Neural Style Transfer

Qingying Luo
Computer Science Department
Duke University
ql111@duke.edu

Abstract

Extracting style and content separately from images and combing them has always been a hot topic to study and research. Many of computer vision applications developed based on that. This report describes how I implemented and modified the neural style transfer network with the basic guideline from Gatys' paper. I experimented with different pairs of content and style, compared the result and finally put them back into VGG19 to get predicton.

1 Background

Convolutional neural network processes visual information hierarchically in a feed-forward manner, with different filters detecting for a variety of features. It will detect from low-level features such as edges, lines, to high-level features like a concrete object from shallow layers to deep layers. Taking use of this advanced technique, our goal here is to transfer style in one picture while also preserving its high-level semantic content. It can be used for transferring the style of a famous painting to a photograph, reproducing the art piece of certain well-known artists. Not only for the static image, this can be further explored in a real-time live video, with new style coming from other pictures blending in the moving objects and environment.

2 Method

I chose the pretrained network VGG19, which contains 16 layers and 1000 categories as outcome. This deep neural net learns in different scales, capture visual information in its layers and provides structural pixel information.

For the content extraction, I used conv4_2 layer to create my target content activation. Since higher layer will focus more on high-level features, like the actual objects of the image, instead of exact pixel values nearby. The content loss function is as below where F is the respective feature representation in layer l for white noise input, and P is for content target.

$$\mathcal{L}_{\text{content}}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} \left(F_{ij}^{l} - P_{ij}^{l}\right)^2 .$$

Figure 1

For the style extraction, I chose the layers conv1_1, conv2_1, conv3_1, conv4_1 and conv5_1 from low to high to have a comprehensive representation of the style. Then I calculated the correlation between different filter responses in each layer separately. The correlation, which is the dot product of different activation, can capture texture information and ignore the actual global arrangement. Since different layers have varied number of filters, to normalize and get a better result, I divided the

ECE 590-03 (2020 Fall) final report.

gram matrix by the number of elements inside. The style loss function is as below, where G is the gram matrix and A is the target.

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^l - A_{ij}^l\right)^2$$

$$\mathcal{L}_{\text{style}}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l,$$

Figure 2

The total loss function is Our goal here is to minimize the loss function. alpha is the weight associated

$$\mathcal{L}_{\text{total}}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{\text{content}}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{\text{style}}(\vec{a}, \vec{x})$$

Figure 3

with content, and beta associated with style. If we put more weights on beta, then when minimizing the loss function, the model will tend to reduce the error in style loss more, making the style blending effect more obvious compared to content effect. For this project, I experimented with a few from beta = 1x10e4, 1x10e5, to 1x10e6, and finally settled down with alpha = 1 and beta = 8 x 1e4 to get the best result.

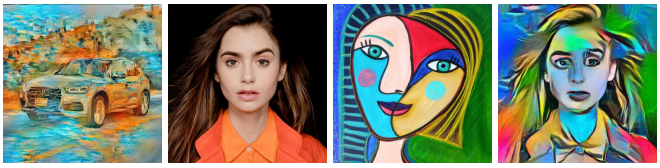
3 Experiment results

3.1 Test under a variety of content, style pairs

3.1.1 Good examples

The first column is the content. The second column is the style. The third column is the generated artificial outcome. The number of iterations below the third column of each generated output image represent how many back propagation it has been gone through.

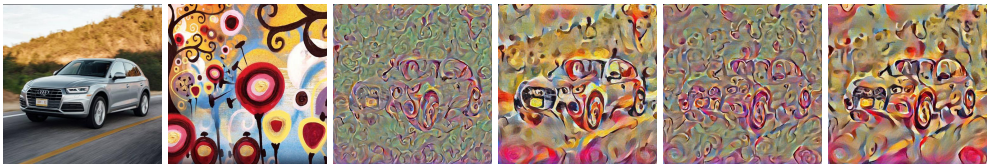
(a) Eiffel Tower (b) Starry Night (c) 4000 iter (d) Audi Q5 (e) Dock



(f) 4000 iter (g) Lily Collin (h) Pablo (i) 4000 iter

Figure 4: Good examples generated by our model

From Figure 4, we can see from (a), (b), (c), the generated image of Eiffel Tower has the starry elements in the sky. In (d), (e), (f), we have both blue water element and orange ship element reflected in the outcome. In (g), (h), (i), we used the style of a face to change the portrait content. We can still perceive the color pattern in the style and facial features of human in the content in our resulted outcome.



