin_l_bfgs_b.html

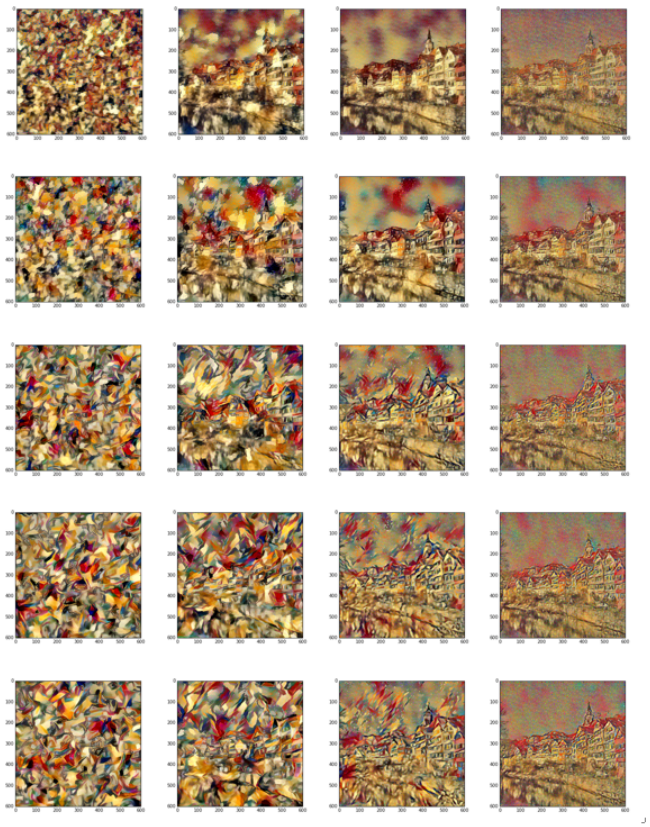

Fig. 10. Our reproduction of the results in paper 1.

V. Results

In this section, we discuss the project results, the comparison of results between our implementation and the original paper, and the discussion of insights we gained.

A. First Paper

Firstly, we reproduce the results in the first paper as shown in Figure 10 The columns show different weights between content and style reconstruction. The number above each column indicates the ratio α/β from 10^{-9}, 10^{-8}, 10^{-7} and 10^{-6}. The rows show the effect of choosing different style reconstruction layers from 'conv1_1' (A), 'conv1_1' and 'conv2_1' (B), 'conv1_1', 'conv2_1' and 'conv3_1' (C), 'conv1_1', 'conv2_1', 'conv3_1' and 'conv4_1' (D), 'conv1_1', 'conv2_1', 'conv3_1', 'conv4_1' and 'conv5_1' (E).

Then we use different photographs and artworks to create our own new images. We use Vincent Van Gogh's The Red Vineyard and a picture of a lighthouse (Figure 11) to demonstrate different ratios on content and style representation produce different new images.

As shown in Figure 12, the larger the ratio between content and style representation, the higher similarity between the

Fig. 11. (First row) The art source image and the content image.

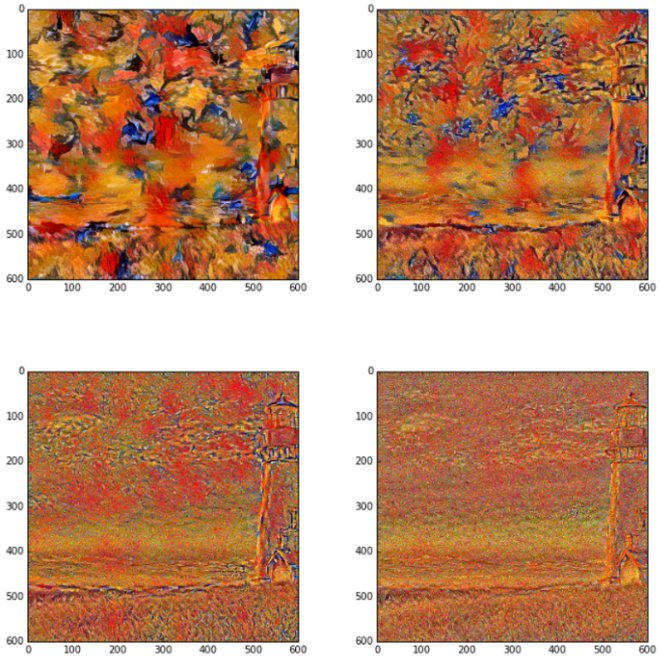

Fig. 12. Our results: from the top-right to the bottom-left, the ratios between content and style representation are 5×10^{-9}, 5×10^{-8}, 5×10^{-7}, and 5×10^{-6}. All images were generated using 'conv4_2' as the content layer and 'conv1_1', 'conv2_1', 'conv3_1', 'conv4_1', 'conv5_1' as the style layer.

