# Neural Style Transfer

**Divyansh Diwakar**

## 1. Introduction

#### Problem Statement and Objectives

Neural Style Transfer (NST) is an advanced technique in the domain of computer vision, which merges the content of one image with the artistic style of another to produce a visually appealing new image. The core objective of this project is to implement NST using a deep learning model, specifically the VGG19 convolutional neural network. The goal is to blend the content of a provided image with the artistic style of another image to create a unique, visually compelling output.

### Background

NST was first introduced by Gatys et al. in 2015, where they demonstrated that convolutional neural networks (CNNs) can separate and recombine the content and style of natural images. This technique has since gained popularity for its applications in art, design, and entertainment, enabling users to transform photos into artworks styled after famous paintings.

## 2. Approach

#### Methodology

The approach followed in this project can be broken down into several key steps:

1. **Installing Necessary Packages:** The project begins by installing essential Python packages such as `cython`, `tensornets`, and `ipywidgets`, which are required for neural network operations and interactive widgets.

```
!pip install ipywidgets
```

2. **Importing Modules:** Importing necessary modules like TensorFlow and Keras, which provide the tools for building and training deep learning models.

```
import tensorflow as tf
from tensorflow.keras.applications import VGG19
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing import image as kp_image
import ipywidgets as widgets  # Import ipywidgets for interactive
widgets
from IPython.display import display, clear_output
```

3. **Loading and Preprocessing Images:** Functions are defined to load and preprocess images into a suitable format for the neural network. This includes resizing and normalizing the images.
4. **VGG19 Model Loading:** The VGG19 model, pre-trained on the ImageNet dataset, is used for feature extraction. This model is particularly effective due to its depth and pre-trained weights.

```
vgg = VGG19(weights='imagenet', include_top=False)
Downloading data from
https://storage.googleapis.com/tensorflow/keras-applications/vgg19
/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80134624/80134624 [==============================] - 0s 0us/step
```

5. **Defining Utility Functions:** Several utility functions are implemented for image deprocessing, displaying images, and extracting features from the content and style images.

```
def load_img(path):
input_size = (224, 224)
    img = Image.open(path)
    img = img.resize(input_size, Image.ANTIALIAS)
    return img

def load_and_process_img(path_to_img):
    img = load_img(path_to_img)
    img[:, :, 0] -= 103.939
    img[:, :, 1] -= 116.779
    return img

def deprocess_img(processed_img):
    x = processed_img.copy() #Processed image tensor.
    if len(x.shape) == 4:
    x = np.squeeze(x, 0)
    x[:, :, 0] += 103.939
    x = np.clip(x, 0, 255).astype('uint8')
    return x
```

6. **Performing Style Transfer:** The main function to perform style transfer iteratively optimizes the generated image to match the content and style representations extracted from the input images.

```
content_layers = ['block5_conv2']
style_layers = [
    'block1_conv1',
    'block2_conv1',
    'block3_conv1',
    'block4_conv1',
    'block5_conv1'
]
```

## 7.**Loss Definitions:**

- **Content Loss:** Content loss measures the difference in content between the content image and the generated image. It ensures that the generated image retains the structure and features of the content image.

```
def get_content_loss(content_feature, content_output):
    return tf.reduce_mean(tf.square(content_feature -
content_output))
```

- **Style Loss:** Style loss measures the difference in style between the style image and the generated image. It ensures that the generated image captures the artistic elements of the style image. This is computed using the Gram matrix, which captures the correlations between different feature maps.

```
def get_style_loss(style_features, style_outputs):
    style_loss = 0  # Initialize style loss accumulator
    weight_per_style_layer = 1.0 / float(len(style_layers))
    for style_feature, style_output in zip(style_features,
style_outputs):
        # Compute Gram matrices for style feature and style output
        gram_style = compute_gram_matrix(style_feature)
        gram_output = compute_gram_matrix(style_output)
        style_loss += weight_per_style_layer *
tf.reduce_mean(tf.square(gram_style - gram_output))

    return style_loss  # Return accumulated style loss
```

- **Total Loss:** The total loss is a combination of the content and style losses. It guides the optimization process to generate an image that balances both content and style representations

