Image Style Transfer Using Convolutional Neural Networks

Apurva Modi, Maruf Sakib, Lalita Sharkey

Department of Computer Science, Old Dominion University

December 11, 2019

1 Introduction

A style transfer is a process of modifying the style or texture of an image while still preserving its content. The overall concept of this project is passing an input image and a style image into pre-trained convectional neural network. The network using in this project is the VGG-19 network architecture. Once an image is passed into a network, the image will then be represented in term of feature maps. The texture from the painting and the content are extracted from the photograph both of that are in term of features of the VGG network. Finally the results from both style and content images are combined to form the new image that has the content of the photograph and the texture or style from the painting. In this project, our team implemented the style transfer algorithm using method suggesting by Gatys et al., 2016 in Image Style Transfer Using Convolutional Neural Networks [2]. Figure 1 depicts the big picture of the method using in the pape.

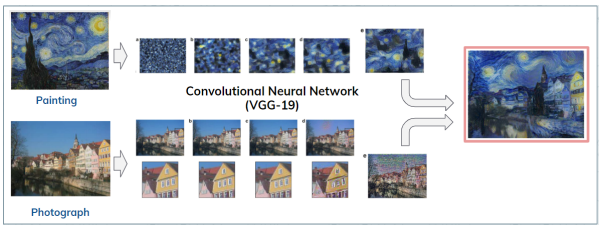

Figure 1: Big picture of the method using in the paper

2 Method using in the research paper

The goal of this project is to create the result image which has the contours of the content-image and the colours and texture of the style-image. According to the research paper, the style transferred image can be accomplished by creating several loss-functions that can be optimized. Figure 2 presents the overall concept of method using in the research paper. The loss-function for the content-image tries to minimize the difference between the features that are activated for the content-image and for the mixed-image, at one or more layers in the network. This causes the contours of the mixed-image to resemble those of the content-image.

3 Our Implementation

Our implementation is mostly based on the research paper. Figure 3 displays the full algorithm that is used as our implementation. There might be some adaptions or modifications due to not all needed information is mentioned in the paper.

3.1 Import the necessary packages and images

First of all, the necessary packages and the content and the style images are loaded. The packages that are used in our implementation are TensorFlow 1.x with Eager Execution enable. TensorFlow's eager

