Neural Style Transfer

Project Report

Manish Aradwad (Sr. No. - 19494)
Pratyush Gauri (Sr. No. - 20227)

Abstract

This project explores methods for artistic style transfer based on convolutional neural networks. The core idea proposed by Gatys et. al became very popular and with further research, Johnson et. al overcame a major limitation to achieve style transfer in real-time. We use the VGG16 model for manual implementation of Neural Style Transfer. Another implementation is based on an article by TensorFlow which uses a pre-trained model for NST.

Additionally, we have extended the idea of style transfer up to 3 style images. We try to represent the content image with all the 3 styles combined.

1. Style Transfer Algorithm

In deep learning, we normally optimize the parameters of some neural network. However, to transfer the style of an artwork $\hat{a}$ onto a photograph $\hat{p}$, we repeatedly optimize the pixel values of the constructed image $\hat{x}$, so that $\hat{x}$ simultaneously matches the style representation of $\hat{a}$ and the content representation of $\hat{p}$. At the beginning, $\hat{x}$ is initialized to random noise. In order to get a good stylized output, $\hat{x}$, we jointly minimize the content loss and the style loss. The final loss function we minimize is:

$$L_{total}(\hat{p}, \hat{a}, \hat{x}) = \alpha L_{content} + \beta L_{style}$$

where α and β are real numbers (they are hyperparameters to be set). A large $\frac{\alpha}{\beta}$ ratio means that we want to emphasize content of the photograph in the constructed image $\hat{x}$, while a small $\frac{\alpha}{\beta}$ means that we want to emphasize the style of the artwork in the constructed image $\hat{x}$.

2. Content

A given image is encoded in each layer of a CNN by that layer's filter responses to that image. A layer with N_l distinct filters has N_l feature maps each of size M_l, where M_l is the height times the width of the feature map. Therefore, the response in a layer l can be stored in a matrix $F_l \in R^{Nl \times Ml}$, where is the activation of i^{th} filter at position j at layer l.

The F_ls can be used to represent the content of the image. The filter response of lower layers of VGGnet(meaning layers closer to the input) look very close to the input image, while the filter representation of the the higher layers of VGG net captures the high level content information(e.g. objects in the scene, the relative positions of objects) and largely ignores the pixel-by-pixel information of the input image.

Let $\hat{o}$ and $\hat{g}$ be the original image and the image that is generated respectively, and O^l and G^l be their respective filters at layer l. The contribution of each layler l to the total content loss is:

$$C(\hat{o}, \hat{g}) = \frac{1}{2} \sum_{i,j} (O^l_{i,j} - G^l_{i,j})$$

The content loss between $\hat{o}$ and $\hat{g}$ is given as:-

$$L_{content}(\hat{o}, \hat{g}) = \sum_{l=0}^{L} c_l \, C_l(\hat{o}, \hat{g})$$

where c_l is a hyperparameter that specifies the weighting factor of the contribution of each layer(note that some c_ls could be 0, indicating that we don't use the filter response of that layer).

3. Style

The style of an image is captured by the correlations between the different filter responses. The feature correlations are given by the Gram Matrix $G_l \in R^{N_l X N_l}$, where G^l_{ij} is the inner product between the vectorized feature maps i and j at layer l:

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}$$

By including feature correlations of multiple layers, we capture the style information of the image while ignoring the content information(e.g. objects present in scenes, global arrangements of objects).

Let $\hat{o}$ and $\hat{g}$ be the original image and the image that is generated respectively, let O^l and G^l be their style representations in layer l respectively, the contribution of layer l to the total style loss is($N^l \times M^l$ is the size of the filter at layer l):

$$S_l(\hat{o}, \hat{g}) = \frac{1}{4 N_l^2 M_l^2} \sum_{i,j} (O_{ij}^l - G_{ij}^l)2$$

and the total style loss between $\hat{o}$ and $\hat{g}$ is:

$$L_{style}(\hat{o}, \hat{g}) = \sum_{l=0}^{L} s_l S_l(\hat{o}, \hat{g})$$

4. Extension

As an extension of the Neural Style Transfer algorithm, we have tried to stylise the content image using 3 separate style images. During implementation, we took the weighted sum of all style images. The images are resized to be 512x512 pixels.

