20XD58 – CAPSTONE PROJECT

REPORT

NEURAL STYLE TRANSFER

20PD05 ANU RAMYA R

20PD24 SAMYUKTHA V

ABSTRACT

Deep neural networks have already surpassed human level performance in tasks such as object recognition and detection. However, deep networks were lagging far behind in tasks like generating artistic artefacts having high perceptual quality until recent times. Creating better quality art using machine learning techniques is imperative for reaching human-like capabilities, as well as opens up a new spectrum of possibilities. And with the advancement of computer hardware as well as the proliferation of deep learning, deep learning is right now being used to create art.

The seminal work of Gatys et al. demonstrated the power of Convolutional Neural Networks (CNNs) in creating artistic imagery by separating and recombining image content and style. This process of using CNNs to render a content image in different styles is referred to as Neural Style Transfer (NST).

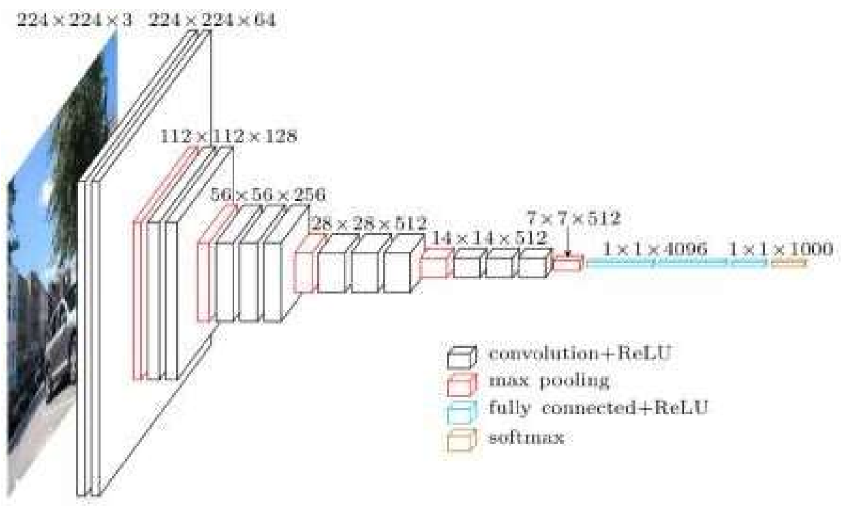
NST builds on the key idea that,

It is possible to separate the style representation and content representations in a CNN, learnt during a computer vision task (for example, an image recognition task).

The aim of this project is to implement Neural Style Transfer by using the technique demonstrated by Gatys et al in his paper “A Neural Algorithm of Artistic Style”. A modification of the algorithm is also done to perform Style Transfer with multiple styles.

ARCHITECTURE OF THE MODEL:

Transfer learning is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned. We have many models which are pre-trained on large image dataset. These models have learnt to extract the features from the image. These learning have been stored in the form of the weights of the model. We will simply use them to extract the style and content of our images. Here, we will be using VGG19 model.