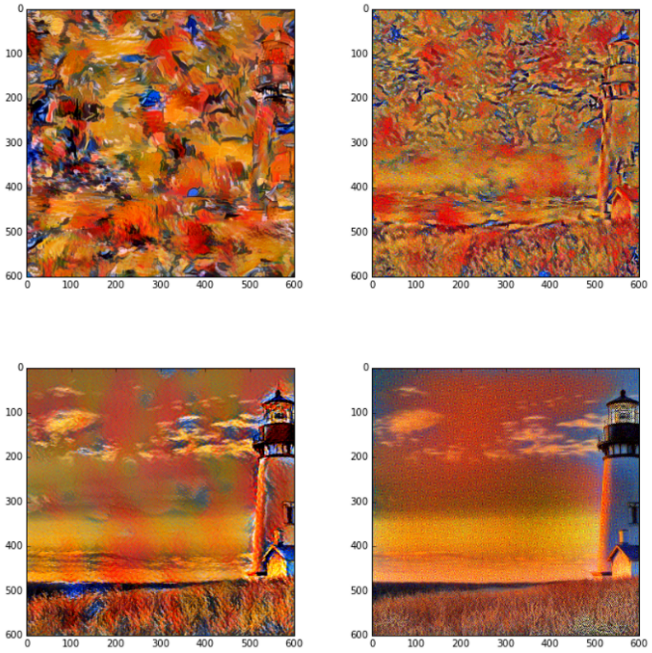

Fig. 13. Images generated using different content representation layers. From the top-left to the bottom-right, content representation layers are 'conv5_2', 'conv4_2', 'conv3_2', 'conv2_2', respectively. The ratio of the content and the style is 5×10^{-8}. We use 5 style layers: 'conv1_1', 'conv2_1', 'conv3_1', 'conv4_1', 'conv5_1'.

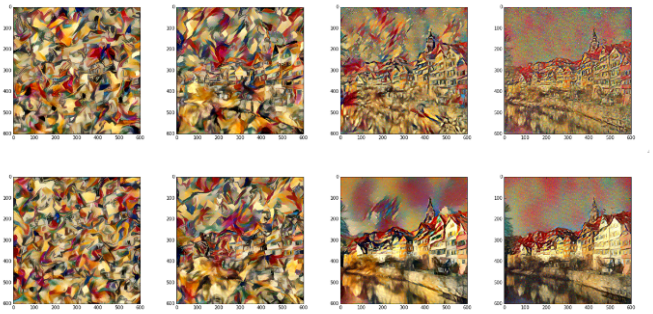

Fig. 14. L-BFGS (first Row) v.s SGD RMSprop optimization results (second row).

generated images and the photograph, which is in accordance with the results presented in the original paper.

Then we fix the ratio between content and style and chose different content layers to reconstruct images and discussed the results.

As show in Figure 13, when using lower content layers (the bottom two images), generated images conserve the content and pixel value of the photograph more while a high content layer is more about the global object arrangement. As stated in the original paper, "higher layers capture the high-level content regarding objects and their arrangement in the input image but do not constrain the exact pixel values of the reconstruction. In contrast, lower layers simply reproduce the exact pixel values of the original image". Our results validate this argument, and we believe this is a fairly interesting insight that we gained: the CNN structure resembles the human visual system that the visual layers near the eye process the lower level pixel/edge/corner information while visual layers near the brain process the higher level shape information.

Lastly, we also present the result of comparing different optimizer: SGD using RMSprop and the L-BFGS. As seen from Figure 14 and Table I, the apparent advantage of using L-BFGS has less runtime overhead. Therefore, we only use the L-BFGS when implementing the second paper.

B. Second Paper

The second paper implementation re-uses the similar data structure and logic design as the first paper, but we re-implement the style loss function by leveraging the Theano's conv2d to compute the per-neural-patch normalized cross-correlation more efficiently.

