Custom Artistic Neural Style Transfer

Onkar Shinde (220150021)
DA312 Advanced Machine Learning Lab

April 20, 2025

Abstract

This project implements a **custom neural style transfer system** from scratch using PyTorch and VGG-19, allowing users to blend the content of one image with the style of another via a visually appealing web interface. It focuses on feature extraction using convolutional neural networks, Gram matrix-based style representation, and optimization through Adam. The webapp allows real-time configuration and preview of results, offering advanced options like color preservation and histogram-based color correction.

1 Introduction

Neural Style Transfer (NST) is a deep learning technique where the *content* of one image is merged with the *style* of another. This technique, introduced by Gatys et al. (2016), utilizes convolutional neural networks (CNNs) to extract and recombine features from different layers of a pre-trained network like VGG-19. This project aims to develop a fully functional NST engine with an intuitive web interface, empowering users to upload images, tune hyperparameters, and download stylized outputs.

2 Key Objectives

The key objectives of this project are:

- **Artistic Stylization**: Combining content and style using CNN-based feature representations.
- **Deep Learning**: Using VGG-19 as the backbone for feature extraction.
- **Optimization**: Implementing iterative optimization using algorithms like Adam or L-BFGS.
- **Web Interface**: Allowing users to control parameters and visualize results seamlessly.
- **Post-processing**: Offering color preservation and histogram matching to correct color distortions.

3 Methodology

3.1 VGG Architecture for Feature Extraction

We chose VGG-19 due to its layer-wise granularity, which helps to separate content and style features effectively. The content features are typically extracted from deeper layers, such as conv4_2, while style features are captured from earlier layers like conv1_1, conv2_1, and others.

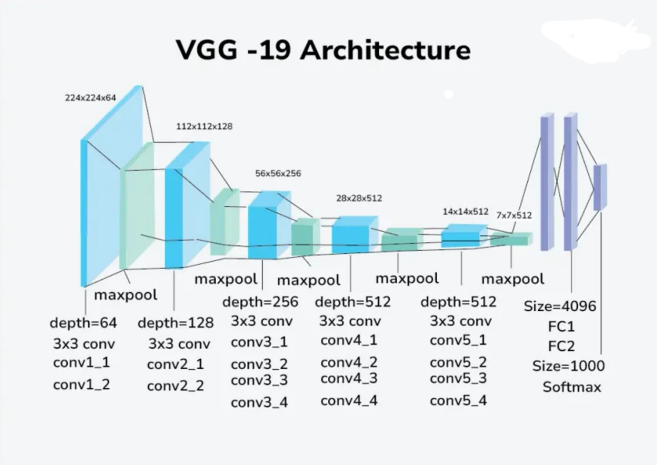

Figure 1: VGG19 Architecture

3.2 Loss Functions

The total loss function used for optimization is a weighted sum of three components: content loss, style loss, and total variation (TV) loss.

$$L_{\text{total}} = \alpha L_{\text{content}} + \beta L_{\text{style}} + \gamma L_{\text{TV}}$$

- **Content Loss**: This measures the difference between the content of the generated image and the content of the original image. It is calculated as the squared Euclidean distance between the feature maps of the content image

and the generated image.

$$L_{\text{content}} = \frac{1}{2} \sum_{i,j} (F_{ij}^C - F_{ij}^G)^2$$

- **Style Loss**: This captures the style of the image using the Gram matrix. The Gram matrix of an image is calculated as the dot product of the feature maps for each layer. The style loss measures the difference in the Gram matrices between the style image and the generated image.

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l$$

$$L_{\text{style}} = \sum_l w_l \cdot \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2$$

- **Total Variation Loss**: This is used to reduce noise in the generated image. It penalizes large spatial variations in pixel values and promotes smoothness in the image.

$$L_{\text{TV}} = \sum_{i,j} (x_{i,j} - x_{i+1,j})^2 + (x_{i,j} - x_{i,j+1})^2$$

4 System Workflow

The workflow of the system can be described as follows:

1. **Upload Content and Style Images**: Users upload both content and style images through the web interface.
2. **Preprocessing and Normalization**: Images are preprocessed and normalized to ensure they are in a suitable format for the VGG-19 network.
3. **Feature Extraction**: VGG-19 is used to extract content and style features from the uploaded images.
4. **Loss Computation**: The content loss, style loss, and total variation loss are computed to evaluate the difference between the generated and target images.
5. **Optimization**: Optimization is performed using either the Adam optimizer or the L-BFGS algorithm to minimize the total loss and improve the stylization.
6. **Stylized Output Image**: The final stylized image is generated based on the optimization process.
7. **Display and Download Results**: The stylized image is displayed on the web interface, where users can preview and download the final result.

