COMP5425 Multimedia Retrieval

Group Assignment Final Report

YourStyle Application

Group: 14

liyu7088(480317999)
rlin5842(480509930)

Abstract

This report is about a Neural Style Transfer application, YourStyle, which implements an special optimization technique to combine two images. We input two images, a content image and a style reference image, the output has the look of the content image, but also showing the features of the style reference image. Details about the technical methods, implementation of neural style transfer and our motivation will be listed below.

1 Introduction

The goal of this project is to improve our skills to search online and learn new theory and technique to complete a project about media retrieval. After multiple team discussions and advices from our tutor, we decide to do Neural Style Transfer, and call our application YourStyle. Neural Style Transfer is an optimization technique for image, the content image will transfer to the style of the style reference image that we upload. The final output will remain the main content but with different style. In the report, we will detail our implementation of this project and the process for different steps and the results we have obtained. The first part will cover the statement of need. Then the next part is about the method we used to achieve this technique. Finally, we will show the result of output we obtained and discuss how to improve it better.

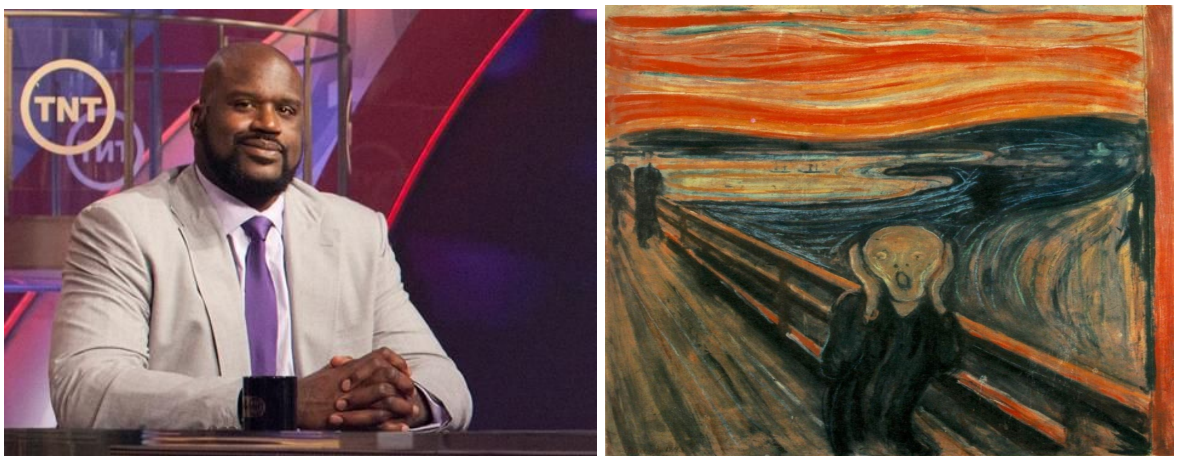

Figure 1: Content Image(Shaquille O'neal) and Style Image(*The Scream*, Edvard Munch)