TABLE I
RUNTIME COMPARISON OF L-BFGS AND RMSPROP OPTIMIZATION, THE IMAGES ARE FROM FIGURE I, THE NUMBERS IN THE TABLE REFER TO THE RUNNING TIME IN SECONDS

	Image 1	**Image 2**	**Image 3**	**Image 4**
Scipy	163.5	166.2	163.6	167.2
RMSprop	799.2	797.1	794.6	796.2

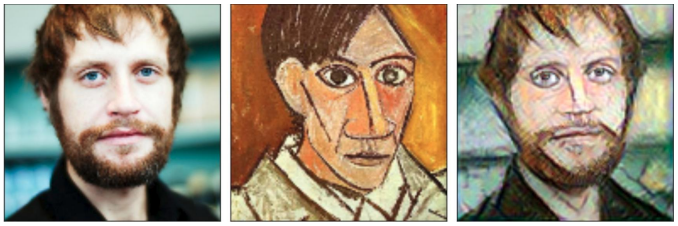

Fig. 15. Our result as with Figure 5

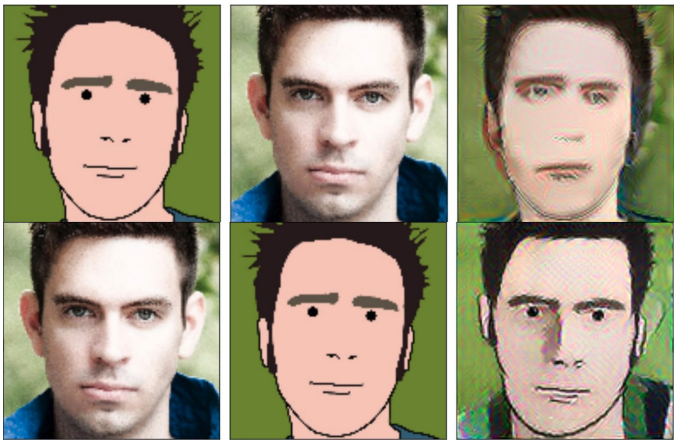

Fig. 16. Our result as with Figure 6

As claimed by the authors, this paper achieves better performance and partially resolves the limitations of the first paper. Thus in this section, we focus more on the comparison between our results of the first paper and the second paper. We leverage the images provided by the authors' released open-source implementation (in Torch) [5]. Then we run our implemented version of the second paper on these images, and we also show the running result of the first paper's implementation on these images. Finally, we summarize the differences we found and the insight we gained.

The result comparison between our implemented version and the original version can be compared to the figures shown in this section and those shown in Section II-B2. As shown later, our result of the second paper achieves near-identical output as in the original paper.

Note that our comparison of the first paper and the second paper in this section was never explicitly conducted by either the original first paper or the second paper. Thus we gained some first-hand interesting insights, which are shown in the following discussion.

Figure 15, 16, and 17 show the result of our implemented

Fig. 17. Our result as with Figure 7, the upper two rows have the ratio of α/β $2e1$ and $0.66e1$, respectively, and the lower two rows have the ratio of α/β $2e1$ and $4e0$, with all other parameters fixed.

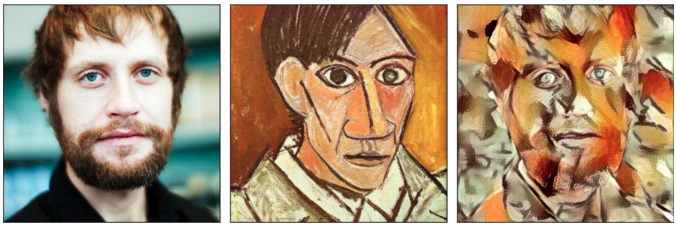

Fig. 18. The first paper's result when taking Figure 15 as the input

version's result, which are very similar to the results shown in the Section II-B2.

In the following, we show the results of the first paper by feeding the same input.

Figure 18, 19, and 20 presents the result using the first paper's technique, which uses the Gram matrix to capture the global texture information as the style constraints. The results are generally not as ideal as the second paper's. This validates that using MRF prior models the style information more accurately than using Gram matrix. However, there is limitation of using MRF prior, as claimed by the author: "it is more restricted to the input data: it only works if the content image can be re-assembled by the MRFs in the style image." For example, images of strong perspective or structure

Fig. 19. The first paper's result when taking Figure 16 as the input

Fig. 20. The first paper's result when taking Figure 17 as the input

Fig. 21. The case that the first paper's technique has better result (lower row) than the second paper's (upper row).

difference are not suitable for using MRF prior. Notice in Figure 21, the upper row uses the MRF prior while the lower row uses the Gram matrix as the style constraints. The content and style images are of strong structure difference and thus the result from the first paper's technique is obviously better than the second paper's.