The weights assigned sum up to 1. With the weights, it is possible to decide which style image's style should be more prominent in the final image.

Following are the results of various content and style images:

NST 1 Style Implementation 1

Content Image

Style Image

Library function output

Generated Image

Manual Implementation output

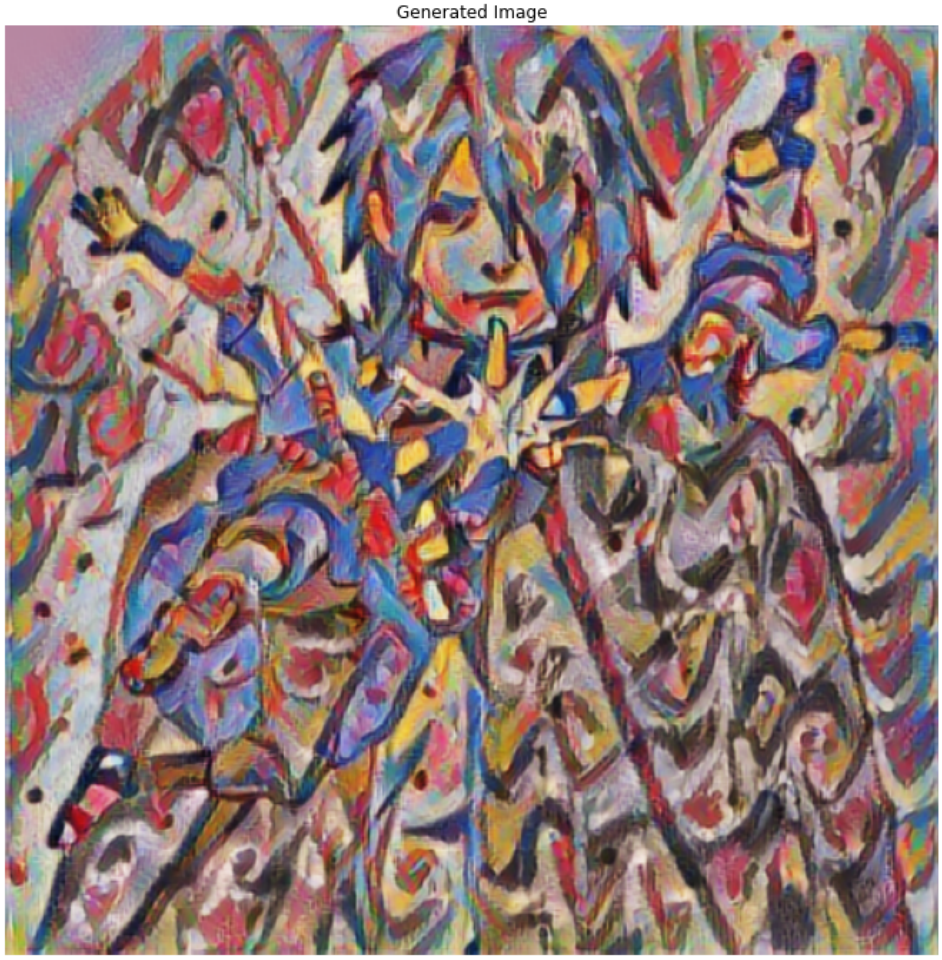
NST 3 Style Implementation 2

Content Image

Style Image

Library function output

Generated Image

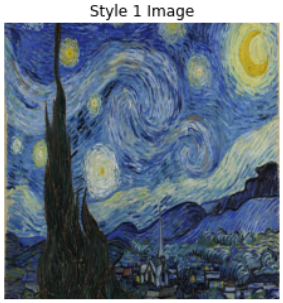


Manual Implementation output

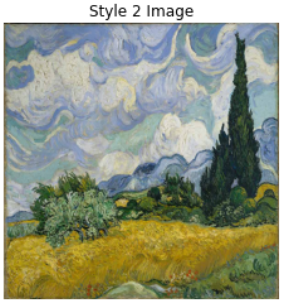
NST 3 Style Implementation 3

Content Image

