MAIS202 Final Project Deliverable 2: Artistic Style Transfer

Rosie Zhao

1 Problem Statement

Neural style transfer involves taking two input images to generate an image that retains the content of one while emulating the style of the other. From my initial proposal, I aimed to perform neural style transfer on landscape photos from Studio Ghibli films on some real-life photos.

2 Data Preprocessing

Since I'm using a pretrained VGG-19 Convolutional Neural Network (see next section for more information), the data that I selected and manipulated were simply my two image inputs; it was an easy implementation, because I just had to ensure that my images were of the same size, and they were sized reasonably to run on my computer. Other implementations of neural style transfer often have content and style images that don't correlate with each other (ex. a dog content image in the style of Van Gogh's *Starry Night*) but in my first attempt at implementing this model, I stuck with image pairings that were more related to each other; however, I'll mention later in this deliverable that the effect this has on the output image isn't as relevant as I initially thought.

3 Machine Learning Model

The model that I used follows Gatys' paper referenced in the last deliverable; as mentioned before, the VGG19 pretrained network was used to perform the 'feature extraction'. The intended purpose of VGG19 is to classify images, containing 5 stacks of convolutional layers with each between 2-4 layers *as well as fully connected layers that perform classification*. Hence for the purpose of neural style transfer, we only use the convolutional layers that essentially track patterns and geometrical features that become increasingly more developed and complex.

Thus what we truly define are two loss functions that track content and style loss respectively and perform backpropagation on a *weighted sum* of the combined style and content loss. The two losses are defined similarly, where content loss is the mean squared error between the two 'feature maps' of the generated image and the content image, and style loss is the mean squared error between the gram matrices of the outputs. The gram matrix is a product of these features and its transpose, so it essentially 'extends' the information it captures in individual rows/columns to a less local sense– a clever way of representing "style" as an overarching representation of certain textures and shapes. In terms of the logistics behind the implementation, I used PyTorch after having used it for Assignment 3, which already contains VGG in its models subpackage.

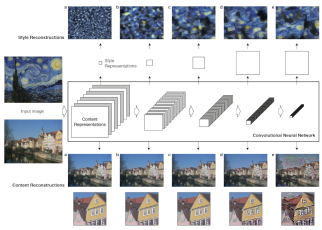

Fig. 1: Convolutional Neural Network as shown in Gatys' paper

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

Fig. 2: Loss function defined as the weighted sum between the content/style losses, which are calculated with MSE.

Defining my own parameters was mainly the number of iterations of the model running and the weights that define the loss function to be backpropagated. As later seen in my results, I asserted that the style loss should contribute more to the overall loss function because the output image's content can still be strongly correlated to the original image with a smaller weight, but injecting the style texture has a more subtle visual effect. And while the style and content loss are received as quantitative outputs, the results are mostly gauged by judging the actual output image; certain textures map onto different features in the content image differently which can't be inferred from the numbers, and even though the losses level off in increasing iterations, the output seems to capture more noise– see results for more.

4 Preliminary Results

As mentioned above, my input images contained similar features and even had comparable perspectives because I was interested in seeing how I could emulate the style in a visually similar real-life environment; however, I initially didn't realize we were stripping the VGG network of its fully connected layers so the network doesn't detect specific objects in the style image like mountains or water. I ran the model on three different Studio Ghibli images with three different photos. As can be seen in the next images, some style transfer is evident, but we cannot say that the output image is something that is distinctly Studio Ghibli:

Fig. 3: Style

Fig. 4: Content

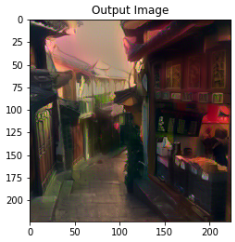

Fig. 5: Output with 300 steps

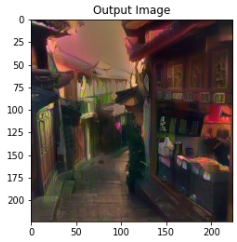

Fig. 6: Output with 1000 steps

As can be seen above, the main part of the 'style' that transferred over was the lighting and other colors; the image quality is sub-optimal and after seeing the 300 iterations result, I decided to try 1000 to see if it would fare better; the image was sharper but also seemed to capture more random noise so it's a point of interest to explore this parameter. I also looked into mixing these two mountainous landscapes together, and came with some outputs that captured the style representation differently; you can see how different areas of the forests were styled as the water/sky from the style image, and the content retained is less evident compared to the prior example. Thus I looked into increasing both the number of iterations as well as the content weight, receiving different outputs for each (see figures 7-11).

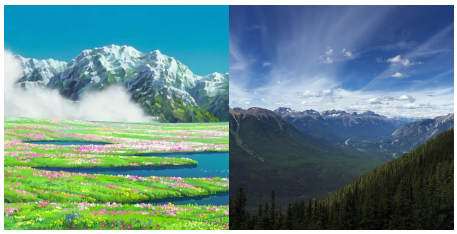

Fig. 7: Style

Fig. 8: Content

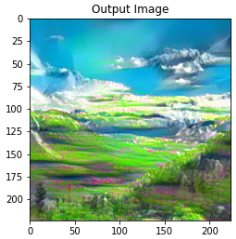

Fig. 9: Output with 300 steps

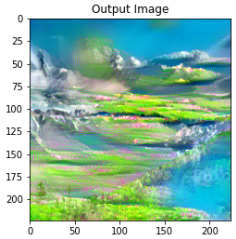

Fig. 10: Output with 1000 steps

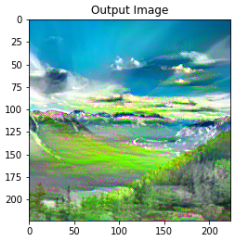

Fig. 11: Output with 10x content weight

While watching my model run, there were these unusual instances where the style loss would spike as shown in the screenshot and provided graph; although it was eventually brought back down to similar levels as the previous iterations, I'm not sure why this sudden spike occurred and I will do some online research. (See Figures 15, 16, 17 on the next page).

