Neural Style Transfer

By: Adarsh Pratap Singh

1. Introduction

Imagine that you could transfer the style of Picasso into your own pieces of art. Your paintings would look *exactly* as if Pablo Picasso was the one creating them. Well, we can achieve that by using Deep Learning and Neural Networks. In this project, we are going to explore the problem of Neural Style Transfer.

Neural Style Transfer is a technique that allows for blending style from one image into another image keeping its content intact. This gives an artistic touch to your image. You can use it to make your picture look like an art made by Picasso, Van Gogh, or any other artist of your choice.

The objective of this project is to make a model of Neural Style Transfer using pre-trained feature detection model such as VGG. We want to develop an interesting algorithmic way of creating complex art forms by playing between the content and style of an image.

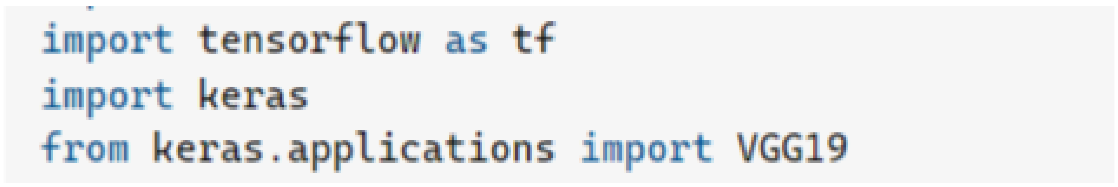
2. Approach

Neural Style Transfer technique requires deep feature detection which can only be achieved by highly optimized and well-trained models.

In this project, we are going to use a pre-trained feature detection model called the VGG19 model. VGG-19 is a convolutional neural network that is 19 layers deep. The pre-trained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals.

2.1 Model Analysis

For feature detection, I am using VGG19 which comes pre-trained with the TensorFlow library and can be easily imported and manipulated.



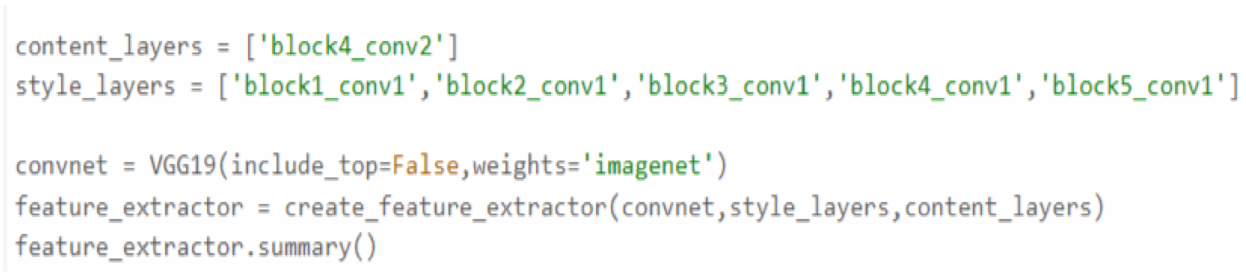
```python
import tensorflow as tf
import keras
from keras.applications import VGG19
```

We just need specific layers from the VGG19:

