



# Project Report

## Neural Style Transfer



Name: Atharva Sonare

Enrollment no: 22116022

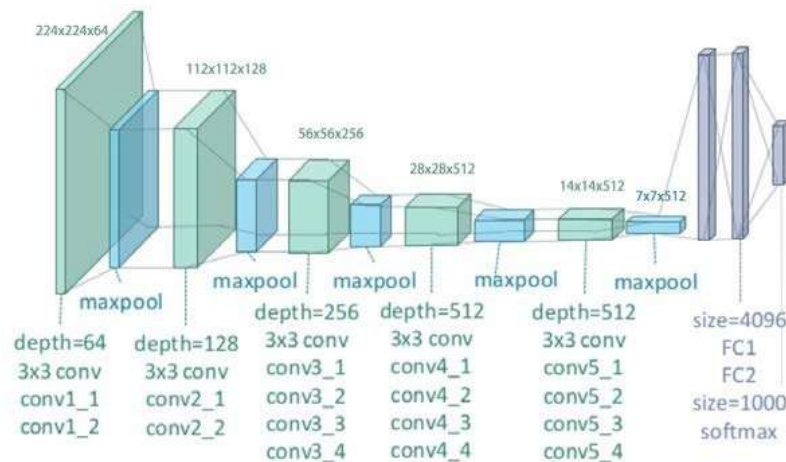
Dept & Year: ECE III

June 2024

# Introduction

The aim of the project was to build a model that allows you to take images and build new images with different artistic styles. So, in order to accomplish this task, using the research paper A Neural Algorithm of Artistic Style, we implemented our own version of Neural style transfer, that generates well balanced images having good representation of content and style, we use VGG 19 pretrained network to compute losses and apply optimizer on image itself.

## Model and Architecture



VGG19 is a variant of the VGG model which in short consists of 19 layers (16 convolution layers, 3 Fully connected layer, 5 MaxPool layers and 1 SoftMax layer). It was trained on millions of images, as a result, the network learnt rich feature representations for a wide range of images, this allows us to use it perform different tasks using it, with our focus primarily on Neural style transfer. We set all the RELU layers as inplace = False, because they cause problems for content loss function discussed ahead.

```
# Load VGG19 model
weights = models.VGG19_Weights.IMAGENET1K_V1
model = models.vgg19(weights=weights).features.to(device).eval()

def set_relu_inplace_false(model):
    for module in model.modules():
        if isinstance(module, nn.ReLU):
            module.inplace = False
```

# Dataset Used

A small dataset named Neural Style Transfer on Kaggle was used. It contains 14 images in total

- Content – 7 images
- Style – 7 images

Link: <https://www.kaggle.com/datasets/burhanuddinlatsaheb/neural-style-transfer/data>

# Libraries

- Torch
- PIL
- Torch vision
- Matplotlib
- NumPy

# Pre / Post processing

**Loader:** Consists of transforms such as Resize, ToTensor() and Normalize

**Unloader:** Consists of transform such as Normalize (but with mean as -m/s and std as 1/s instead to de-normalize) , torch. clamp to ensure that tensor values are in valid range, ToPILImage(), to get an image as output.

# Loss Functions

**Content Loss:** It measures how different is generated image from content image, helps in retaining structure and features of content image.

$$\mathcal{L}_{\text{content}}(x, p) = \frac{1}{2N} \sum_{i=1}^N (x_i - p_i)^2$$

where  $x$  is the feature representation of the generated image, and  $p$  is the feature representation of the content image.

**Style Loss:** It measures the difference between style and generated image, ensures that the generated image captures texture and patterns from style image.

$$\mathcal{L}_{\text{style}}(a, x) = \frac{1}{4N^2M^2} \sum_{i,j} (G_{ij}^a - G_{ij}^x)^2$$

where  $G^a$  is the Gram matrix of the style image, and  $G^x$  is the Gram matrix of the generated image.

Gram Matrix

$$G_{ij} = \frac{1}{N} \sum_k F_{ik} F_{jk}$$

$$\mathcal{L}_{\text{total}} = \alpha \mathcal{L}_{\text{content}} + \beta \mathcal{L}_{\text{style}}$$

Where  $\alpha$  is content weight and  $\beta$  is style weight

# Training / Prediction

```
def forward(x, layer_indices, model):  
    output = []  
    for i, layer in enumerate(model):  
        x = layer(x)  
        if i in layer_indices:  
            output.append(x)  
    return output
```










Forward Function: It passes out input through all the layers of the network while returning the output of desired layers back to us, this allows us to take the feature representation of images through specific layers.

1. Optimizer = Adam
2. Learning Rate = 0.01
3. GPU: Kaggle P100
4. Epochs = 1000
5. Style weight =  $1e6$
6. Content weight = 1
7. Input = clone of content image

Its quite interesting to know that we are training the image itself and use the network to get only content and style losses, hence the optimizer updates the input image itself. So, this could be called prediction instead of training.

All these parameters were found out through hyperparameter tuning over a fixed search space.

## Evaluation

CONTENT	STYLE	GENERATED
		
		
		



# Limitations and Challenges

- GPU limits on Kaggle and 12 hour running limits in Kaggle presented an obstacle to tuning parameters.
- Although the pipelines produce good results, it may not always be able to apply styles to very complex images.
- Training requires lots of time, which restrains us from running hyper parameter tuning for too long, hence we can run less no of trials, which means we could have potentially better hyper parameters. Also, a reason why GPUs are must if you want results quickly.

## Observations

- Interesting results were observed when I used another version of my code, although the image quality produced degraded a little bit, it was able to better apply style, the major difference was normalization method and learning rate in both version of code.



Primary version of code.



Another version of code.

- If you don't clamp values in the unloader or at all, the pixels will explode, alternatively if you clamp the output itself you will get color loss.



Clamp in unloader



Clamp output

# References

A Neural Algorithm of Artistic Style: <https://arxiv.org/pdf/1508.06576>

Neural Transfer using Pytorch:

[https://pytorch.org/tutorials/advanced/neural\\_style\\_tutorial.html](https://pytorch.org/tutorials/advanced/neural_style_tutorial.html)