# Neural Style Transfer Project Report

By: Kamakhya Mehta, 22112049

## Introduction

- Problem Statement:

Neural Style Transfer (NST) is a technique that allows us to combine the content of one image with the style of another image. This powerful technique has applications in digital art, photo editing and creative industries, where it can transform ordinary images into artistic masterpieces. The primary objective of this project is to implement NST using the VGG19 model and TensorFlow, generating an image that maintains the content of the target image while adopting the artistic style of the style image.

- Objectives:

- Preprocess and load the content and style images to ensure compatibility with the VGG19 model.

- Build a VGG19 model to extract features from the content and style images.

- Define and compute content, style, and total variation losses to measure the similarity between images.

- Optimize the generated image using gradient descent to achieve the desired style transfer.

- Document and analyse any challenges and failed attempts encountered during the process.

# Approach

- Methodology:

The implementation of Neural Style Transfer involves several key steps:

1. **Preprocessing Images**: Load and preprocess the content and style images to ensure they are compatible with the VGG19 model.

2. **Building the Model:** Use the pre-trained VGG19 model to extract features from the content and style images.

3. **Defining Loss Functions**: Calculate the content loss, style loss, and total variation loss to guide the optimization process.

4. **Optimization**: Use gradient descent to minimize the combined loss and update the generated image iteratively.

5. **Postprocessing**: Convert the generated image back to a displayable format and save it.

- Steps Followed:

## 1. Preprocessing Images

- **Loading Images**: Load the content and style images using TensorFlow's load_img and img_to_array functions.

- **Resizing Images**: Resize the images to a fixed height (224 pixels) while maintaining the aspect ratio.

- **Preprocessing**: Preprocess the images using VGG19's preprocessing function to ensure they are in the correct format for the model.

## 2. Building the Model

- **Loading VGG19**: Load the VGG19 model without the top fully connected layers, using pre-trained ImageNet weights.

- **Feature Extraction**: Extract features from multiple layers of the VGG19 model to capture both content and style representations.

## 3. Defining Loss Functions

- **Content Loss:** Calculate the mean squared difference between the features of the content image and the generated image at a specific layer.

- **Style Loss**: Use the Gram matrix to capture style information and calculate the mean squared difference between the style image and the generated image across multiple layers.

- **Total Variation Loss:** Encourage spatial smoothness in the generated image to reduce noise.

## 4. Mixed Precision Training

- Implementation:

To speed up the training process and utilize GPU resources efficiently, mixed precision training was implemented. This approach uses both 16-bit and 32-bit floating point types during training to reduce memory usage and increase computational efficiency.

- Benefits

- **Faster Training**: Mixed precision allows for faster computation.

- **Reduced Memory Usage**: Using 16-bit precision reduces memory consumption, allowing for larger batch sizes or more complex models.

- Implementation Details
- **Setting Global Policy**: Used tf.keras.mixed_precision.set_global_policy('mixed_float16') to enable mixed precision globally.

- **Loss Scaling**: The optimizer was wrapped with tf.keras.mixed_precision.LossScaleOptimizer to prevent numerical underflow during training.

## 5. Optimization

- **Combined Loss:** Combine the content, style, and total variation losses into a single loss function.

- **Optimizer:** Use the Adam optimizer to iteratively update the generated image by minimizing the combined loss.

## 6. Postprocessing

- **Converting Format**: Convert the generated image back to RGB format and reverse the preprocessing steps.

- **Saving Image**: Save the generated image at regular intervals during the optimization process to track progress.

# Failed Approaches

- **Initial Attempt with High Learning Rate**

- **Approach**: Started with a high learning rate of 100.0.

- **Failure Reason**: The loss became infinite due to large updates in the generated image, causing instability.

- **Solution**: Reduced the learning rate to 5.00 for more stable updates.

- **Incorrect Tensor Operations**

- **Approach**: Flattened the image tensor directly for optimization.

- **Failure Reason**: Tensor operations were not handled properly, leading to shape mismatches and errors.

- **Solution**: Ensured correct reshaping and tensor operations, using tf.Variable for the combination image.

- **Inadequate Loss Normalization**

- **Approach**: Used tf.reduce_sum for content and style losses.