Style 1 Image



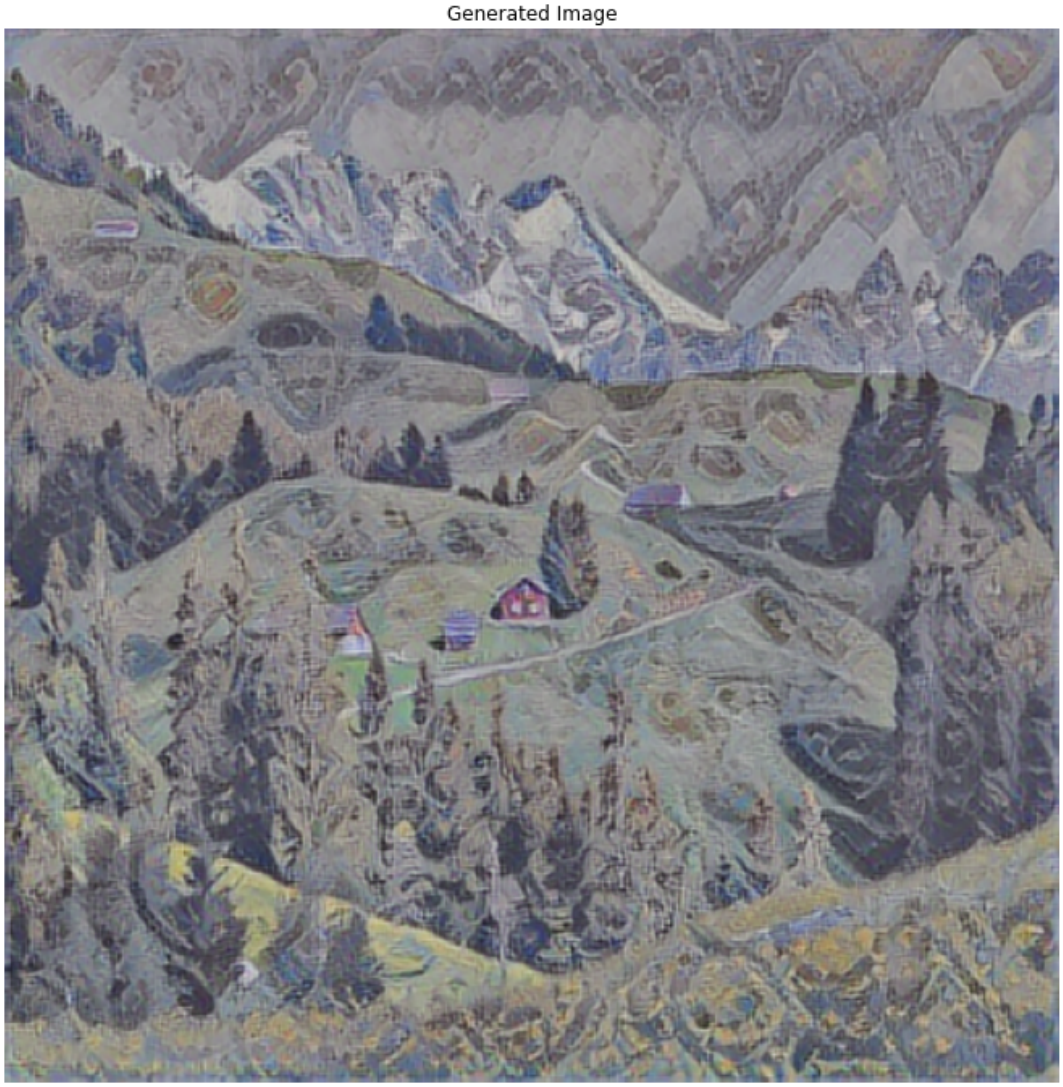
Style 2 Image



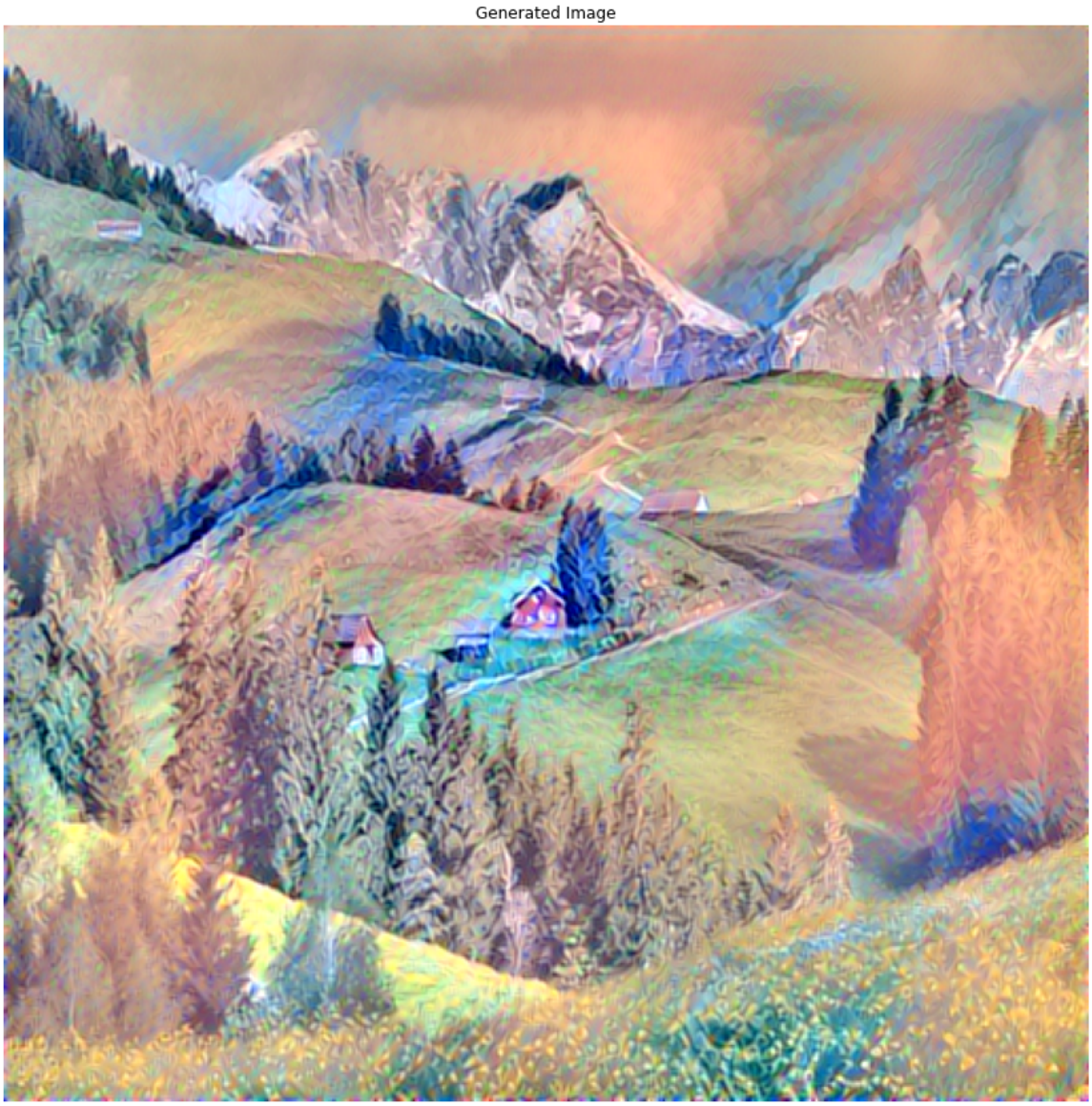
Style 3 Image

Library function output

Generated Image

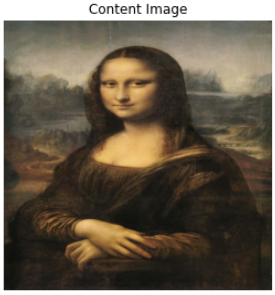


Manual Implementation output

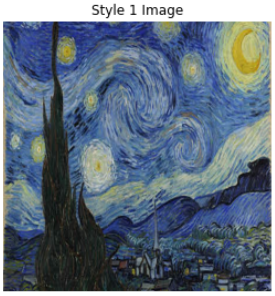


NST 3 Style Implementation 2