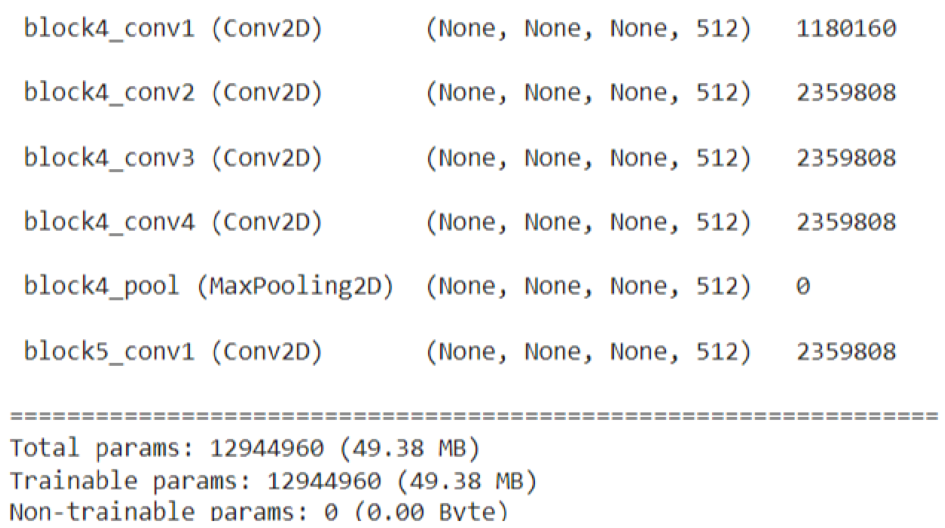
- Initial layers for Style (for basic info)
- Mid layers for Content (for more complex info)

Hence, for style and content, we will use the following layers of VGG19:

- Style: [block1_conv1, block2_conv1, block3_conv1, block4_conv1, block5_conv1]
- Content: [block4_conv2]



```python
content_layers = ['block4_conv2']
style_layers = ['block1_conv1','block2_conv1','block3_conv1','block4_conv1','block5_conv1']

convnet = VGG19(include_top=False,weights='imagenet')
feature_extractor = create_feature_extractor(convnet,style_layers,content_layers)
feature_extractor.summary()
```



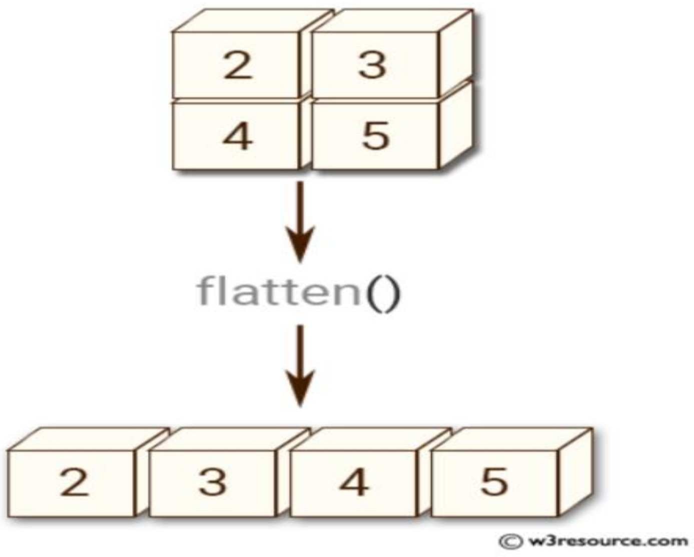
```
 block4_conv1 (Conv2D)       (None, None, None, 512)   1180160

 block4_conv2 (Conv2D)       (None, None, None, 512)   2359808

 block4_conv3 (Conv2D)       (None, None, None, 512)   2359808

 block4_conv4 (Conv2D)       (None, None, None, 512)   2359808

 block4_pool (MaxPooling2D)  (None, None, None, 512)   0

 block5_conv1 (Conv2D)       (None, None, None, 512)   2359808

=================================================================
Total params: 12944960 (49.38 MB)
Trainable params: 12944960 (49.38 MB)
Non-trainable params: 0 (0.00 Byte)
```

As shown in above mentioned summary the model has 5 Blocks each of which collects features at its level, from shallow detection to deep as we move from block 1 to block 5. We are going to collect feature vectors from these layers and then use those features to train our own delicate balance so that all the output images must remain the basic structural composition of the content image but also contain appealing features of the style image.

2.2 Gram Matrix

We need a way to compare the relation of shapes, edges, and textures between style images and generated images. We need to extract basic info from the style image and compare it with the basics of the generated image in a specific way, that specific way is 'Gram Matrix'.

We have output (feature maps) from style layers that we defined. we flatten each feature map from a layer into a vector stack it and make a matrix for each layer. Now we multiply this matrix with its transpose and the resultant matrix is called 'Gram Matrix'.