- **Failure Reason**: This led to large loss values, causing instability during optimization.

- **Solution**: Normalized the losses using tf.reduce_mean to prevent excessively large values.

- **Attempt Without Mixed Precision Training**

- **Approach**: Initially used single-precision floating-point (float32) operations for all computations.

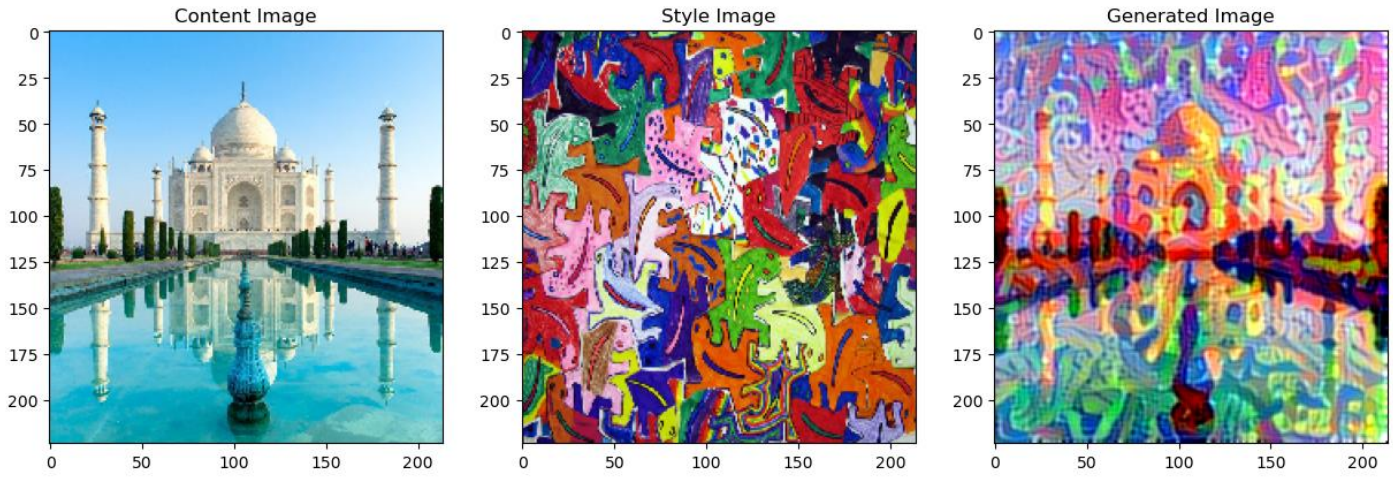
- **Failure Reason**: Computations were slower and more resource-intensive, leading to prolonged training times.

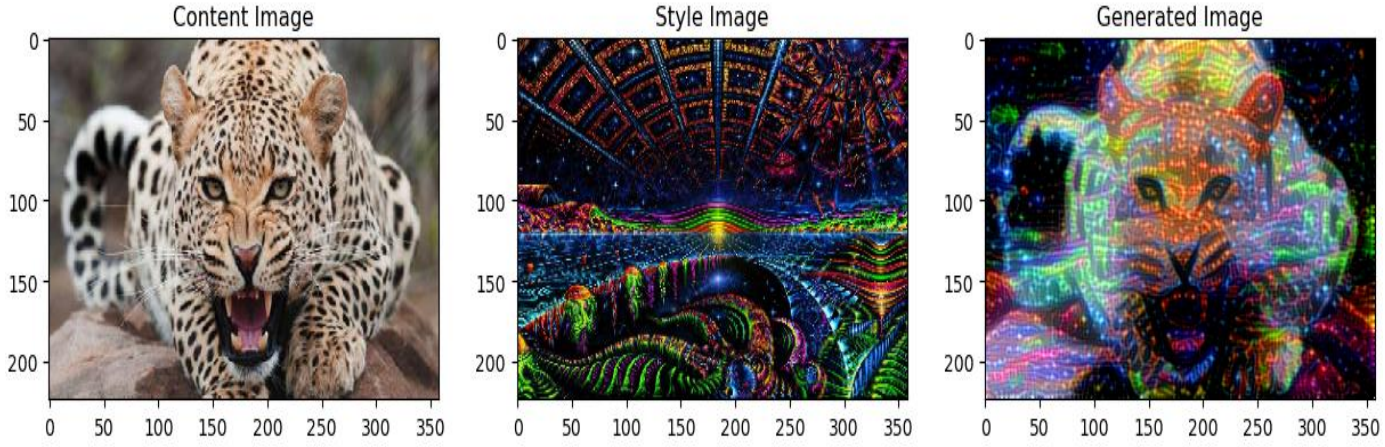
- **Solution**: Implemented mixed precision training by using tf.keras.mixed_precision.set_global_policy('mixed_float16') to speed up training and reduce memory usage.