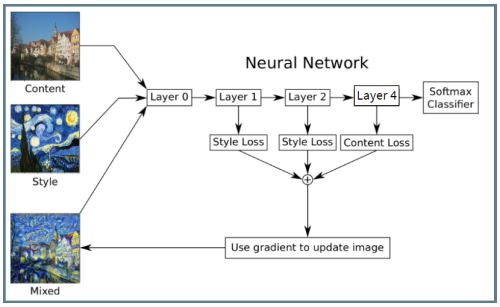

Figure 2: Image style transfer algorithm

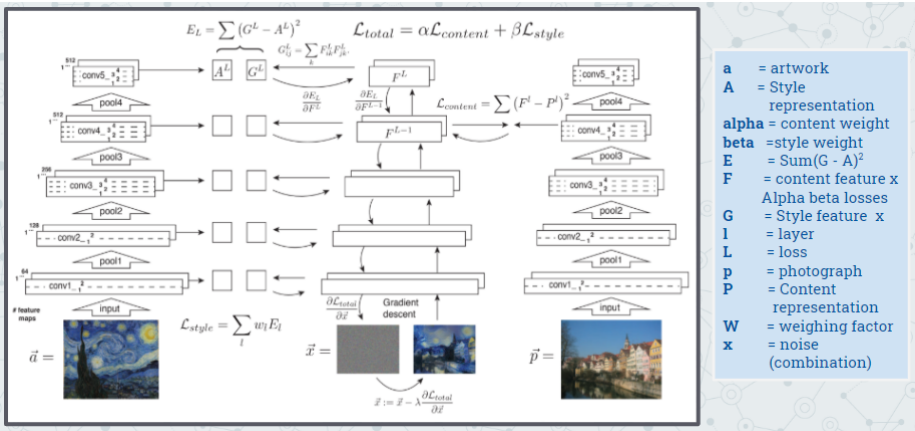

Figure 3: The grand scheme of our approach

execution is an imperative programming environment that evaluates operations immediately, without building graphs: operations return concrete values instead of constructing a computational graph to run later. This makes it easy to get started with TensorFlow and debug models, and it reduces boilerplate. Another important package that needs to be imported is the Tensorflow.keras.applications.vgg19 package. It is a pre-trained Convolutional Neural Network. The VGG networks are trained on image with each channel normalized by mean = [103.939, 116.779, 123.68] and with channels BGR. Therefore, both style image and input image need to be resized so that they are compatible for the pre-trained network. Both images need to be reprocessed as VGG input. Also, a target image needs initialized to produce white noise image with random pixel values.

3.2 Content extraction and content loss

The content features will be extracted from the second convolutional layer from the fourth block. The content features will be compared to the target image to measure the content loss. To construct the smooth stylistic image, the custom loss function will need to be optimized. Content Loss captures the root mean squared error between the activation produced by the generated image and the content image. The custom loss function consists of two different losses. The content loss is defined as

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

Where F_{ij}^{l} is the activation of the l th layer, i^{th} feature map, and j^{th} position obtained using the generated image. P_{ij}^{l} represents the activation of the l th layer, i^{th} feature map, and j^{th} position obtained using the noise image image.

3.3 Style Extraction and Style Loss

Style loss is extracted from the multiple first convolution layer includes the activation of conv_1_1, conv_2_1, conv_3_1, conv_4_1, and conv_5_1. The multiple layers that defined by the style loss are

used for learning multiple scale representation. On each layer included in the style representation, the element-wise mean squared difference between the noise feature and style representation is computed to obtain the style loss. The loss-function for the style-image is slightly more complicated, because it instead of trying to minimize the difference between the Gram-matrices for the style-image and the mixed-image. This is done at one or more layers in the network. The Gram-matrix measures which features are activated simultaneously in a given layer. Changing the mixed-image so that it mimics the activation patterns of the style-image causes the colour and texture to be transferred.Gram Matrix is relation between the filters(distribution) and is similar to correlation but in gram matrix we need not to subtract the mean value before multiplying it. Gram matrices allows detecting global repeating patterns-textures in image and be "blind" to local features. Gram matrix from activation of each layer is defined by:

$$G_{ij}^{l} = \sum_{k} F_{ij}^{l} F_{ij}^{l}$$

3.4 Total toss and termination

Total Loss derivative or the gradient with respect to the pixel values can be computed using error back-propagation and can be used as input for some numerical optimization strategy. Total loss can be computed by adding the weighted content loss (α) to the weighted style loss(βs).

$$totalLoss = \alpha L_{content} + \beta L_{style}$$

The optimization method using in the research paper is the L-BFGS algorithm. However, ADAM algorithm is used in our implementation. L-BFGS solver is is an optimization algorithm in the family of true quasi-Newton method that it estimates the curvature of the parameter space via an approximation of the Hessian. It has the downside of additional costs in performing a rank-two update to the Hessian approximation at every step. Similarly, ADAM is a first order method that attempts to compensate for the fact that it doesn't estimate the curvature by adapting the step-size in every dimension. In some sense, this is similar to constructing a diagonal Hessian at every step, but they do it cleverly by simply using past gradients. In this way it is still a first order method, though it has the benefit of acting as though it is second order. As depicted in figure 4, TensorFlow automatically derive the gradient for these loss-functions. The gradient is then used to update the mixed-image. This procedure is repeated a number of times until we are satisfied with the resulting image.