So we flatten each output from the filters of a layer, stack them in a matrix, and do matrix multiplication of the matrix with its transpose so that all the combinations of dot-product between two vectors from the matrix take place.

After calculating the relations between each feature map, we compare the relations in the generated image and style image and bring them closer to each other by training the generated image.

2.3 Loss Function

Content Loss:
Content Loss is calculated by the difference between feature maps of content layer filters. Feature maps of the content layer for generated image and content image are compared directly because we want spatial information from the content image.

Style Loss:
Style Loss is calculated by the difference between the gram matrix of all feature maps of style layers. The Gram matrix of all feature maps of style layers for the generated image and content image are compared directly so that we can get the same texture in the generated image.

Total Variation Loss:

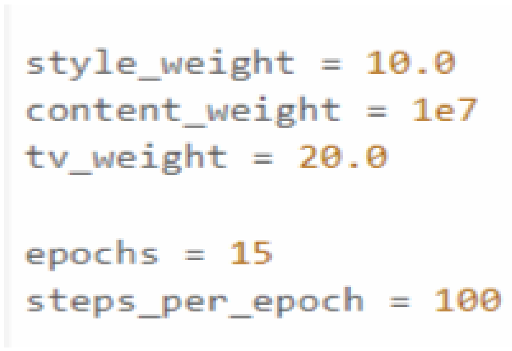
Total Variation Loss encourages spatial smoothness in the generated image to reduce noise.

2.4 Defining content and style layers along with epochs and steps per epoch:

We also define style weight and content weight: the amount by which we want style to be added and content to be added to the generated image.

There is also one more weight here, tv weight (total variation weight). This weight is applied to total variation loss. This loss measures the amount of noise in an image, and reducing this loss reduces the noise and gives a smooth image.

We define epochs and steps per epoch (the number of times we will change the pixel values of the generated image per epoch)



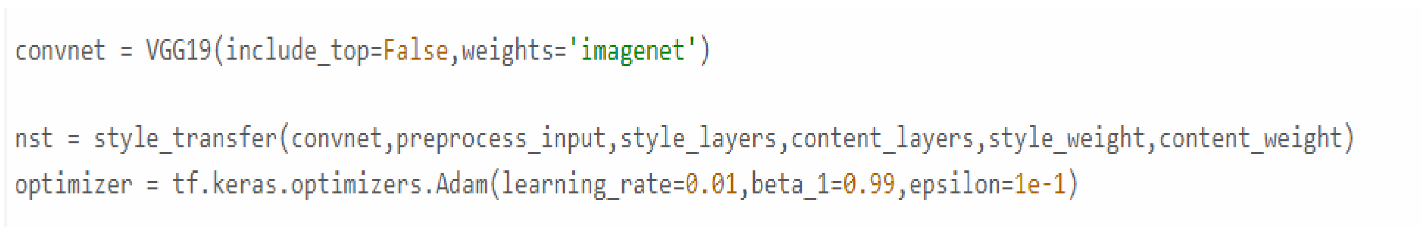
```
style_weight = 10.0
content_weight = 1e7
tv_weight = 20.0

epochs = 15
steps_per_epoch = 100
```

We can change any of the above parameters according to our need.

2.5 Loading the Optimizer:

Use the Adam optimizer to iteratively update the generated image

by minimizing the combined loss.



```
convnet = VGG19(include_top=False,weights='imagenet')

nst = style_transfer(convnet,preprocess_input,style_layers,content_layers,style_weight,content_weight)
optimizer = tf.keras.optimizers.Adam(learning_rate=0.01,beta_1=0.99,epsilon=1e-1)
```

2.6 Training the Images

We define a function. In this function, we calculated the feature maps of the content layer and the Gram-Matrix of feature maps of style layers for the generated image, calculated the style-loss and content-loss, calculated gradients according to the loss, and applied it to the generated image.

And we call this function for epochs x steps_per_epoch times and we have the final image of NST!