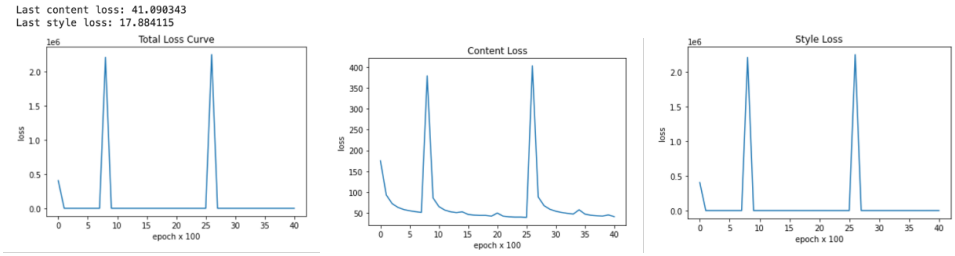
(a) Content (b) Style (c) 1000 iter (d) 2000 iter (e) 3000 iter (f) 4000 iter

Figure 5: Audi Car + Color Style

3.1.2 Bad examples

In Figure 5, (c) to (f) are four different output generated under different number of iterations. (d) and (f) look similar, with (d) having a clearer shape of the Audi car. This is probably the best the network could do, although we still cannot recognize the car quickly by only looking at the output images if we don't know the content beforehand.

I also plotted the loss curve on total loss, content loss and style loss, where each is defined in our method part.



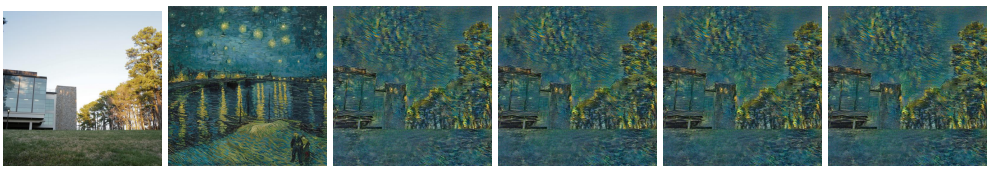
(a) Total Loss (b) Content Loss (c) Style Loss

Figure 6: Loss Curve on Audi Car + Colorful Style

From Figure 6, we can see there are huge increases near iteration 2000 and iteration before 3000. The corresponding generated images (c) and (e) contained so many noise due to this large loss increase and resulted in bad performance. Maybe the updates had been to drastic at some point and caused divergence from optimum. But it would decrease back, like in (f). So it is not training the more the better.

3.1.3 Evaluation

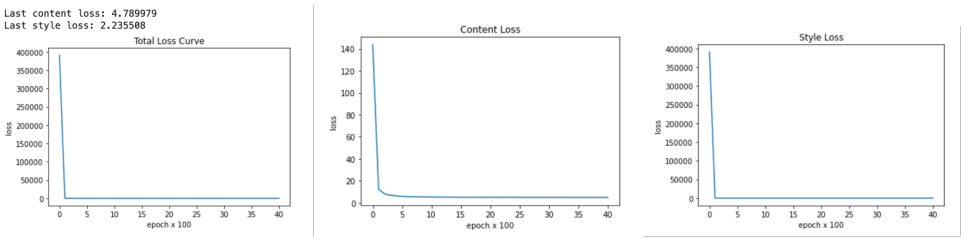
Unlike other supervised learning examples on image classification, where we can use the accuracy of test set to evaluate the success of model. The evaluation of whether the neural style transfer network is successful or not is hard to define. By looking into the generated artificial image itself, it involves lots of subjectivity. Different people may have different perspectives and tastes on whether the output manages to mix the content and style together, whether it is smooth and good-looking enough and so on.