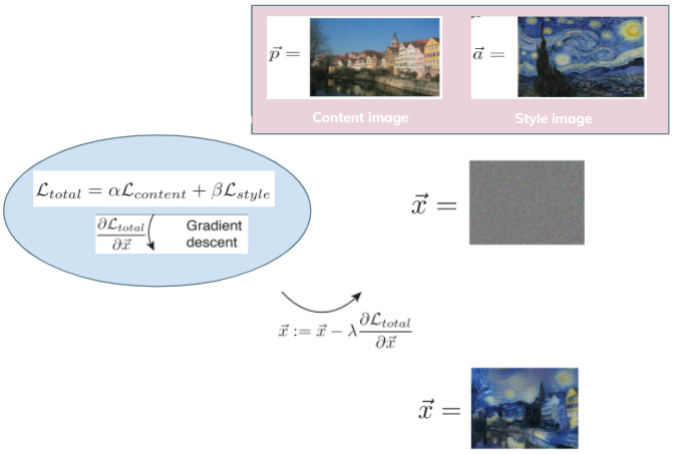

Figure 4: Update the target image using gradient descent

4 Dataset Description

The data used in this study was collected to replicate the use of the same dataset on the original research paper. As the original research paper did not provide the dataset to execute the algorithm on, an internet search was conducted to locate the same or most similar set of images in figure 5. In this study, the images used to create the results will be identified as the followings:

Content A: Neckarfront in Tubingen, Germany
Style B: Shipwreck of the Minotaur by J.M.W. Turner, 1805
Style C: The Starry Night by Vincent van Gogh, 1889
Style D: Der Schrei by Edvard Munch, 1893
Style E: Femme nue assise by Pablo Picasso, 1910
Style F: Composition VII by Wassily Kandinsky, 1913

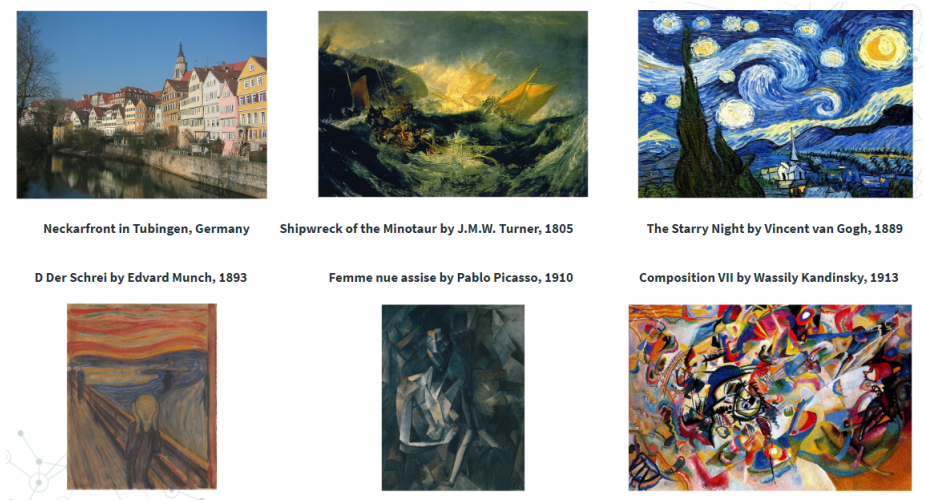

Figure 5: Dataset of image used in the project

5 Our results

The results for this study are visualized in this section with comparison to what the actual researchers got in their original paper. The first section discusses and compares the results between this study and the original study. The next subsection indicates to some points for causing the different results from the original study. The last subsection takes a quick look at the content and style loss metrics.

5.1 Results comparison

The first result comparison was done between this study and the original paper was for content A and style C in figure 6. The result of this study came out a little different where the style picture had more dominance than the content picture. The result from the original paper had less color influence from the style but imported some of the shapes as style from style C.

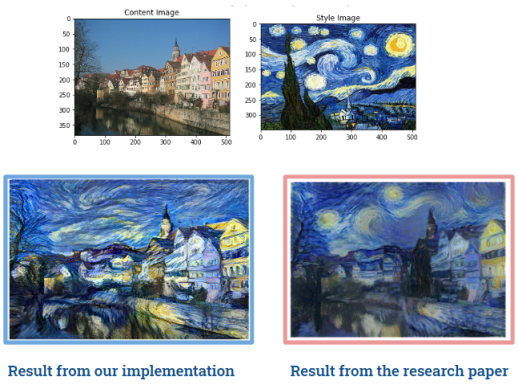

Figure 6: Results comparison

In figure 7, the top four and bottom four pictures set represent the results from the original study and our study respectively. From left to right, the images are generated from content A and style B, D,

E, and F respectively. The results generated from style B and D were more close to the original paper results than sty;e E and F. As, style E and F had more shapes in them, the algorithm approach in this study created somewhat different shape while importing the texture in the result.