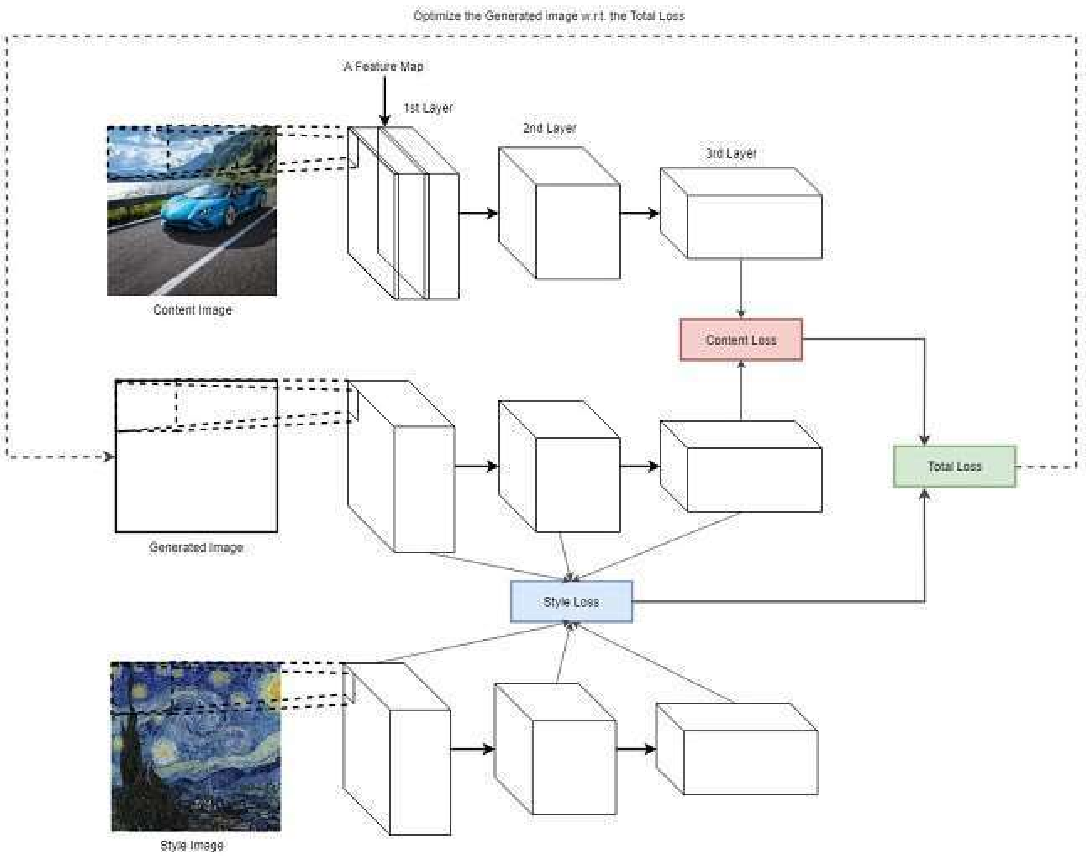
Architecture of VGG-19

Then to define a loss function which blends two images seamlessly to create visually appealing art, NST defines the following inputs:

Content image - the image we want to transfer a style to

Style image - the image we want to transfer the style from

Input (generated) image - the image that contains the final result



Intermediate layers for style and content

CNNs are able to generalize well in image classification because they are able to capture the invariances and defining features within classes (e.g., cats vs. dogs) that are agnostic to background noise and other nuisances. Thus, somewhere between where the raw image is fed into the model and the output classification label, the model serves as a complex feature extractor. By accessing intermediate layers of the model, you're able to describe the content and style of input images.

Intermediate layer to extract content:

Complex features are extracted in the final layers. Hence, the feature maps from higher layers provide a better representation of the content of the image. Here, to extract content from the image, we use higher layer block5_conv2 from the model.

Intermediate layers to extract style:

At each layer, CNN keeps learning a few features. In the starting layers, the network learns simple features such as detecting horizontal, vertical or diagonal lines in the first layer and detecting corners in the second layer and so on. As at each layer some of the other patterns are being detected therefore we will use multiple layers(one from each block) to extract the style information.

Here, we use the 'block1_conv1', 'block3_conv1' and 'block5_conv1' layers to extract style information.

Defining loss function

The total loss function can be divided into two parts:

- Content Loss
- Style Loss

$$L = \alpha L_{content} + \beta L_{style}$$

where α and β denote the weights assigned to content loss and style loss respectively. In this project, we use $\alpha = 10000$ and $\beta = 0.1$

Content Loss: It is used to check how much the generated image is similar to our content image. Let $A^{l}_{ij}(I)$ be the activation of the l th layer, i th feature map and j th position obtained using the image I. Then the content loss is defined as,

$$L_{content} = \frac{1}{2} \sum_{i,j} (A^{l}_{ij}(g) - A^{l}_{ij}(c))^2$$

Style Loss: Style Loss is used to check how much the style of the generated image differs from the style of the style image. But the difference is that the style of any image is not simply represented as in the case of content. We use gram matrices to represent style of an image. The correlation of all the channels of the feature maps from different layers w.r.t each other is given by the Gram Matrix of an image. We will use the Gram Matrix to measure the degree of correlation between channels which later will act as a measure of the style itself.