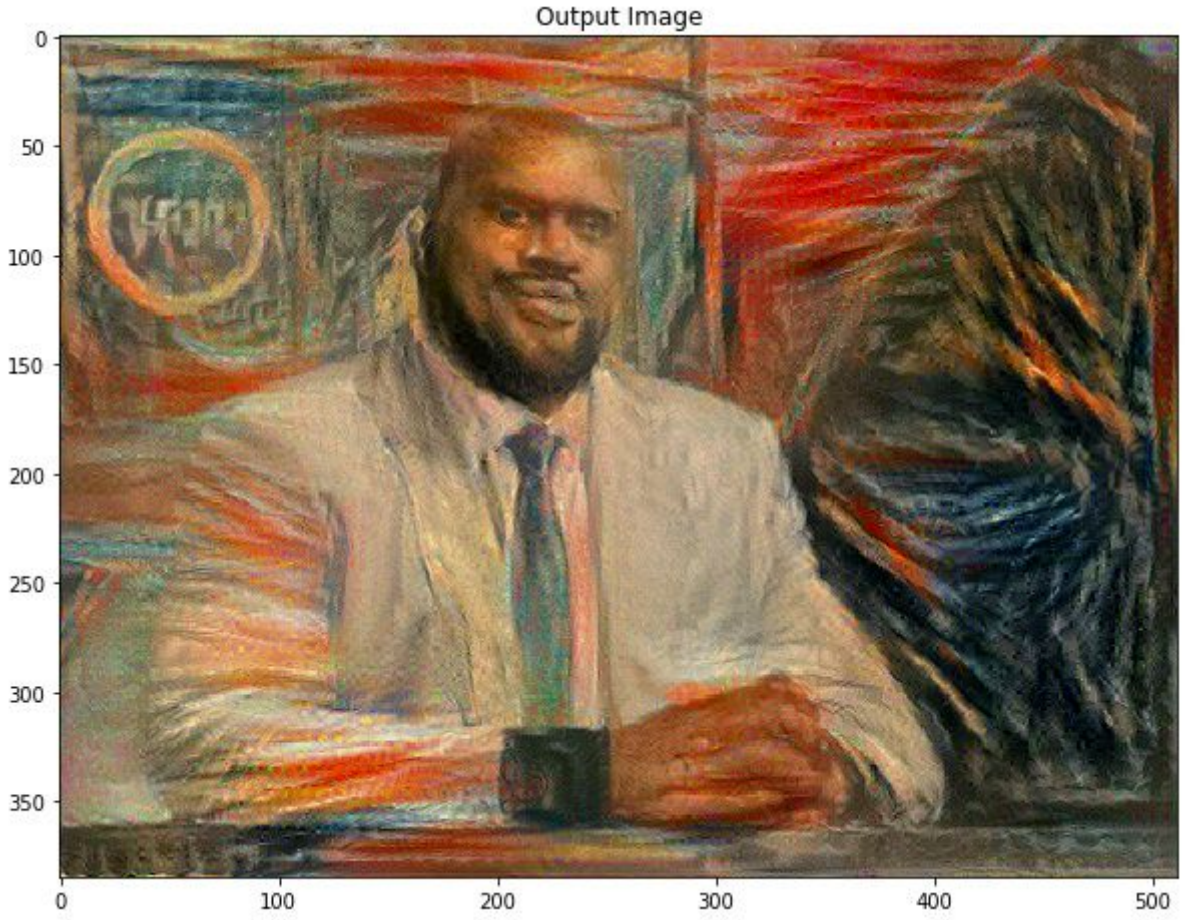

Figure 2: Generated image through neural style transfer process.

2 Business goals

YourStyle is targeted to convert an image into a different style. Different from simply adding a filter, our application uses deep learning methods to recreate the whole image, which possesses a better effect on style transfers, compared to adding filters. People can download pictures from websites or upload their own photos into the application, it will automatically create a new image combined with the content and style image. We hope to provide a new way for people to customize pictures and implement the results in commercial purposes. For example, people can use this to make protection cases for mobile phones with converted images printed on, or customize special clothings in their favor. Another aim for our application is that we hope to stimulate designers and artists on their creation, allowing them to have a quick visualization to any ideas in their minds, inspiring them on their art works. There are many similar applications out in market, but they only provide basic filters, not style transfer methods using Convolutional Neural Networks(CNN) to generate images. In our application, users can choose any style they like, and the result is pretty well, since CNN learns the features of image. Because our application has lots of computation, we recommend running our application on Google Colab, which has powerful GPU to reduce the runtime for computation.

3 Method

Pretrained VGG-19 is a convolutional neural network that is trained on more than a million images from the ImageNet database. It is capable of classifying 1000 objects, and the network has learned feature representations for a wide range of images. In our project, the pretrained VGG model performs well by understanding the image by extracting the features of the image from the feature map. Here we use the Keras Functional API to define our model to extract the image features, so we input our preprocessed image, and the output would be the generated features.

The total loss is weighted between the content loss and the style loss. In our model, we take 1 content layer(*'block5_con2'*), and 5 style layers(*'block1_conv1', 'block2_conv1', 'block3_conv1', 'block4_conv1', 'block5_conv1'*) from the VGG model. For the content loss, we simply calculate the euclidean distance between our initial content image, and the output image in the intermediate layer. The formula for content loss is defined as:

$$L^{l}_{content}(\mathbf{x}, p) = \sum_{i,j} F^{l}_{ij}(x)^2 - P^{l}_{ij}(p)^2$$

where (x, p) is the output image and the initial input content image, and (F,P) is the respective intermediate feature representations of the network at layer l.

Computing style loss is similar with the content loss, but we compare the distance between the output image and the initial style input image, not by calculating the raw output, but the correlations between the gram matrices. We define our gram matrice G_l, where G^{l}_{ij} is the inner product of i and j in layer l of the vectorized feature map. A_l is the output style representation matrice in layer l. We calculate the mean squared distance between the two feature correlation maps in each layer by

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

where N is the number of feature maps, M is the size, height*width. So the total style loss across each layer would be:

$$L_{style}(x, a) = \Sigma_{l \in L} w_l E_l$$

where (x, a) is the output image and the initial input style image. For the weight w_l, we simply consider each layer equally.