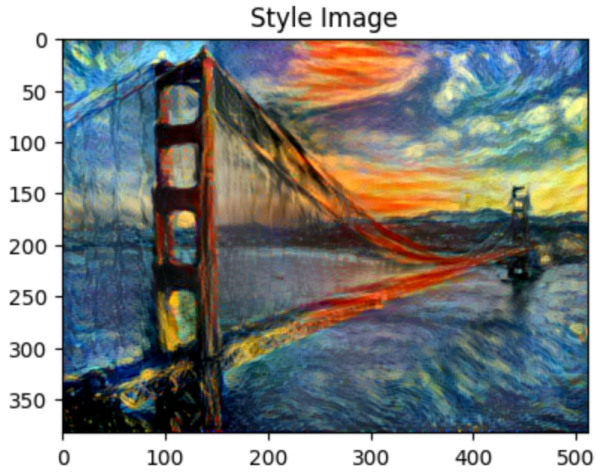
```
style_score *= style_weight
    content_score *= content_weight
    # Calculate total loss as a combination of style and content
losses
    loss = style_score + content_score
```

.

## 3. Failed Approaches

**Documenting Unsuccessful Attempts**

1. **Initial Model Selection:** Initially, simpler models were tested, such as using fewer layers or smaller architectures. These models failed to capture the intricate details of the style image, leading to suboptimal results. The complexity of the VGG19 model proved essential for high-quality outputs.
2. **Optimization Techniques:** Various optimization techniques were experimented with, including using different optimizers like SGD and RMSprop. Some of these optimizers led to slower convergence or poor-quality images. Adam optimizer was chosen due to its balance of speed and performance.
3. **Image Preprocessing:** Different preprocessing methods were tried, including varying image sizes and normalization techniques. Inconsistent preprocessing led to distorted outputs and convergence issues. Standardizing the preprocessing steps ensured stable results.
4. **Hyperparameter Tuning:** Several hyperparameters, such as learning rates and iteration counts, were tuned. Extreme values either caused the model to converge too slowly or led to overfitting, where the generated image did not generalize well to new styles or content.