$$L_{style} = \sum_{l} \sum_{i,j} (G_{i,j}^{s,l} - G_{i,j}^{p,l})^2$$

$G_{i,j}^{s,l}$ is the Gram Matrix of the style image

$G_{i,j}^{p,l}$ is the Gram Matrix of the generated image

Platform used:

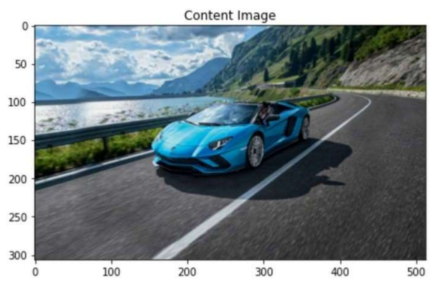
This project was developed on Google Colab platform.

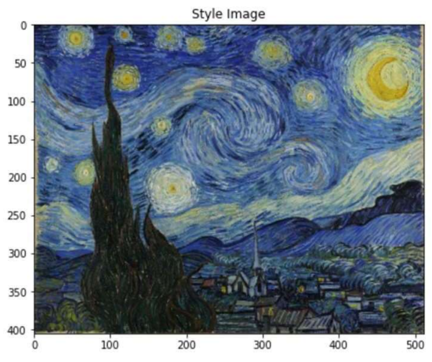
Language used: Python

Module: TensorFlow

STEPS INVOLVED IN NST

- **Load content image and style image**

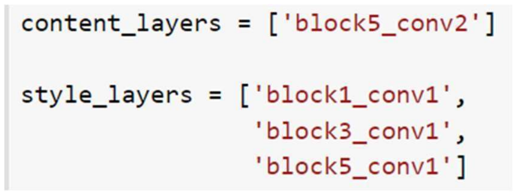




- **Pre-process the images**

This project is implemented in Python using TensorFlow library. The images are converted to tensor datatype which facilitates use of specialized image-processing functions on them efficiently. The image is resized so that maximum dimension is 512 pixels, and the pixel values of image are transformed to the range 0 to 1.

- **Define the content and style layers**



```
content_layers = ['block5_conv2']

style_layers = ['block1_conv1',
                'block3_conv1',
                'block5_conv1']
```

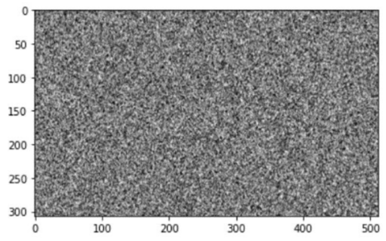
- **Define the extractor**

The extractor builds a model to return the outputs from specified layers of the VGG-19 model.

- **Define style and content outputs**
- **Define style and content targets**

Style and content targets denote the style and content outputs of the style and content image respectively.

- **Initialize generated image**



- **Define loss function**

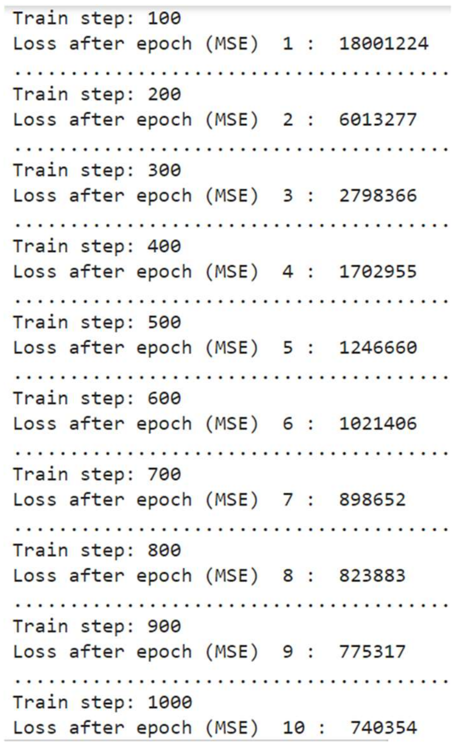
Total loss is a weighted sum of content and style loss. The two losses are computed with the content and style outputs of the input image w.r.t the content and style targets respectively.