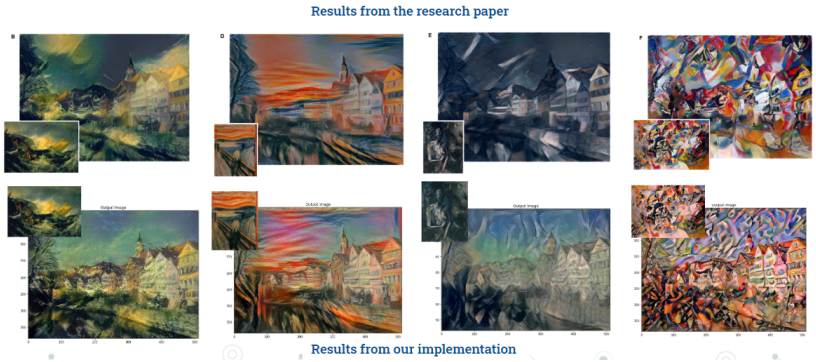

Figure 7: More results comparison

This study had the best visually similar result to the original paper while working with style transfer between two real images in figure 8. As the researchers, the same London in daytime as content and New York by night as style image was used where the result was very similar to the original research paper. Similarly, the content and style images were swapped and the result was a visually pleasing output from the combination. Thus, the algorithm used in this study seemed to work great with the realistic picture used in the original study.

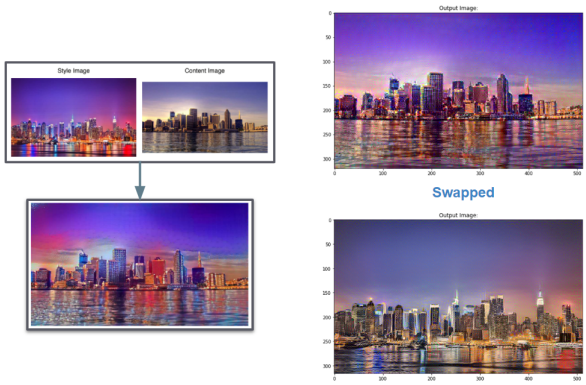

Figure 8: Realistic results comparison

5.2 Implementation result justification

In many cases, this study had somewhat more different results than the original paper. Several points contributed to such an occurrence of the results. Not all parameters were listed on the original research paper such as the number of iteration, learning rate etc. We randomly picked the numbers and tried with many different numbers to generate the results most visually close to the original paper. Moreover, the images used in our implementation were taken from Google as the images were not provided with the paper. As a result, the colors, clarity and even the shape of the images were different. More computational power is also needed so we can perform more optimization iterations with smaller step-sizes and for higher-resolution images. Last but not least more sophisticated optimization method is needed to be used and tested for better results.

5.3 Performance comparison

Among content and style image loss values, in all styles with content A, content loss seemed to be higher than the style loss. In figure 9, the style loss was higher at the beginning for style images C and E. But, at the end of 3000 iterations, style loss was always less than content loss in all cases. The reduction

rate of loss also seemed to stabilize after 1000 iterations. In all attempts, the style loss was around 500000 or below. The results with higher content loss like style E lost most of it's content shapes in comparison to others.