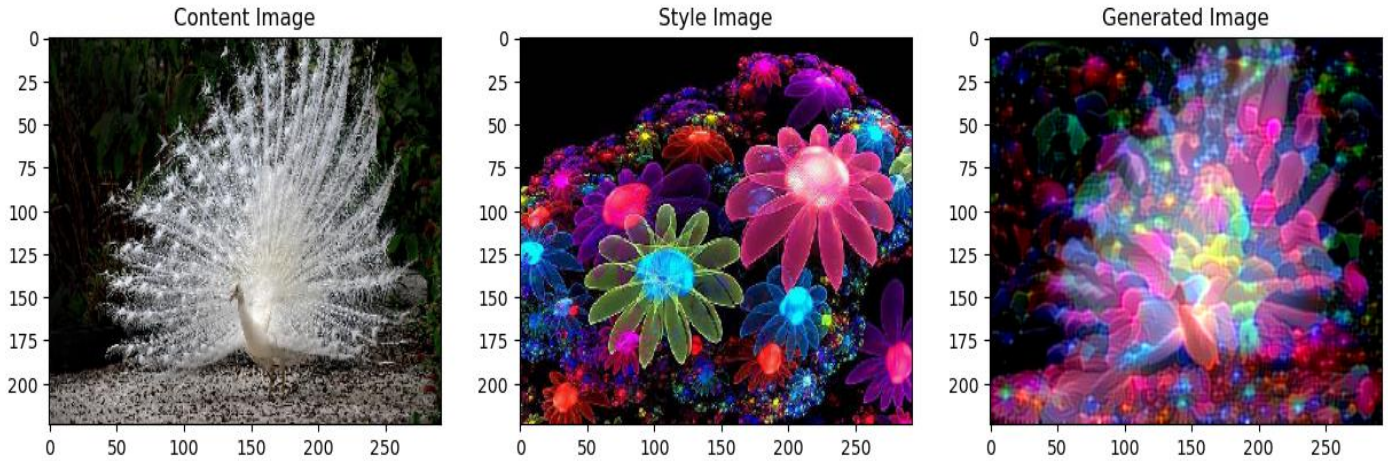
# Results

- 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:



- Metrics

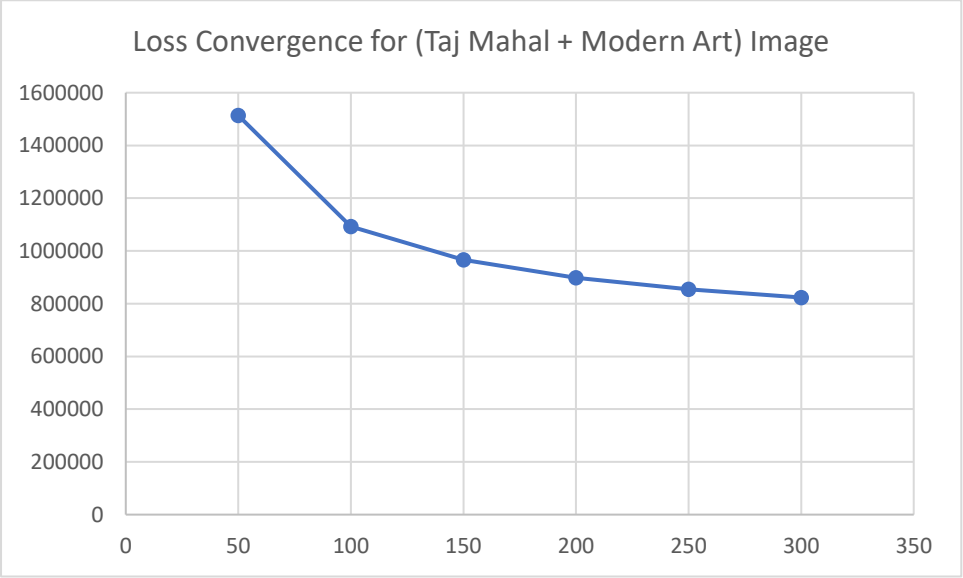
- **Content Loss**: Measured the preservation of content between the target and generated images.