- **Define optimizer**

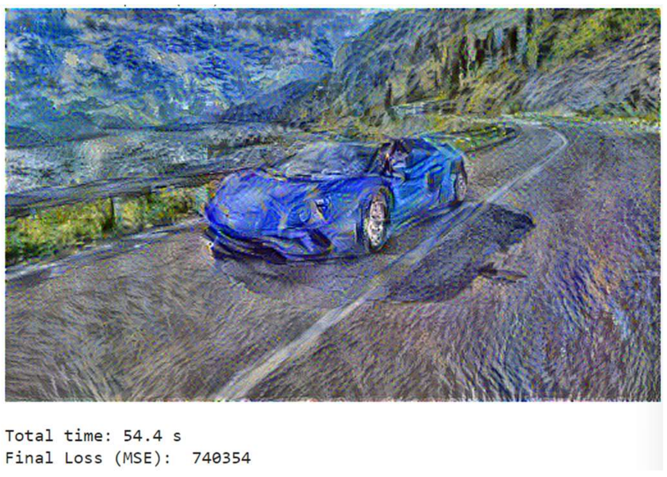
The Adam optimizer is used to implement gradient descent in order to minimize the total loss. The output image(generated) is adjusted at each step based on the gradient. The image is also re-normalised after each update.

- **Run the optimizer**

We run the optimizer for 10 epochs with 100 training steps for each epoch. We compute the time taken to execute this training process and also the final loss at the end of the training phase.



```
Train step: 100
Loss after epoch (MSE)  1 :  18001224
.......................................
Train step: 200
Loss after epoch (MSE)  2 :  6013277
.......................................
Train step: 300
Loss after epoch (MSE)  3 :  2798366
.......................................
Train step: 400
Loss after epoch (MSE)  4 :  1702955
.......................................
Train step: 500
Loss after epoch (MSE)  5 :  1246660
.......................................
Train step: 600
Loss after epoch (MSE)  6 :  1021406
.......................................
Train step: 700
Loss after epoch (MSE)  7 :  898652
.......................................
Train step: 800
Loss after epoch (MSE)  8 :  823883
.......................................
Train step: 900
Loss after epoch (MSE)  9 :  775317
.......................................
Train step: 1000
Loss after epoch (MSE)  10 :  740354
```

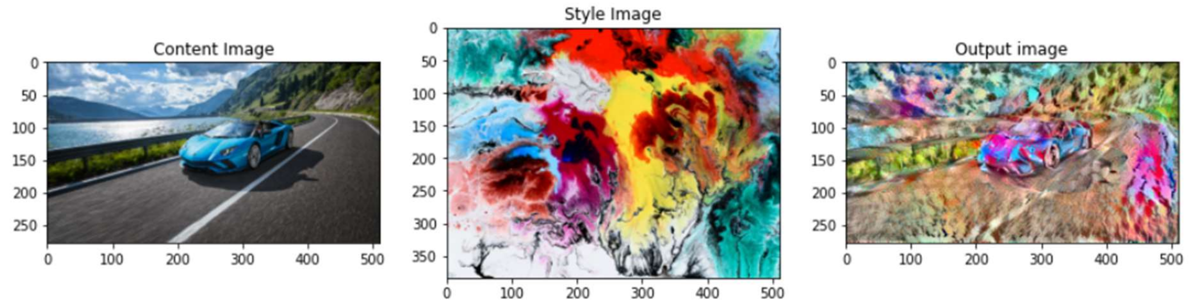


```
Total time: 54.4 s
Final Loss (MSE):  740354
```