(a) Content (b) Style (c) 1000 iter (d) 2000 iter (e) 3000 iter (f) 4000 iter

Figure 7: Duke Grass + Van Gogh

The above example shows a smooth and monotonic decreasing loss curve in Figure 8, but the resulting images in Figure 7 are not as expected. They only capture the color pattern but not the star element in



(a) Total Loss (b) Content Loss (c) Style Loss

Figure 8: Loss Curve on Duke Grass + Van Gogh Style

the sky. Maybe because there is yellow star reflection in the river so it is hard for the net to distinguish between the actual star and reflection, and only captures the yellow color element.

If we instead use the loss as the standard to evaluate, like in Figure 6, then we noticed that when loss increased, the performance became worsen. Indeed, the loss can reflect the degree of success to some extent, however, considering only loss is not enough.

To conclude, not only the loss is important for evaluation, our subjectivity also plays a critical role when we judge our results. A nice loss curve does not necessarily imply a nice-looking generated picture. Sometimes certain content and style images are not suitable for blending with others, no matter how well you train your model, like the colorful style in Figure 5, it performs bad in other content images also.

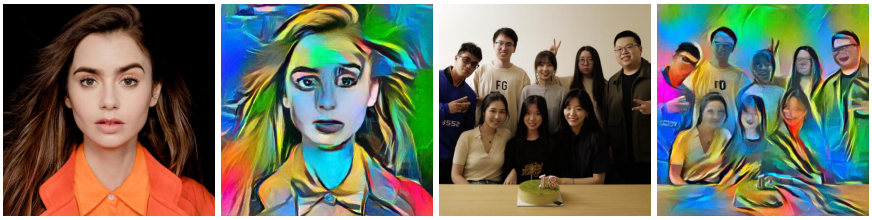
3.2 Test with different complexities

We now explore the how the complexity in image affect the generated output.

3.2.1 Same style with different complexities in content

I used the same style image on different content.

(a) Eiffel Tower (b) + Starry Night (c) Duke Chapel (d) + Starry Night



(e) Lily Collin (f) + Portrait Painting (g) People (h) + Portrait Painting

Figure 9

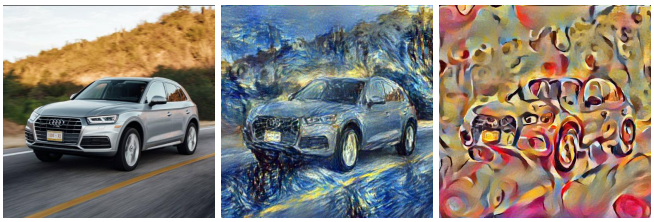
For (a), (b), (c), (d) in Figure 9, I used the Starry Night from Van Gogh and applied in both Eiffel Tower and Duke chapel. The content in (a) only had one main object in the field, and the background is clear. On the contrary, there are tree, grass, chair blocking in front of the Duke chapel in (c), making the content more complex compared to (a). When the neural net blended in the style, it could not detect the exact boundary of different boundaries, so the boundary was vague and the blending was not as smooth as in (b).

For (e), (f), (g), (h), when the content only contains a single person, the resulting image is clear and the features of person could still be detected. While there are more people in the content, the details will be cut off to a varying degree. (h) is better than (d) because nothing blocks in front of the person, so the boundary in (h) is still clear and makes the resulting image more nice-looking.

3.2.2 Same content with different complexities in style

Now we move on to variation in style on same content.

In the below Figure 10, I focused on Audi Q5 as content image. (b) is the best output with Starry Night, a style image with a suitable degree of complexity. (c) is the example I showed earlier in our bad example part, where the colorful style image contains many colors and high value pixels, so it is a complex style. For (e) and (c), the color and texture is not that obvious and strong as in (b), a simple style, so the output (e) and (g) only reproduce their color pattern, which is in a same color shade, making the output images look bland and insipid, not as vivid as in (b).



(a) Audi Q5 (b) + Starry Night (c) + Colorful Style

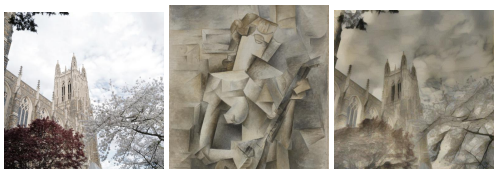
(d) Picasso Painting 1 (e) + Picasso 1 (f) Picasso Painting2 (g) + Painting 2

Figure 10

3.3 Different layers to optimize

If we used layers in the first part of our pretrained model VGG19 as our style target layers, say the first 5 convolutional layer, then the network can only capture the relative color change in the style image but not the detailed pixels on shape, edge, and pattern.