Now we combine the content loss and the style loss, by the number of layers among the total layers.

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

where (α, β) is the weight between content and style, in our project we use $(10^3, 10^{-2})$, which show the best result(Bethe, Ecker & Gatys, 2015).

After we have our loss function, we perform gradient descent to get the minimized loss. Here we use *tf.GradientTape* from tensorflow to calculate our gradient, and use Adam as our optimizer and run 1000 epoch, updating the image with the smallest loss.

4 Detailed Result

Through the process of our image, total loss decreases through the iterations, with our total generated loss between 10^5 to 10^7 on average. We show two tables of the statistics of the style transfer we tested in the appendix. Content loss starts at zero, since we compare with the original image itself, then style loss and content loss is

relatively equivalent, and summed up as the total loss. Total loss decreases gradually after each iteration, and finally comes to our optimized loss.

Figure 3: Other style image examples.(*The great wave off Kanagawa*, Katsushika Hokusai. *The weeping woman*, Pablo Picasso. *The starry night*, Vincent van Gogh.)

Figure 4: Generated image with *The great wave off Kanagawa*.

Figure 5: Generated image with *The weeping woman*.

Figure 6: Generated image with Starry night.

Here we show some more examples of different style images we tested with our application, shown in figure 3. Art pieces from different countries, time and art style perform pretty well through our application. We can see that the content does not lose its features, still capable of showing who the original person is, but showing the main features of the style image as well.

From figures 4,5,6, we can see the splatters of the wave, the twirling of the starry night, and the strong strokes from Picasso's art piece.

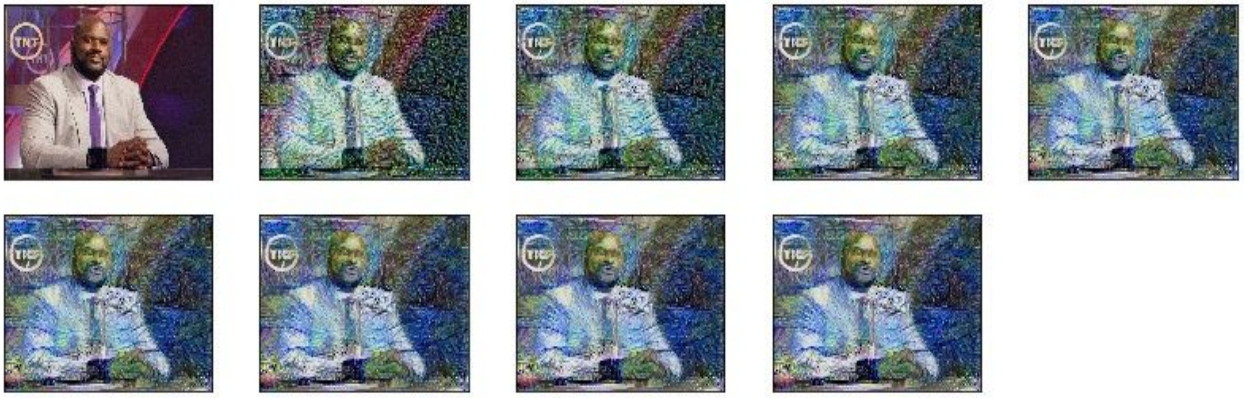

Figure 7: Images of the process of neural style transfer with *The great wave off Kanagawa , The weeping woman, and Starry night*.

In figure 7, we extract each image through our process, one image every 100 iterations. We can see the initial image change through the iterations, the waves form, the strokes becoming more and more strong and clear, and the twirls show.

Finally, for the running time of our application, we first ran our application on Jupyter Notebook with our own CPU, spending about four and a half hours to generate an image, then we switched to the GPU from Google Colab, which gives us a vast computing speed, with the average run time of 120 seconds.

5 Conclusion and future work

Although the result is quite good to the human eye, with a fine run-time, we still hope to make our application run even faster, since users would not have the patience to wait for two minutes for a photo. Less than 10 seconds would be our optimized running time. For the quality of the photo, we will try and work on the algorithms so that content images would not only be fitted for human faces, but every object. Another future work would be to design and create a more well-functioned and user friendly application, so end users can simply choose a photo from their phone, and choose options of styles they want to apply or upload their own style image, and get the results as fast as possible. For the methods, we will continue our research on

different methods of implementing neural style transfer, and take others research into consideration, choosing the most optimized method on generating the best image.