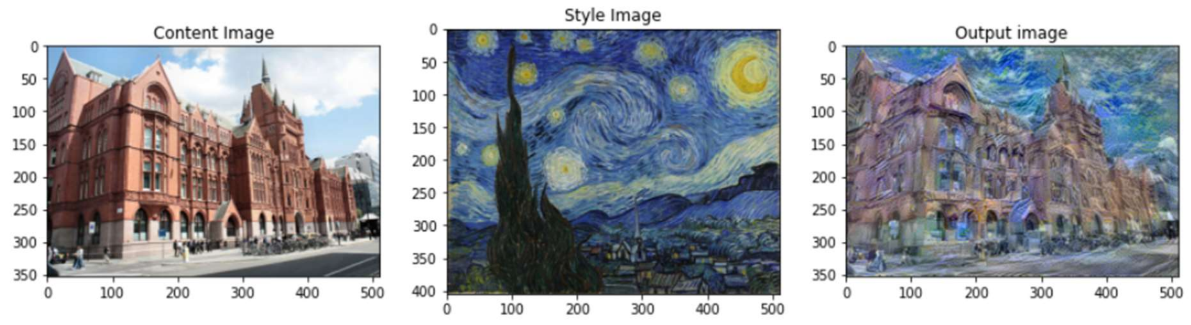
Testing the algorithm on a set of images to compute average measures:

We run the algorithm for 10 sample instances (pair of content and style image) and compute total time and final loss at the end of each run:

```
Image 1 :
```

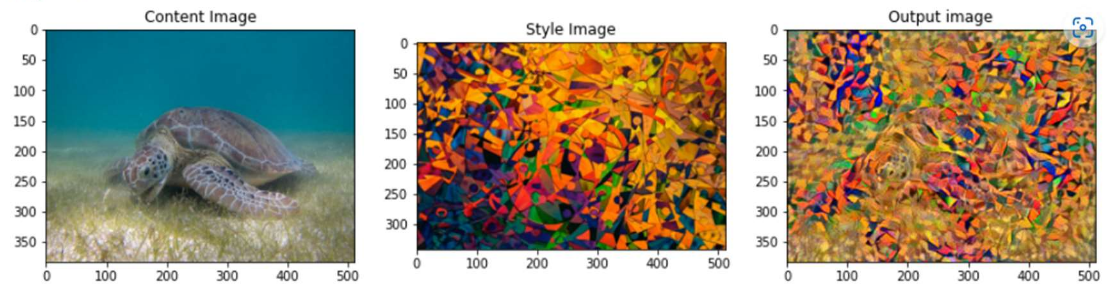


```
Total time: 46.3 s
Final Loss (MSE):  2695649
Image 2 :
```



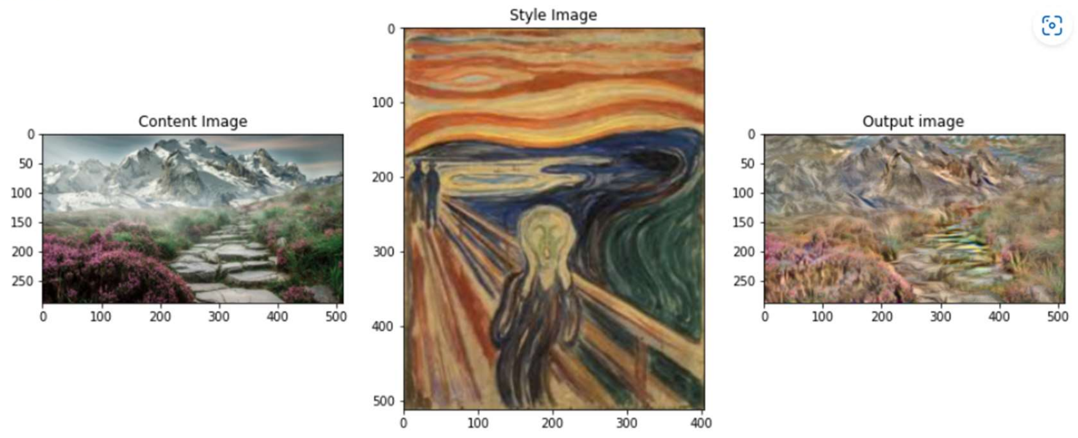
```
Total time: 56.5 s
Final Loss (MSE):  3398983
```

```
Image 3 :
```