Despite the better performance in the artistic transfer, the second paper's technique has much higher runtime and memory overhead than the first paper's technique. The runtime overhead is the need to compute per-neural-patch best matching style-patch. The memory overhead is due to the cost to maintain a large number of patches of the feature maps, especially when the input image has high resolution.

TABLE II
RUNTIME COMPARISON OF THE FIRST PAPER AND THE SECOND PAPER OF THE FIGURES 18, 19, 20, AND 21, THE NUMBERS IN THE TABLE REFER TO THE RUNNING TIME IN SECONDS

	Image 1	**Image 2**	**Image 3**	**Image 4**	**Image 5**
First paper	163.5	166.2	163.6	167.2	162.4
Second paper	1826.1	1799.2	1847.6	1810.1	1885.6

Table II compares the runtime overhead between the first paper and the second paper on the showed input images. The second paper's technique has 10 times higher overhead. In the future work, we may explore the direction to find the best point that balances this trade-off between the two techniques.

C. Discussion of Insights Gained

There are two major insights we gained during the implementation: (1) Higher layers capture the high-level content in terms of objects and their arrangement in the input image but do not constrain the exact pixel values of the reconstruction. In contrast, lower layers simply reproduce the exact pixel values of the original image. (2) The MRF prior generally achieves better results than the Gram matrix style constraints, but with higher overhead and does not perform as good when feeding with inputs (content and style sources) that are strongly different in image structure.

VI. Conclusion

We implemented two of the most representative works in the area of synthesizing content by examples using the neural network. In particular, we studied the specific problem of transferring artistic style from one image to another image, namely, combining the artistic style and actual content from two different images using the VGG network.

We achieved near-identical results shown in the original papers. Besides, we gained valuable insight and knowledge of architecture and the detailed internal computation performed by the deep convolutional neural network. During the implementation, we are more aware of the potential limitation of the application of neural network in artistic style transfer, and it also triggered our further interests in exploring the possibility of the future work, such as using the recurrent neural network (RNN) for the art style transfer.

VII. Acknowledgment

We referred to the open-source implementations [4], [5] while implementing our own version. However, note that the open-source implementations use the Lasagne [1] and Torch [3], while we implement the papers using the Theano framework [2]. Our implementation efforts are thus nontrivial due to the fundamental programming language differences.

References

[1] A lightweight library to build and train neural networks in Theano. https://lasagne.readthedocs.io/en/latest/.
[2] A numerical computation library for Python. http://deeplearning.net/software/theano/.

[3] A scientific computing framework for LuaJIT. https://github.com/torch/torch7.
[4] Art style transfer - Lasagne implementation. https://github.com/Lasagne/Recipes/blob/master/examples/styletransfer/Art%20Style%20Transfer.ipynb.
[5] Code for paper "Combining Markov Random Fields and Convolutional Neural Networks for Image Synthesis". https://github.com/chuanli11/CNNMRF.
[6] L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.
[7] C. Li and M. Wand. Combining markov random fields and convolutional neural networks for image synthesis. *arXiv preprint arXiv:1601.04589*, 2016.
[8] D. C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.
[9] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In *2015 IEEE conference on computer vision and pattern recognition (CVPR)*, pages 5188–5196. IEEE, 2015.
[10] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.
[11] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
[12] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4(2), 2012.

APPENDIX

The contribution of our team member is summarized in the following table:

TABLE III
CONTRIBUTION OF THE TEAM MEMBERS

	gx2127	**hs2917**	**kp2699**
Last Name	Xu	Shi	Pei
Fraction of (useful) total contribution	1/3	1/3	1/3
What I did 1	Revised codes into testing wraps	Build the network of first paper	Take the full responsibility of the second paper, including implementation and evaluation
What I did 2	Tuned parameters for all outcome of the first paper	Train the result by both scipy and SGD	Write the report (anything related to the second paper)
What I did 3	Wrote the report (Section 1, 2 and 5.1, 5.2)	Write implementation detail of the first paper	Write the abstract, conclusion, and take care of all the report re-structure and formatting