As in Figure 11 shown below, the network only captures the color shade in the Picasso painting and the output is like being simply applied with a color filter. So we only get the color shade from the style and no texture, edge, pattern information.



(a) Duke Building (b) Picasso Painting (c) + Picasso

Figure 11: Used first 5 conv to optimize style layers

3.4 Data augmentation with generated image

Since neural network is hungry for data, being in short of training data is an important problem to be solved. I tried to use generated artificial output images from my neural style transfer net to do the data augmentation. Here are some of the results when I input the generated images back to our pretrained VGG model.

In the below Figure 12, the first part below the image is prediction on content, and the second part is the prediction on the generated output. We can see VGG19 still takes style, texture, color into consideration. If the original content is clear and obvious, with object in the center and background clear, like (c), and the style does not dominate so much, then it can still identify the object correctly in the generated output. On the contrary, if the style dominates, like in (d), then VGG19 will classify based mostly on style information. The shower curtain prediction may be caused by the colorful elements in the style image. If the content image itself is not clear, like in (a) the Eiffel Tower is too small, and in (b) the image is complex, then VGG19 gets more information from the style and classifies the tweaked Starry Night element into the water snake, which is pretty reasonable. In (e), French Bulldog is only the third prediction returned by VGG19, then when in (f) the style blends in, the model may tend to consider this as paper towel because of the color.

(a) Beacon+Water Snake (b) Monastery+Water Snake (c) Minivan+Race Car

(d) Minivan+Shower Curtain (e) French Bulldog (f) Paper Towel

Figure 12: Prediction from VGG19

4 Conclusions

It is proven that CNN is able to well separate the content and style, and we are able to transfer the style into content successfully, using the layers in the pretrained model. However, in terms of optimizing the model, there are still lots of choices to experiment, like which layer to optimize, how many iterations to run, etc. And not all content and style images can produce the result you expect, no matter how well you train your model. We now do not have a generalized model and hyper-parameter setting that satisfies all images and produce state-of-art result. But we can still utilize this technique to reproduce famous artists' work and also apply it to further research on computer vision area.

References

[1] Gatys et al., *Image Style Transfer Using Convolutional Neural Networks*, `https://www.cv- foundation.org/openaccess/content`$_{c}vpr_{2}016/papers/Gatys_{I}mage_{S}tyle_{T}ransfer_{C}VPR_{2}016_{p}aper.p$

[2] Leon Gatys, leongatys/PytorchNeuralStyleTransfer
`https://github.com/leongatys/PytorchNeuralStyleTransfer/blob/master/NeuralStyleTransfer.ipynb`

[3] ProGamerGov/neural-style-pt
`https://github.com/ProGamerGov/neural-style-pt`

A Timeline and task allocation

Please provide your working timeline of the project in this section. I spent the first week reading papers on neural style transfer, understanding the theories, formulas, and techniques in them. Then another week on PyTorch tutorial, reading reference code, and started to run my code and trained a little bit. Lastly, I spend few days on fine-tuning my model, gathering the results, writing my report and cleaning up my code.

B (Optional) Implementation detail

Here are more subtle details of my implementation.

The image will be read into PIL object first, then it will be cropped to shape 512 x 512, be standardized into 0 and 1 range, and be normalized with mean = (0.4985, 0.456, 0.406), standard deviation = (0.229, 0.224, 0.225) for red, green, blue three channels separately to gather best performance in VGG19, which is argued to be the normalization for using VGG19. Following with Gatys' paper, I used optimizer with L-BFGS. After the loss is back propagated back to the input pixels and values get updated, the pixel values will be made to 0 if they are smaller than 0, and 1 if they are larger than 1, so that they are still in the range of 0 and 1. Otherwise the negative pixel values will produce noise in the image. We repeat the exact same procedure, get 4000 iterations of back propagation until the desired result is reached.