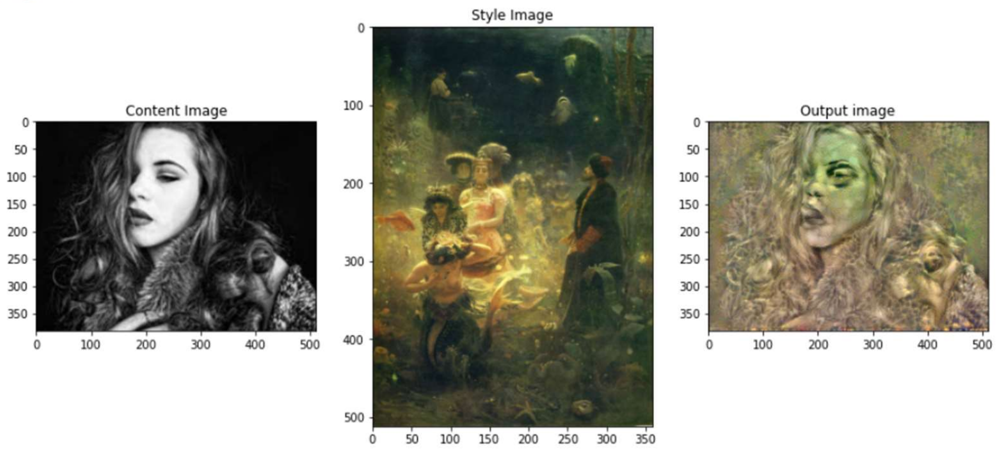
```
Total time: 60.8 s
Final Loss (MSE):  4126239
```

```
Image 4 :
```



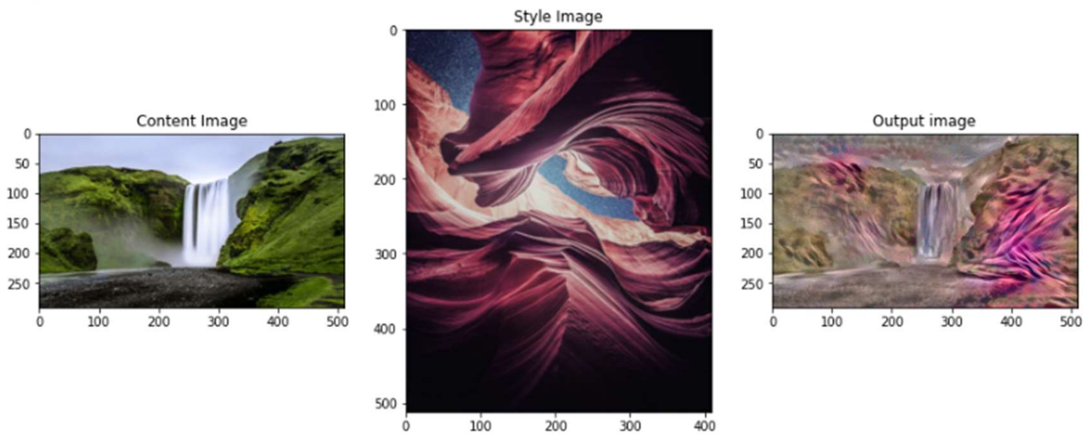
```
Total time: 48.7 s
Final Loss (MSE):  2088794
```

```
Image 5 :
```



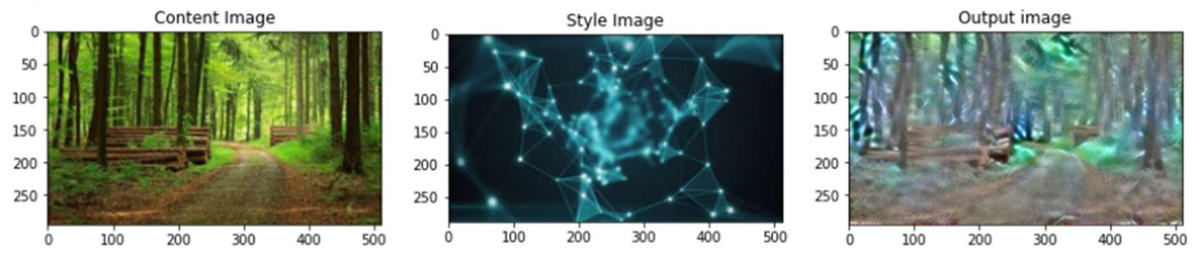
```
Total time: 61.8 s
Final Loss (MSE):  1102338
```

```
Image 6 :
```