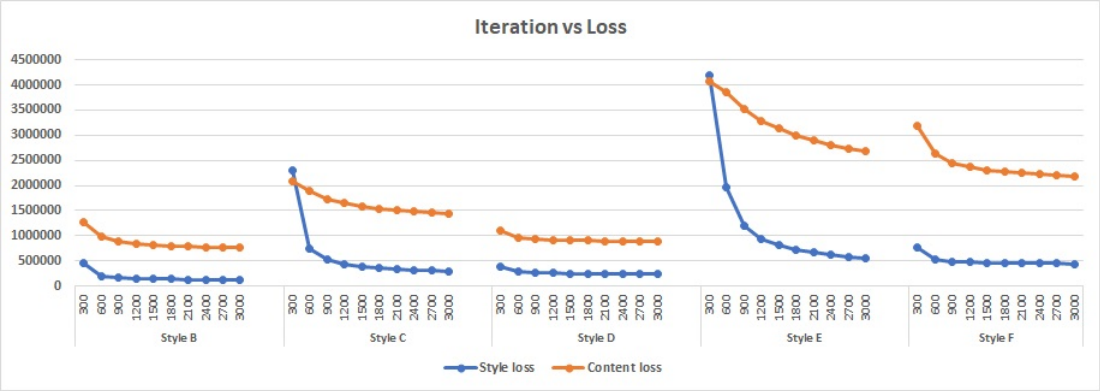

Figure 9: Content and style loss against number of iteration

6 Additional works

Our team did go beyond the requirement which implementing the algorithm which is used in the research paper. We tried and compare different approach on the same dataset and observe the results. We also run the our implemented image style transfer on the different images. The details of each additional works will be discussed in this section.

6.1 Comparing other relevant algorithm

Even though Gatys et al. image style transfer algorithm works well, it is quite slow. The possible reasons could be the neural network is used many time for both forward and backward propagation. Another possible reason could be that there are many optimization steps per new image. Johnson et al. (2016) [4] proposed a neural style transfer algorithm that is up to three orders of magnitude faster. Johnson et al. method bases on Gatys et al. image style transfer algorithm. Johnson et al. provide documentation and some pre-trained models on their GitHub [3].

We did not implement the method proposed by Johnson et al. (2016). However, we used the pre-trained models trained by them. According to their documentation, the method that they used is training feedforward neural networks that apply artistic styles to image. However, there is one downside of this method which is that you cannot arbitrarily select your style images as they are a pre-trained models for each style image. To illustrate, if you would like to transfer style from the Starry Night painting to a photo, you will need to use a starry night pre-trained model (`starry_night.t7`).

We use Python, OpenCV and the pre-trained feedforward models from Johnson et al. to generate the output images. Similar to Gatys et al. implementation, the input images are loaded and resized. The pre-trained neural style transfer also need to be loaded to the memory. After that, using cv2.dnn.blobFromImage package in OpenCV to construct a blob by performing mean subtraction follow by performing forward passing of the network. The output image will then need go though post-processing which includes reshaping the output tensor, adding back in the mean subtraction, and swapping the channel ordering. Lastly, the output of the neural style transfer process will be showed to the screen. The results comparison can be found in figure 10.

6.2 Apply the algorithm to additional Images

The goal of running our implementation on different dataset is to observe how well can the algorithm perform on images that was not apart of the images used on the research paper. Figure 11 illustrates one of the output images resulting from running it through the image style transfer algorithm implemented by our team. We select a picture resulting from transferring the style image of Norfolk by night to the image of Norfolk by day image. The output image does not look as what we anticipate, however it is only because we needed to spend more time finding the right parameters for style transferring.

Figure 10: Results comparison between OpenCV result and the result from the research paper

Figure 11: Using the implemented algorithm on additional images

7 Conclusion

The results from our team implementation version of image style transfer using the VGG-19 architecture convectional neural network consider to be satisfying. Even though we could not replicate the exact outcomes from the research paper, our output images are quite similar according to visual judgement given the fact that not all parameters was mentioned in the paper. Out method was also not completely the as what proposed in the research paper namely the optimization algorithm. Instead of using L-BFGS algorithm, we decided to use ADAM algorithm instead. According to an article Numerical Optimization: Understanding L-BFGS [1] and a journal Adam: A Method for Stochastic Optimization[5], ADAM algorithm should out perform L-BFGS. We also did some additional work where we compared Gatys et al. method and Johnson et al. method. Both method had different advantages and this advantages. If the performance is what critical then a method proposed by Johnson et al. is a better choice. Whereas if the need for flexibility for style image selection is more important then Gatys et al. method is more appropriate.

References

[1] Numerical optimization: Understanding l-bfgs, December 2014.

[2] L.A. Gatys, A.S. Ecker, and M. Bethge. *Image style transfer using convolutional neural networks.* CVPR, 2016.

[3] Jcjohnson. fast-neural-style, Jul 2017.

[4] Justin Johnson, Alexandre Alahi, and Fei-Fei Li. Perceptual losses for real-time style transfer and super-resolution. *CoRR*, abs/1603.08155, 2016.

[5] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. cite arxiv:1412.6980 Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.