6 Reflction on preparing a presentation

In order to give an impressive presentation for our application, we did research on the internet, finding articles on key points, and main methods to present and clarify our product. We also sampled some videos on Youtube and TED, teaching people how to present. To summarize, the key concepts are to show passion, keep it simple and concentrate on the main concept, smile and eye contact, start strongly, use your body and voice tone. After understanding the concepts, we practice presenting with each other, hoping to give the best to the class. Some of the resources we sampled are listed in the references.

References

Bethge, Ecker & Gatys, 2015. A Neural Algorithm of Artistic Style. Retrieved from : https://arxiv.org/abs/1508.06576.

Alahi, Johnson & Li, 2016. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. Retrieved from: https://arxiv.org/abs/1603.08155.

Shen, Yang & Zeng, 2017. Meta Networks for Neural Style Tranfer. Retrieved from: https://arxiv.org/abs/1709.04111.

References for presentation skills:
https://www.skillsyouneed.com/present/presentation-tips.html

https://www.youtube.com/watch?v=dEDcc0aCjaA

https://www.ted.com/talks/nancy_duarte_the_secret_structure_of_great_talks?referrer=playlist-how_to_make_a_great_presentation

Appendix

Shaq with Picasso	**Shaq with MonaLisa**
Iterations: 0, Time: 0.1415 s Total loss: 4.2939e+08, Style loss: 4.2939e+08, Content loss: 0.0000e+00	Iterations: 0, Time: 0.1060 s Total loss: 5.6177e+07, Style loss: 5.6177e+07, Content loss: 0.0000e+00
Iterations: 100, Time: 0.1974 s Total loss: 9.2353e+06, Style loss: 7.2418e+06, Content loss: 1.9935e+06	Iterations: 100, Time: 0.0622 s Total loss: 2.4540e+06, Style loss: 1.4855e+06, Content loss: 9.6848e+05
Iterations: 200, Time: 0.2089 s Total loss: 4.5487e+06, Style loss: 3.0391e+06, Content loss: 1.5097e+06	Iterations: 200, Time: 0.0617 s Total loss: 1.1673e+06, Style loss: 5.5818e+05, Content loss: 6.0913e+05
Iterations: 300, Time: 0.2085 s Total loss: 2.8628e+06, Style loss: 1.6650e+06, Content loss: 1.1977e+06	Iterations: 300, Time: 0.0631 s Total loss: 7.3922e+05, Style loss: 3.1274e+05, Content loss: 4.2648e+05
Iterations: 400, Time: 0.1476 s Total loss: 2.1321e+06, Style loss: 1.1484e+06, Content loss: 9.8368e+05	Iterations: 400, Time: 0.0638 s Total loss: 5.5235e+05, Style loss: 2.2083e+05, Content loss: 3.3152e+05
Iterations: 500, Time: 0.1469 s Total loss: 1.7446e+06, Style loss: 9.0349e+05, Content loss: 8.4108e+05	Iterations: 500, Time: 0.0610 s Total loss: 4.5821e+05, Style loss: 1.7978e+05, Content loss: 2.7843e+05
Iterations: 600, Time: 0.2521 s Total loss: 1.5145e+06, Style loss: 7.7051e+05, Content loss: 7.4395e+05	Iterations: 600, Time: 0.0636 s Total loss: 4.0277e+05, Style loss: 1.5900e+05, Content loss: 2.4377e+05
Iterations: 700, Time: 0.2601 s Total loss: 1.3646e+06, Style loss: 6.8460e+05, Content loss: 6.8001e+05	Iterations: 700, Time: 0.0627 s Total loss: 3.6869e+05, Style loss: 1.4503e+05, Content loss: 2.2366e+05
Iterations: 800, Time: 0.3380 s Total loss: 1.2613e+06, Style loss: 6.2620e+05, Content loss: 6.3505e+05	Iterations: 800, Time: 0.0636 s Total loss: 3.4577e+05, Style loss: 1.3542e+05, Content loss: 2.1035e+05
Iterations: 900, Time: 0.3382 s Total loss: 1.1848e+06, Style loss: 5.8372e+05, Content loss: 6.0112e+05	Iterations: 900, Time: 0.0627 s Total loss: 3.2919e+05, Style loss: 1.2880e+05, Content loss: 2.0039e+05
Total time 124.9072 s	Total time 124.2888 s