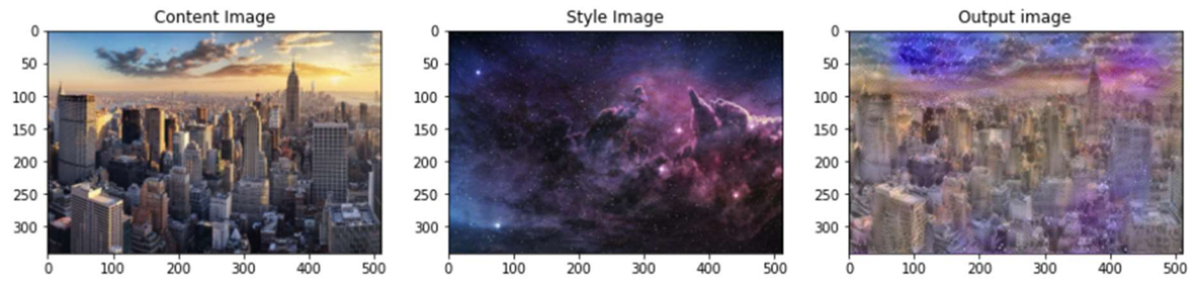


```
Total time: 49.6 s
Final Loss (MSE):  2194769
```

```
Image 7 :
```

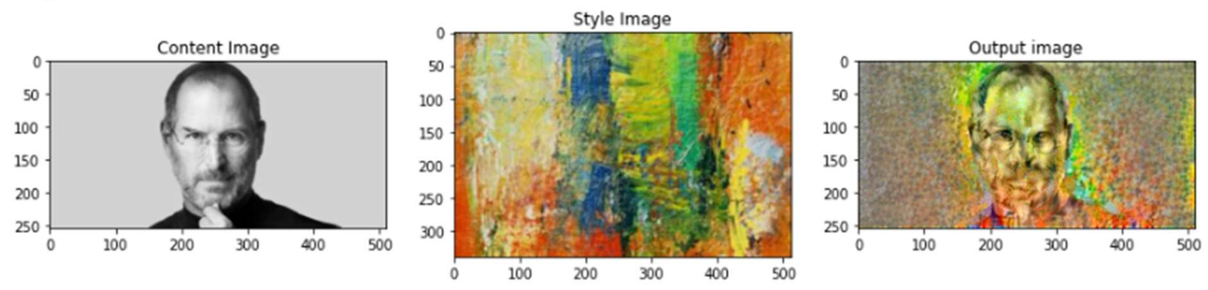


```
Total time: 49.5 s
Final Loss (MSE):  2988092
Image 8 :
```

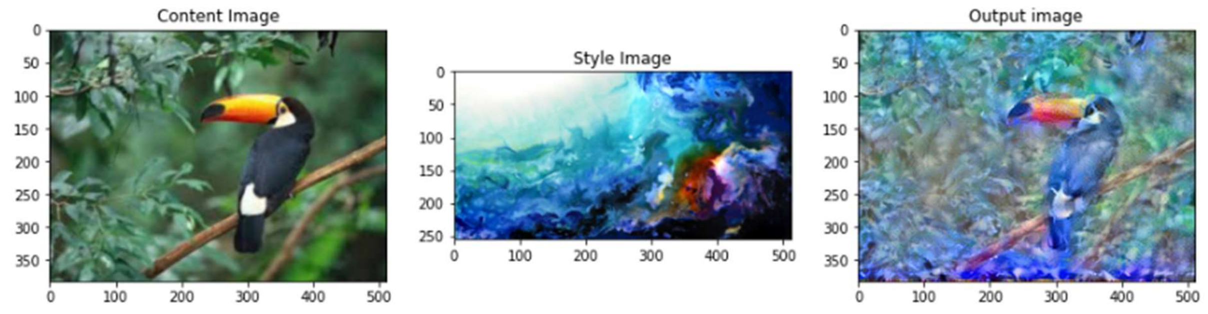


```
Total time: 57.1 s
Final Loss (MSE):  2251605
```

```
Image 9 :
```



```
Total time: 45.1 s
Final Loss (MSE):  1064436
Image 10 :
```



```
Total time: 60.8 s
Final Loss (MSE):  1528151
```