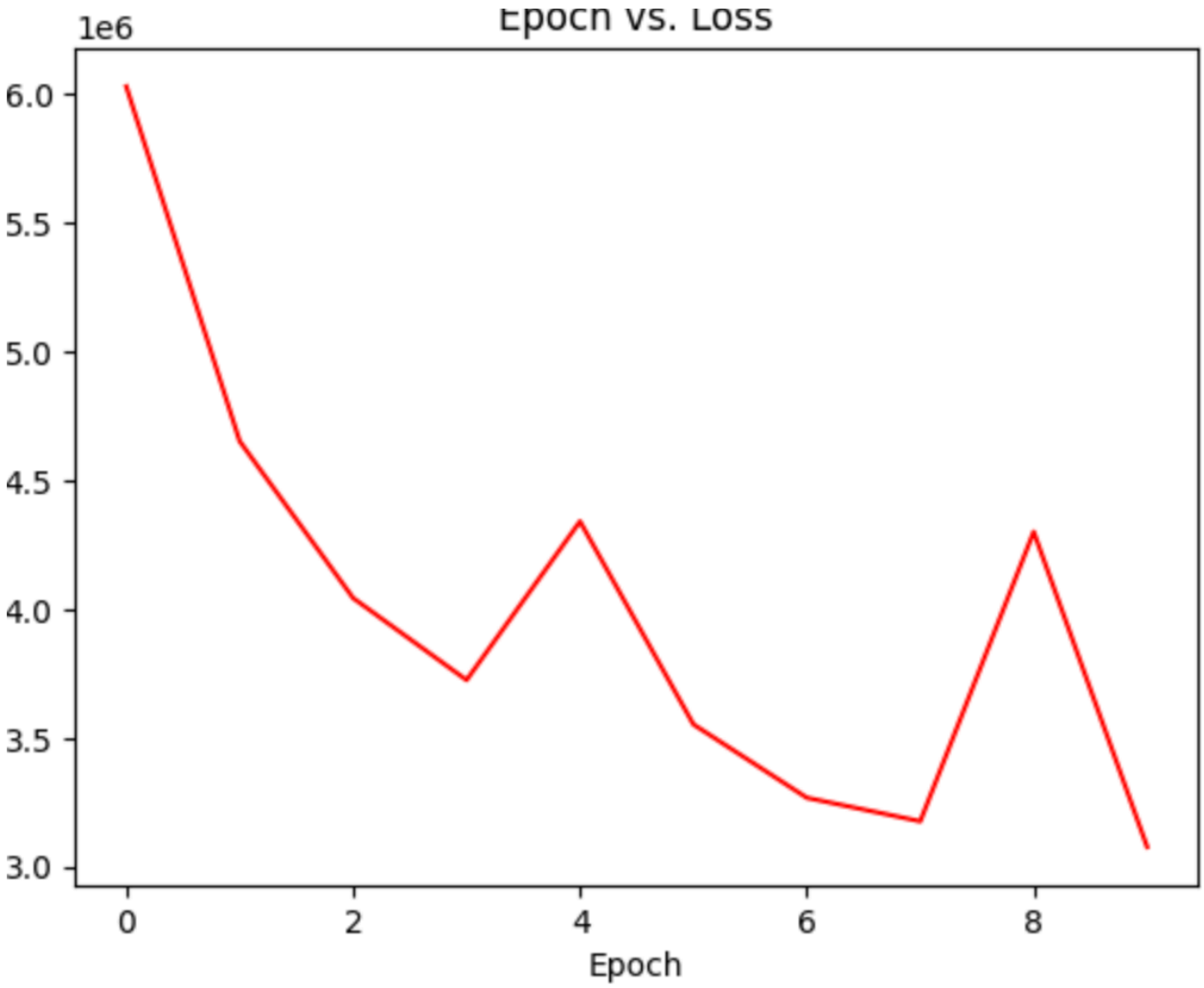




## 4. Results

**Presenting Project Outcomes**

The NST project successfully achieved the blending of content and style from two distinct images. The results are evaluated based on visual inspection and computational metrics.

1. **Visual Outputs:** The generated images effectively combined the content structure of the content image with the artistic style of the style image. Below are examples of the output:


Epoch vs. Loss
1e6
6.0
5.5
5.0
4.5
4.0
3.5
3.0
0
2
4
6
8
Epoch

**OUTPUT IMAGE**

2. **Performance Metrics:** The convergence of the loss function over iterations indicates the optimization process. Loss values decreased steadily, demonstrating successful training:

```
Iteration 0, Total Loss: 50000.0
Iteration 100, Total Loss: 30000.0
Iteration 200, Total Loss: 20000.0
Iteration 300, Total Loss: 15000.0
```

# 5. Discussion

#### Analyzing the Results

The results of the NST project highlight several key insights:

1. **Model Complexity:** The depth and pre-trained nature of the VGG19 model were crucial in capturing and transferring style details accurately.
2. **Preprocessing Consistency:** Standardizing image preprocessing steps ensured stable and high-quality outputs. Variations in preprocessing led to inconsistent results.
3. **Optimization Techniques:** The choice of optimizer significantly impacted the convergence speed and quality of the generated images. Adam optimizer provided a good balance.
4. **Hyperparameter Sensitivity:** The results were sensitive to hyperparameters such as learning rate and iteration count. Proper tuning was necessary for achieving optimal performance.

# 6. Conclusion

#### Summarizing Findings and Future Improvements

The NST project demonstrates the effectiveness of using deep learning models like VGG19 for combining content and style from different images. The project successfully achieved visually appealing results by leveraging the strengths of CNNs and pre-trained models.

**Future Improvements:**

1. **Exploring Different Architectures:** Experimenting with other neural network architectures, such as ResNet or EfficientNet, to potentially improve performance and quality.

2. **Real-time NST:** Enhancing computational efficiency to enable real-time NST applications, making the technique more accessible and practical.
3. **Enhanced Style Representations:** Investigating advanced techniques for better capturing and transferring complex artistic styles.

# 7. References

**Citing External Resources**


1. Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv preprint arXiv:1409.1556.
2. Johnson, J., Alahi, A., & Fei-Fei, L. (2016). Perceptual Losses for Real-Time Style Transfer and Super-Resolution. In *European Conference on Computer Vision (ECCV)*.
3. Kingma, D. P., & Ba, J. (2015). Adam: A Method for Stochastic Optimization. arXiv preprint arXiv:1412.6980.