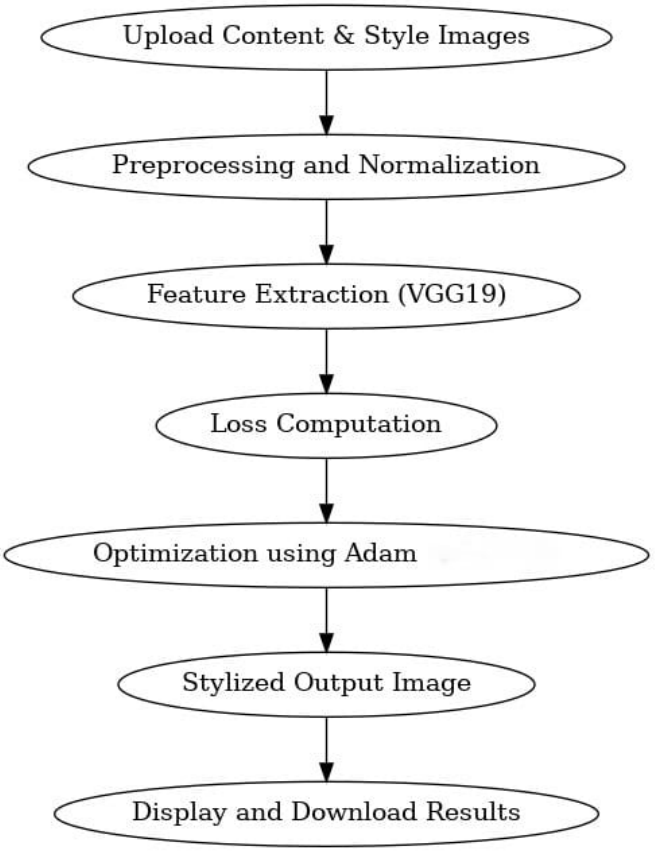

Figure 2: System Workflow

5 Web Interface

The frontend of the system is built using `HTML`, `CSS` (via `Bootstrap`), and `JavaScript`, while the backend is powered by `Flask`. Key features of the interface include:

- File upload functionality for both *content* and *style* images.
- A slider or input box to select the number of output images to generate.
- A loading bar that indicates the progress of image stylization, along with an estimated time.
- A scrollable terminal output section to display logs and messages in real-time.
- Image preview functionality with an option to download the stylized image.

6 Results

Below are the results of the neural style transfer process:

Figure 3: Content Image

Figure 4: Style Image

Figure 5: Output Image

Figure 6: Output Image with Corrected Colors

7 Web Dashboard Preview

Here are some screenshots of the interface and results:

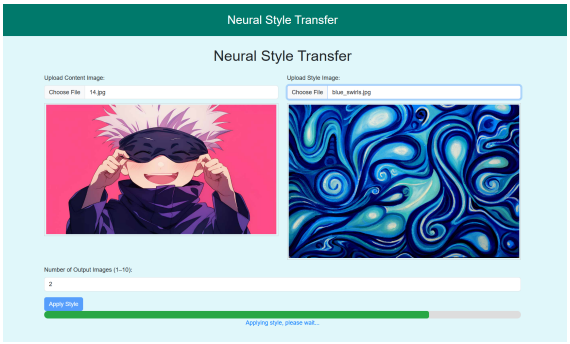

Figure 7: Upload Section of the Web Dashboard

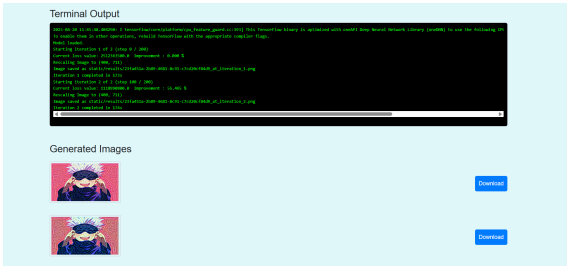

Figure 8: Stylized Results and Download Options

8 Challenges and Solutions

During the project, several challenges were encountered. Below are the key issues along with their respective solutions:

- **High compute cost**: To mitigate this, images were optimized by resizing them to lower resolutions during training. Additionally, efficient techniques were employed to reduce computational load.
- **Color distortion**: This was addressed by integrating color preservation techniques such as histogram matching, ensuring the output image maintained the intended color characteristics.
- **Long iteration time**: Adaptive learning rates were implemented, and users were given control over the number of iterations. This allowed for faster results without a significant compromise in output quality.

9 Conclusion

This project successfully implements a custom neural style transfer pipeline from scratch using PyTorch, VGG-19, and Flask. The system allows for real-time generation

of stylized images while giving users full control over parameters such as content and style blending. This demonstrates the power of deep learning techniques, particularly convolutional neural networks, in generating artistic transformations.

10 References

1. Gatys, L. A., Ecker, A. S., & Bethge, M. (2016). *Image Style Transfer Using CNNs.* CVPR.

2. Simonyan, K., & Zisserman, A. (2014). *Very Deep Convolutional Networks for Image Recognition.* arXiv:1409.1556.

3. Mahendran, A., & Vedaldi, A. (2015). *Understanding Deep Image Representations by Inverting Them.* CVPR.