Average time taken to transfer: 53.64 s

Average loss (MSE) after 5 epochs: 2343905

Modified implementation : NST with two style images

In the technique discussed earlier, we saw how the generated image extracts content from the content image and style information from the style image.

In this modified technique, we create the output image which extracts content information from a content image, but extracts style information from two different style images.

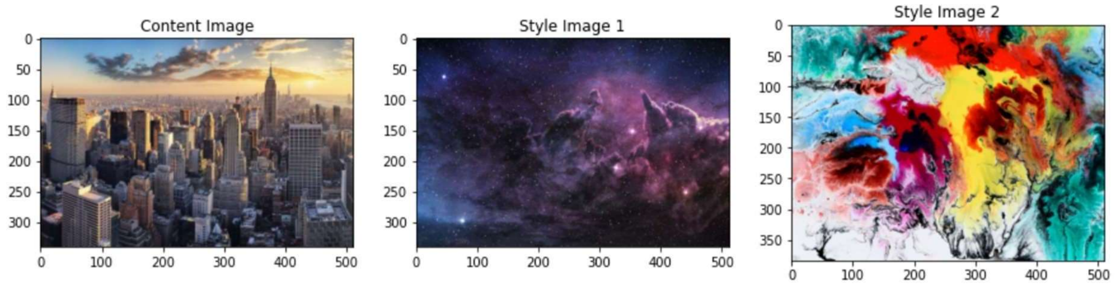
The architecture of the model remains the same. We use the same intermediate layers to extract content and style. We only make changes to the loss function:

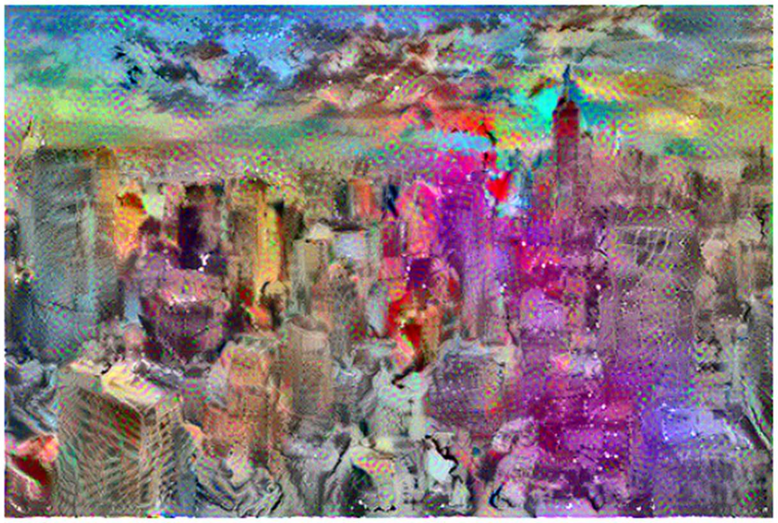
Here, the total loss is given by the weighted sum of content loss, style loss from style image 1 and style loss from style image 2

$$\boxed{L = \alpha\, L_{content} + \beta\, L_{style1} + \gamma\, L_{style2}}$$

Here, $\alpha = 10000$, $\beta = 0.1$, $\gamma = 0.1$

Sample run:





```
Train step: 1000
Total time: 110.7 s
Final loss:  38931220
```