We saved the final image into our drive.

3. Failed Approaches

- **Tensor and Image Failures**

Approach: Did not convert the image into tensor

Failure Reason: Feature Extraction and the making of gram matrix demand a tensor.

Solution: Converted the image into tensor using

```
tf.convert_to_tensor(image_rgb)
```

- **Optimization Problem**

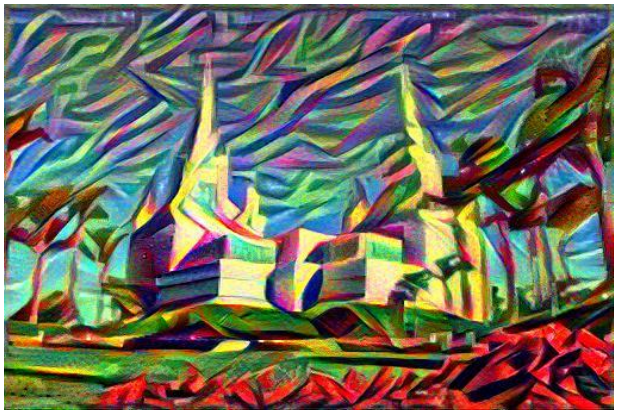
Approach: An optimization problem occurs every time when uploading a new image for style transfer.

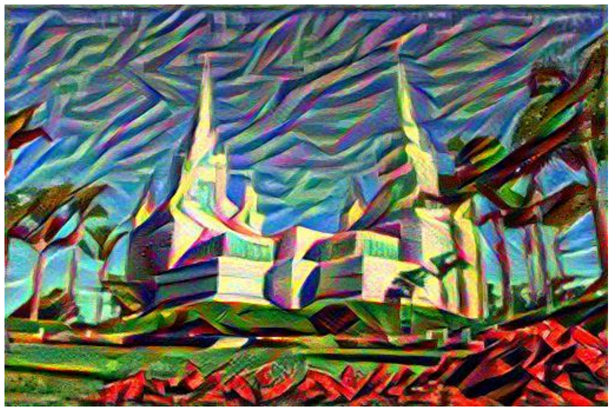
Failure Reason: Preprocessing was not done for the new Image.

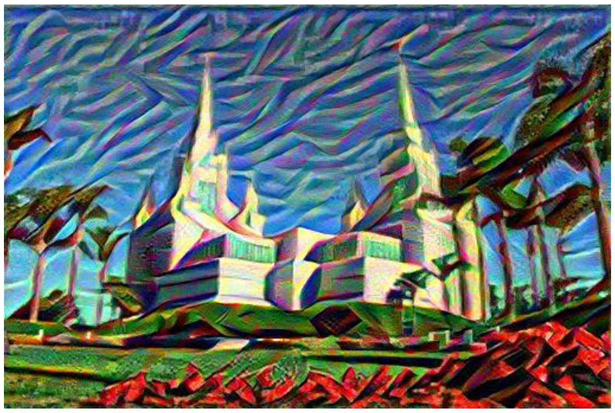
Solution: Run all the code cells to preprocess the new image.