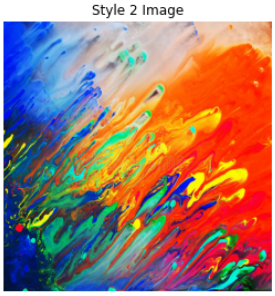
Content Image



Style 1 Image



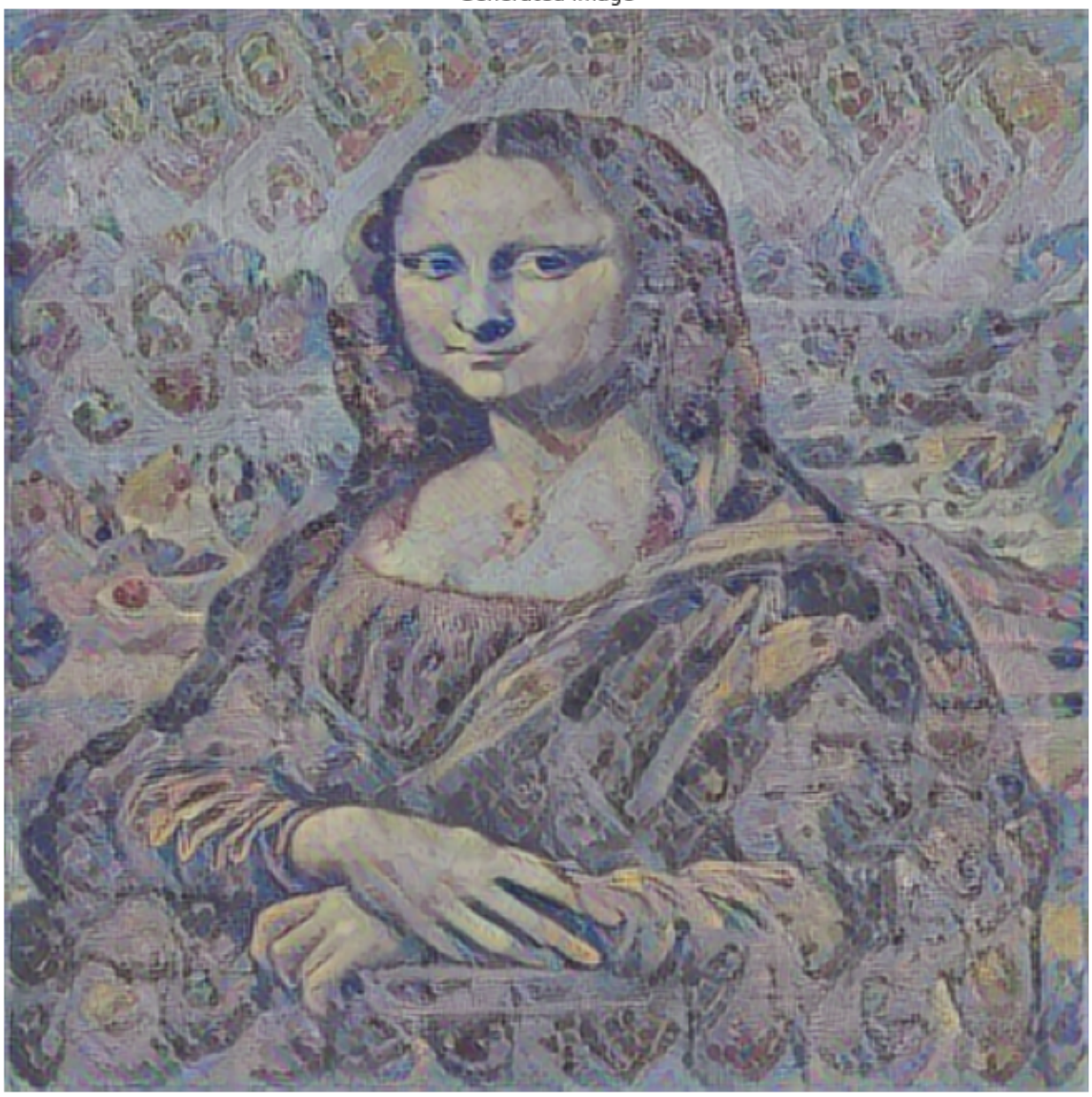
Style 2 Image



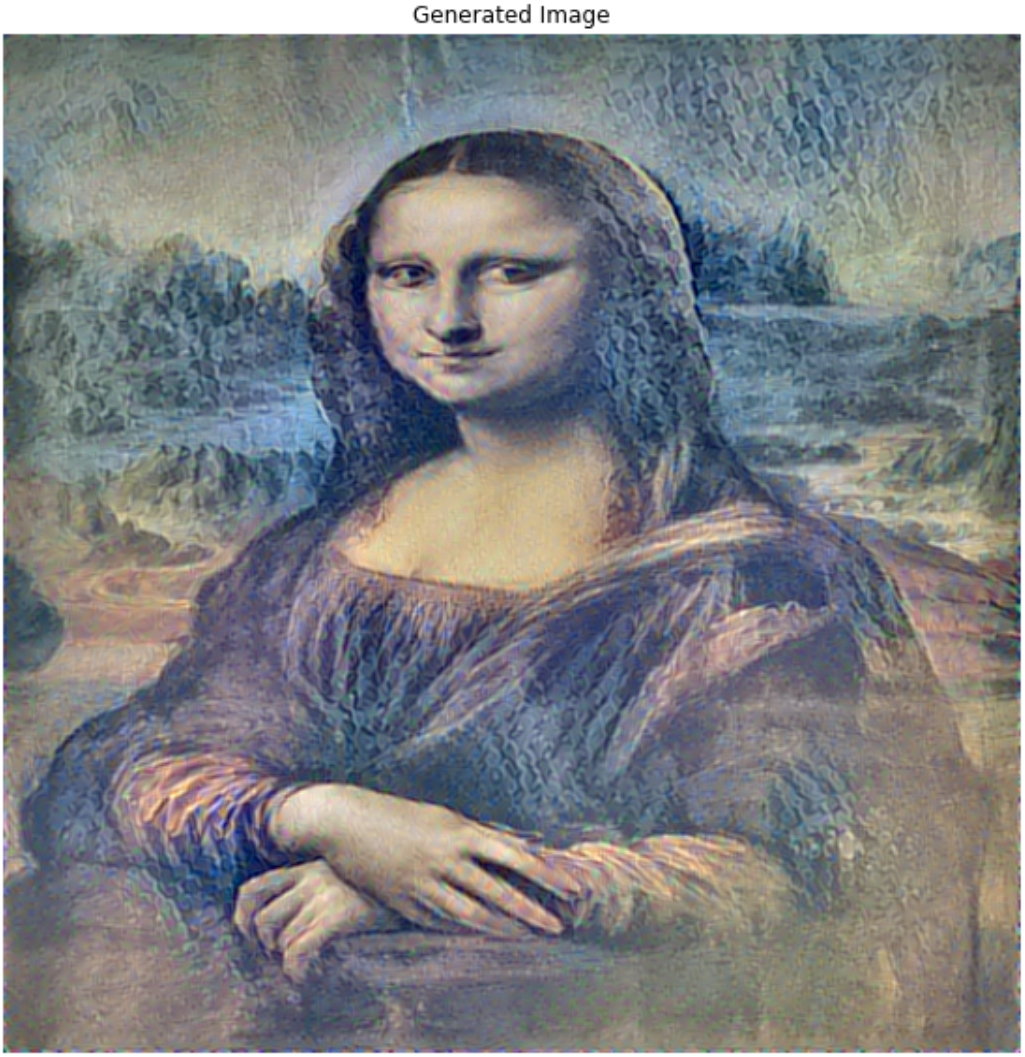
Style 3 Image

Library function output

Generated Image



Manual Implementation output



NST 3 Style Implementation 3

Content Image

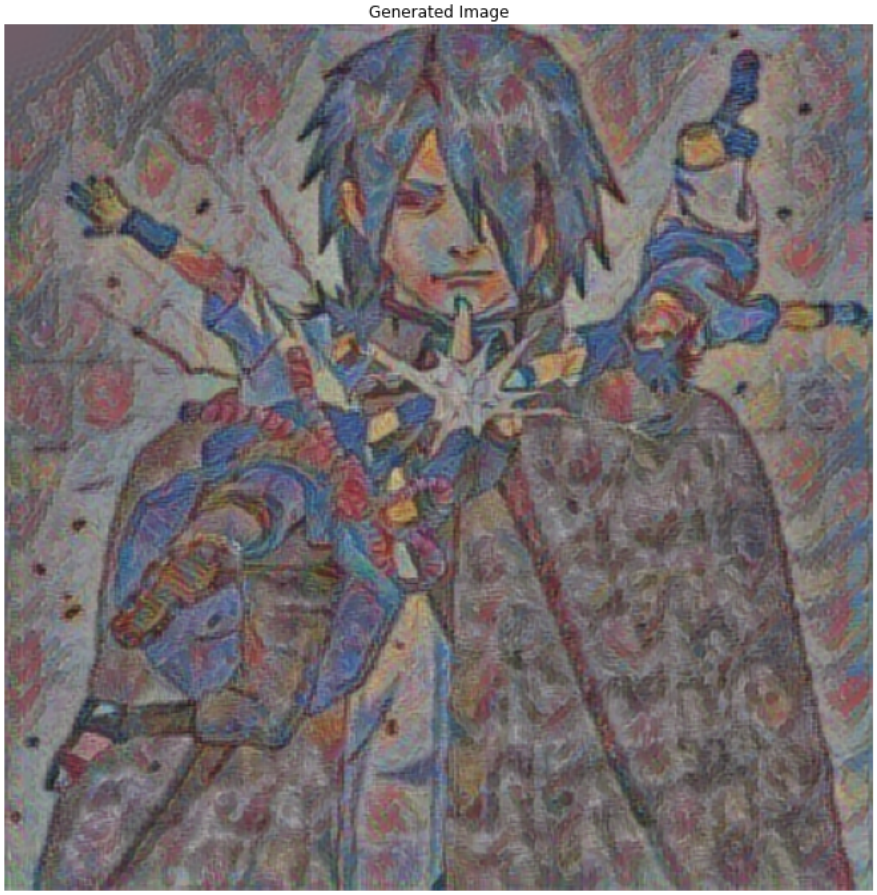
