Project Structure

The hands on project on Neural Style Transfer is divided into following tasks:

Task 1: Introduction

We use a content image and a style image. Then, we write an algorithm to generate a new image, where the algorithm basically tries to retain some features from the content image and apply the style from the style image. So, we get our content image sort of stylized in the style of this style image. This can give some really cool results. This technique was proposed by L.A. Gatys, Alexander S. Ecker and Matthias Bethge. They wrote a paper called A Neural Algorithm of Artistic Style back in 2015.

Task 2: Import VGG19

Let’s start by importing a model which we will use to perform the Neural Style Transfer. The way the algorithm works is by using a model pre-trained on a large image data set. The intermediate layers of this pre-trained model work like feature detectors. We will use the output of these intermediate feature detectors and compare that output for say our content image vs a proposed stylized image. This comparison can give us a content cost. Similarly, we can use the output of some of the intermediate layers and compare it with the output of these layers given a proposed stylized image. This can be our style cost. Then, we add the content cost and style cost together. Now, if we run an optimization algorithm to try and minimize this total cost, and updating the proposed style image along the way, we should get a result which retains some features of the original content image but also imparts the stylistic features of the style image to the proposed stylized image. The original paper uses VGG19 model and fortunately, it’s easily accessible in TensorFlow.

Task 3: Libraries and Helper Functions

We need to import a few helper functions to import and process our images. This includes some image pre-processing functions from Keras as well as popular python packages like NumPy and Matplotlib.

Task 4: Image Processing and Display

The helper functions make it really simple for us to load and process our images. Let’s write a function which does all the pre-processing for us using the helper functions. We want to transform an ordinary image into a format the model can understand and work with in an efficient manner. We will take help of the pre-process input method we imported from the vgg19 sub-module from Keras.

We will also write a function with the intention that we should be able to use it on a generated image, a proposed stylized image, which would be a processed image because we would have processed our input content and style with the function above. Essentially, the results from the intermediate layers are going to be arrays and in order to display those as images, we will have to convert them, de-process them to a human-understandable images.

Task 5: Content and Style Models

In order for us to compute content cost, we need to take a look at the activation at some intermediate layer. In the VGG19, there are 5 blocks of layers with each block being made up of 2 to 4 convolution layers followed by one pooling layer. For content cost, we want to use activation from a layer by which layer the features are already well represented so that when we compare this output with the proposed stylized image, these features match in the two images as we try to minimize the overall cost using some optimization algorithm. More specifically, we will use the block5\_conv2 layer.

For the style cost, we can do something similar. We will use 3 different intermediate layers from different blocks to compute our style cost. This is because we want different kind of stylistic features to impact our cost and not just high level or complex features extracted from the style image. So, we will use three convolution layers from different blocks in VGG19. Some will give us low level, broader understanding of the stylistic features and others will be more complex.

Task 6: Compute Content Cost

Content cost is quite simple to calculate. We need to find out the output of the content model with both the content image and the proposed stylized image.

Task 7: Gram Matrix

In order to compute style cost, we will need to define what’s known as Gram Matrix. We calculate Gram Matrices for the activation of the style and the generated image and calculate the style cost by finding the mean squared difference between these two matrices. Gram Matrices give us a strong feature correlation. And, you could try using other techniques here but the original paper on Neural Style Transfer uses Gram matrices so that’s what we are gonna use as well. But the fundamental idea here is that we are going to use these matrices to match feature distribution as opposed to presence of specific features.

Task 8: Compute Style Cost

We have a bunch of style models each corresponding to a different intermediate layer from the VGG19 model. Our total style cost is going to be weighted sum of the costs for each of the models.

Task 9: Training Loop

In order to generate a stylized image, we now need to follow these steps:

1. Initialize the content image, the style image and also store our initial content image in another variable because we will use this to compute content cost as we update the content image.
2. Instantiate an Optimizer. We are going to use the Adam Optimizer.
3. Run the training loop for a given number of iterations.

The training loop will have the following operations:

1. Compute the total cost for each iteration by calculating the Content Cost and the Style Cost.
2. Calculate the gradients of the cost with respect to the generated image using gradient tape.
3. Update the gradients.
4. Save the lowest cost and the generated image associated with the step with the lowest cost. Sometimes the cost may start to increase after hitting a minima, so we want to ensure that we save the image with the lowest cost during all the iterations in a separate variable that we can use later.

Task 10: Plot the Results

Now that the training loop is complete, we will take a look at the best image! If you run this loop for about 100 times, you will start to get even better results but I’d say for just 20 iterations, the algorithm actually spits out something that kinda looks like a painting of the Exeter Cathedral in the style of The Great Wave painting or at least an attempt at that. And like I said before, if you run this algorithm for longer, you can get more interesting and probably aesthetically better results.