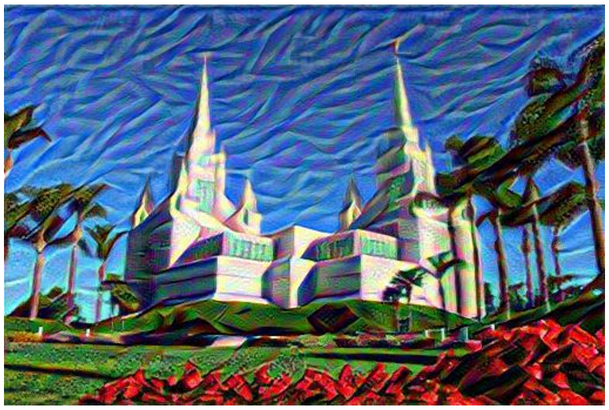
4. Results

4.1 Same Image with Different no. of Iterations (50 Iterations = 1Epoch)


50 Iterations

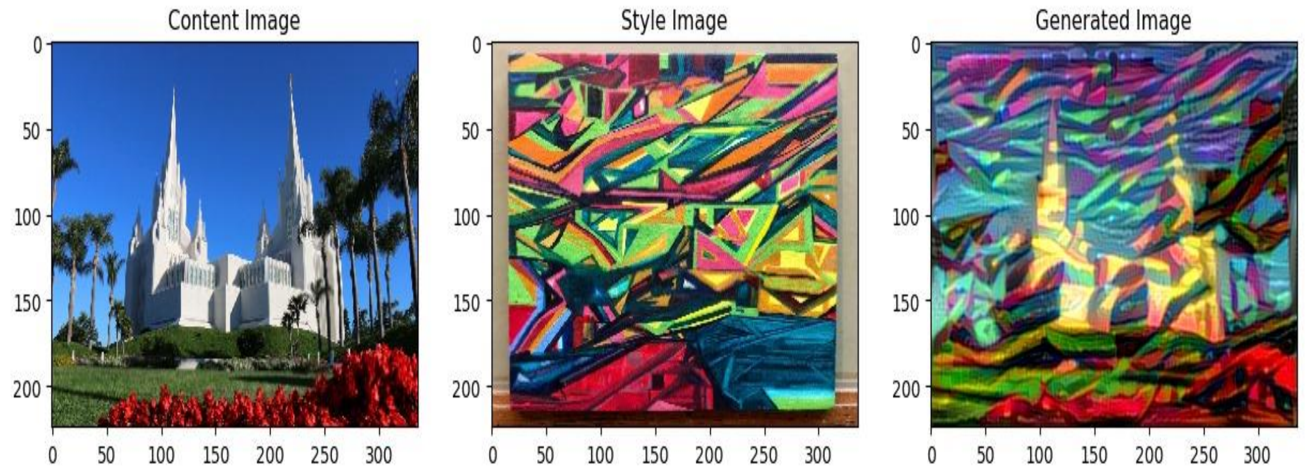

100 Iterations


150 Iterations


200 Iterations

- **Visual Results:**

The project successfully generated images that combine the content of the target image with the style of the style image. Below are the results for various datasets:



5. Discussions

- **Analysis of Results:**

Content loss: Content loss of the images is less. The images retained their structure and features.

Style Loss: The generated images beautifully captured the essence of the style images.

Loss Convergence: The total loss decreased as the number of epochs and total iterations per epochs increased.

- **Insights:**

Adjusting Weights: Adjusting weights of content image and style image is very crucial. You have to play with different no. to get the image you want.

Number of Epochs: The number of epochs and steps per epoch should be considered according to the machine. If the device has gpu available then only go for a higher no. of epochs otherwise it is going to take too much time.

Loss Convergence: The total loss decreased as the number of epochs and total iterations per epochs increased.

Preprocessing of Images: Preprocessing of images is very important. Without the preprocessing optimizer is going to throw an error. Preprocess the images using VGG19's preprocessing function to ensure they are in the correct format for the model. Resize the images to a fixed height (224 pixels) while maintaining the aspect ratio.

6. Conclusions

- **Summary Findings:**

1. Implemented Neural Style Transfer using the VGG19 model and TensorFlow.

2. You can generate an image according to your choice by taking the right weight of content and style images.

3. Less training time can be achieved by the use of GPU and by lowering the number of iterations.

4. Preprocessing of the image is a very important step while training the model.

- **Possible Future Improvements:**

1. with the help of more powerful GPUs that have more processing power, iterations can be increased and training could become much faster resulting in a better image.

2. Instead of Adam optimizer more advanced optimizers can be used such as AdamW or LAMB.

3. More pre-trained models will be available that will improve the results.

7. References

1. TensorFlow Documentation:
https://www.tensorflow.org/tutorials/generative/style_trans

2. Machine Learning A-Z course on Udemy:
https://www.udemy.com/course/machinelearning/