Style 1 Image

Style 2 Image

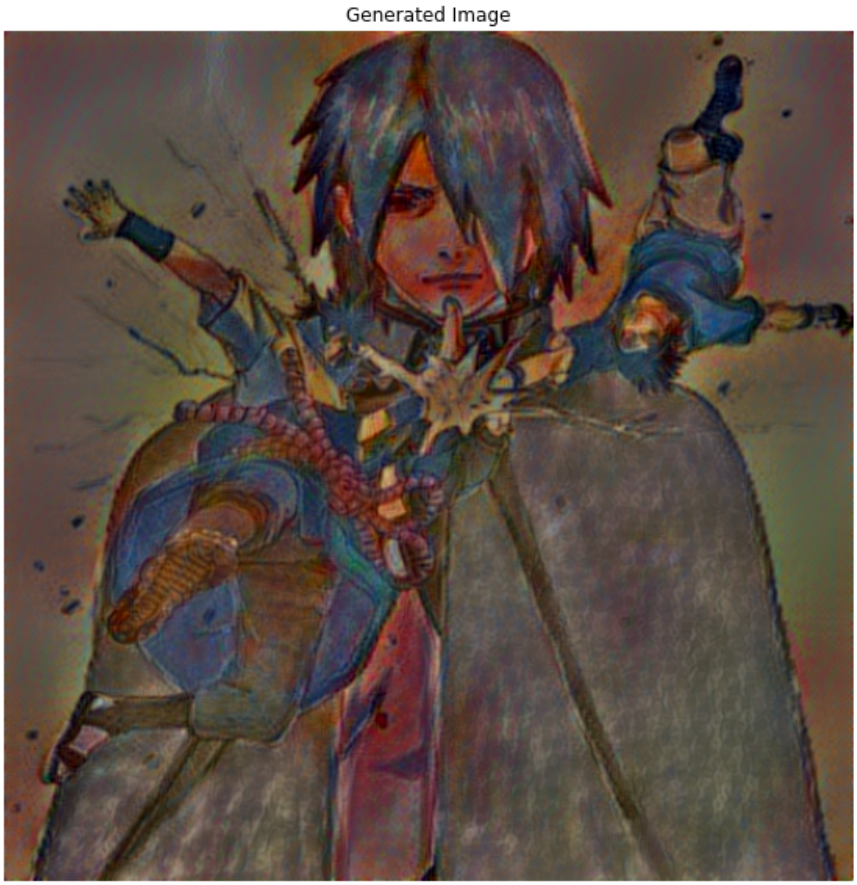
Style 3 Image

Library function output

Generated Image



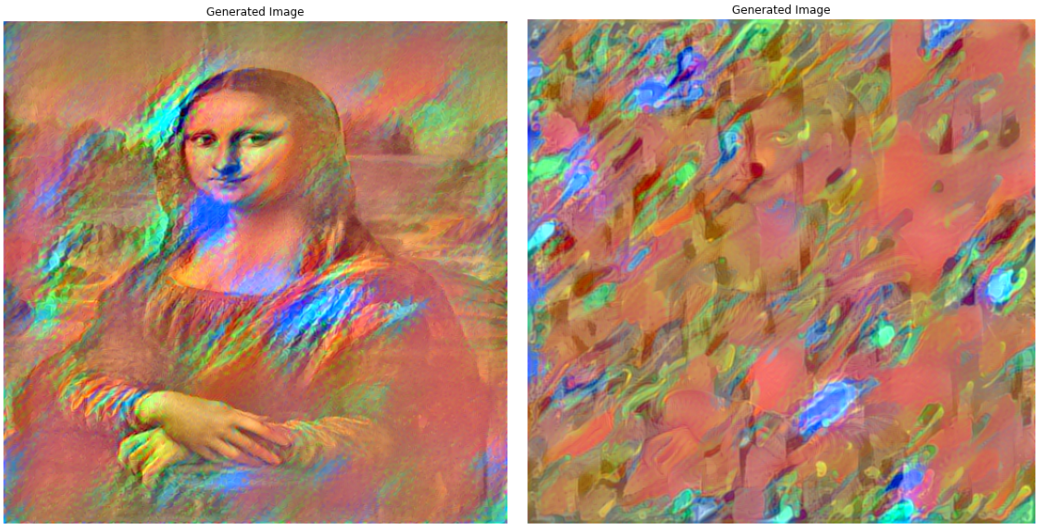
Manual Implementation output



Conclusions

Different experiments were performed and outputs were observed and analyzed. This lead us to the following limitations:-

- For each set of content and style images, we have to do fine variations in weight values for the output to be better. It is not possible to have a fixed set of weights which work on all images. If the weights are not right then the output might be unvalid like below:



- The time taken for output image generation is almost 8 seconds per iteration. We need atleast 10 iterations to get a valid output. This can further be reduced using an end-to-end CNN model built specifically for NST as introduced in Johnson et al.

References

- Tensorflow NST tutorial
- Gatys et al
- NST explanation article.
- Johnson et al.