The combination below had an interesting and aesthetic output, but the amount of content retained is less than the previous– perhaps in part due to the lack of lighting in the style image– and image quality is lower in general. It would be interesting to explore how the optimal number of iterations varies depending on your input.

Fig. 12: Style

Fig. 13: Content

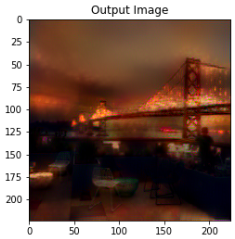

Fig. 14: Output with 300 steps

I believe there is plenty to grow and add onto this project, and it was very entertaining feeding the images through the model; I'm looking forward to improving it for future deliverables!

```
run [500]:
Style Loss: 0.455225, Content Loss: 6.090023
run [525]:
Style Loss: 1463093.250000, Content Loss: 118.467369
run [550]:
Style Loss: 2292882.000000, Content Loss: 147.835785
run [575]:
Style Loss: 1620554.875000, Content Loss: 126.593224
run [600]:
Style Loss: 528.839233, Content Loss: 13.142427
run [625]:
Style Loss: 111.640251, Content Loss: 13.370964
run [650]:
Style Loss: 21.541271, Content Loss: 12.525756
run [675]:
Style Loss: 10.061790, Content Loss: 11.898201
run [700]:
Style Loss: 5.764795, Content Loss: 11.234029
run [725]:
Style Loss: 3.674924, Content Loss: 10.460725
```

Fig. 15: Significant spike in style and content loss corresponding to Fig. 10

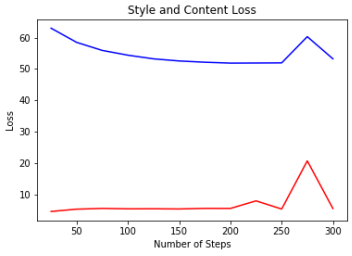

Fig. 16: Loss progression corresponding to Fig. 11

```
run [25]:
Style Loss: 14.562276, Content Loss: 4.355816
run [50]:
Style Loss: 5.923722, Content Loss: 4.232989
run [75]:
Style Loss: 3.860537, Content Loss: 3.706448
run [100]:
Style Loss: 3.269200, Content Loss: 3.364498
run [125]:
Style Loss: 2.766975, Content Loss: 3.222800
run [150]:
Style Loss: 2.208624, Content Loss: 3.157234
run [175]:
Style Loss: 1.692211, Content Loss: 3.131548
run [200]:
Style Loss: 1.299350, Content Loss: 3.089679
run [225]:
Style Loss: 1.078737, Content Loss: 3.024467
run [250]:
Style Loss: 0.957168, Content Loss: 2.953321
run [275]:
Style Loss: 0.892609, Content Loss: 2.893160
run [300]:
Style Loss: 0.850574, Content Loss: 2.851326
run [325]:
Style Loss: 0.817760, Content Loss: 2.819460
run [350]:
Style Loss: 0.788208, Content Loss: 2.796227
run [375]:
Style Loss: 0.763351, Content Loss: 2.777627
run [400]:
Style Loss: 0.739059, Content Loss: 2.764120
run [425]:
Style Loss: 0.721607, Content Loss: 2.753074
run [450]:
Style Loss: 0.705114, Content Loss: 2.743176
run [475]:
Style Loss: 0.691454, Content Loss: 2.735714
run [500]:
Style Loss: 0.681052, Content Loss: 2.729104
run [525]:
Style Loss: 0.672516, Content Loss: 2.723257
run [550]:
Style Loss: 0.665996, Content Loss: 2.717833
run [575]:
Style Loss: 0.659724, Content Loss: 2.713238
run [600]:
Style Loss: 0.653884, Content Loss: 2.709396
run [625]:
Style Loss: 0.648580, Content Loss: 2.706800
run [650]:
Style Loss: 0.645559, Content Loss: 2.702542
run [675]:
Style Loss: 0.642044, Content Loss: 2.699651
run [700]:
Style Loss: 0.638754, Content Loss: 2.697110
run [725]:
Style Loss: 0.636093, Content Loss: 2.694515
run [750]:
```

Fig. 17: Loss progression for Asian street market photo

5 Next Steps

There are several next steps that I would like to take in this process. Besides the most obvious area of improvement in finding ways to improve the quality of the output image (increasing working dimensions by using cloud GPUs, reducing the noise/transferring style more cleanly by tweaking the weights and number of step parameters), I could explore using other models, after doing some research online and seeing users report that VGG-16 presents "smoother" images, whereas VGG-19 generates finer details.

Furthermore, while my results do show evidence of some 'style' being transferred, it is a large part due to colors being transferred and a sense of Studio Ghibli implementation is still missing. It's not as simple of a matter as increasing the number of steps so I want to look more into the sufficient number of iterations to generate a satisfactory image; it may also be in part due to the downsizing of the images, and I will be looking into using other resources to run my model rather than using my computer's CPU. There are also "tiling" techniques that allows one to create larger images, but I have not done much research into that. Finally, I will be trying to mix images that have unrelated features together to see what is produced. Any other feedback on how to proceed is much appreciated!