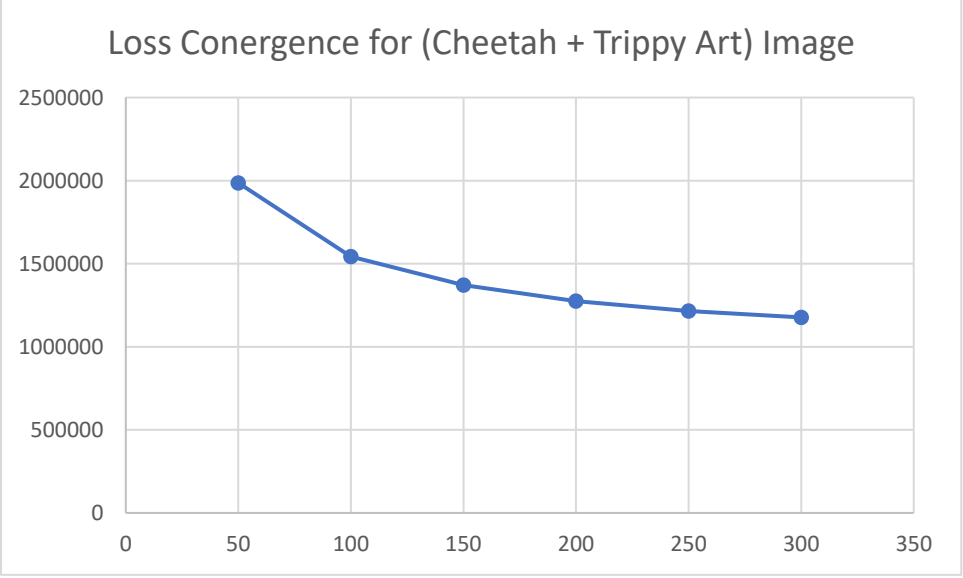
- **Style Loss**: Measured the adoption of style from the style image to the generated image.

- **Total Loss**: Combined the content, style, and total variation losses to guide optimization.

- Graphs

The following graph shows the convergence of the total loss over iterations:



# Discussion

- Analysis of Results

- **Content Preservation**: The generated images retained the structure and recognizable features of the content image.

- **Style Transfer**: The generated images effectively adopted the colour patterns and textures of the style image.

- **Loss Convergence**: The total loss decreased steadily over iterations, indicating successful optimization.

- Insights

- **Balance Between Content and Style Weights** : Adjusting the weights of content and style losses is crucial to achieve a good balance.

- **Learning Rate**: A lower learning rate ensures stable updates and prevents the loss from becoming infinite.

- **Mixed Precision Training**: Using mixed precision training with tf.keras.mixed_precision.set_global_policy('mixed_float16') significantly improved training speed and reduced memory usage.

# Conclusion

- Summary Findings:

- Successfully implemented Neural Style Transfer using the VGG19 model and TensorFlow.

- Achieved a balance between content preservation and style adoption.

- Improved training efficiency with mixed precision training.

- Addressed and documented challenges encountered during the process.

- Future Improvements

- **Dynamic Weight Adjustment**: Implement dynamic adjustment of content and style weights during optimization.

- **Advanced Optimizers**: Experiment with advanced optimizers like AdamW or LAMB for potentially better convergence.

- **Real-time Style Transfer**: Explore real-time style transfer techniques using faster models like MobileNet.

## References


1. **Original NST Paper**: Leon A. Gatys, Alexander S. Ecker, Matthias Bethge - Neural Algorithm of Artistic Style.

2. **TensorFlow Documentation**: https://www.tensorflow.org/tutorials/generative/style_transfer

3. **Blogs** : https://towardsdatascience.com/an-intuitive-understanding-to-neural-style-transfer-e85fd80394be