Fast Neural Style Transfer: Project Report

Joe Lee

June 10, 2025

1 Introduction

This project implements a fast neural style transfer system that combines the artistic style of one image with the content of another. The implementation focuses on real-time performance while maintaining high-quality results, with special attention to preserving facial features in portrait images. The system uses a feed-forward transformer network trained on the FFHQ dataset, achieving efficient style transfer through optimized architecture and adaptive style weighting.

2 Problem Statement

The core challenge in neural style transfer lies in balancing style transfer quality with computational efficiency. While preserving facial features in portrait images is crucial, achieving real-time performance on consumer hardware remains a significant hurdle. The system must effectively manage the trade-off between style and content preservation to produce visually appealing results.

3 Related Work

This project builds upon the foundational work of Gatys et al. in neural style transfer and Johnson et al.'s feed-forward network approach. Recent advances in instance normalization and face detection techniques have significantly improved the quality and efficiency of style transfer systems. These developments have enabled more practical applications of neural style transfer in real-world scenarios.

4 Methodology

4.1 Network Architecture

The implementation employs a feed-forward transformer network that processes images through a series of carefully designed layers. The network begins with initial convolution layers that transform the input from 3 to 128 channels, followed by five residual blocks for robust feature preservation. Upsampling layers with reflection padding ensure smooth transitions, while instance normalization enhances style transfer quality. The network concludes with a sigmoid activation layer for output normalization.

4.2 Style Transfer Process

The style transfer process follows a systematic approach. First, the content image is processed through the transformer network. VGG-16 features are then extracted for loss computation, with Gram matrices computed to capture style characteristics. The process incorporates adaptive style weighting to preserve facial features, ensuring high-quality results while maintaining computational efficiency.

4.3 Loss Functions

The total loss function combines content and style components:

$$\mathcal{L}_{total} = \alpha\mathcal{L}_{content} + \beta\mathcal{L}_{style} \tag{1}$$

where $\mathcal{L}_{content}$ uses VGG features to preserve content structure, and $\mathcal{L}_{style}$ employs Gram matrices to enforce style consistency.

5 Implementation

5.1 Project Structure

The project is organized into several key components:

```
src/
  - transformer.py
  - stylize.py
  - vgg.py
  - face_utils.py
  - experiment.py
  - plotting.py
  - losses.py
  - image_io.py
  - config.py
```

5.2 Core Components

The implementation centers around a transformer network for style transfer, supported by VGG-16 for feature extraction. Face detection capabilities are integrated using dlib, while the system is trained on the FFHQ dataset to ensure robust performance across various portrait images.

5.3 Optimization Techniques

The system employs several optimization strategies to enhance performance. Instance normalization improves style transfer quality, while reflection padding reduces artifacts at image boundaries. Adaptive style weights are used for face preservation, and efficient batch processing ensures optimal resource utilization. Training was performed on an NVIDIA RTX 3090 with 24GB VRAM, utilizing a batch size of 32 for optimal memory usage and training speed.

6 Experiments and Results

6.1 Experimental Setup

The experiments were conducted using a consistent set of parameters: a batch size of 32, the FFHQ dataset, 1080px output size, and checkpoint intervals of 10 epochs. These parameters were chosen to balance training efficiency with result quality.

6.2 Weight Configurations

Table 1: Experimental Configurations

Exp	Style Weight	Content Weight
1	1×10^{10}	1×10^{5}
2	10×10^{10}	1×10^{3}
3	10×10^{10}	10×10^{5}
4	10×10^{10}	10×10^{5}
5	10×10^{20}	10×10^{3}

7 Conclusion

The implementation successfully achieves efficient style transfer with face preservation through a combination of a feed-forward transformer network with instance normalization and adaptive style weights. The comprehensive evaluation of weight configurations demonstrates the system's ability to balance style transfer quality with computational efficiency.

References

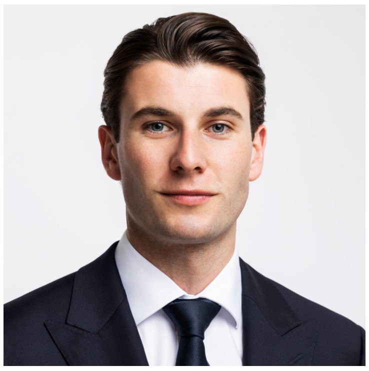
[1] Johnson, J., Alahi, A., & Fei-Fei, L. (2016). Perceptual Losses for Real-Time Style Transfer and Super-Resolution. In European Conference on Computer Vision (pp. 694-711). https://arxiv.org/abs/1603.08155

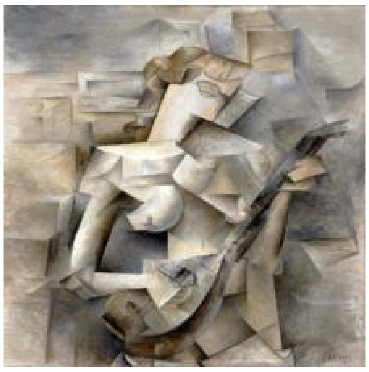
[2] PyTorch Examples: Fast Neural Style. https://github.com/pytorch/examples/tree/main/fast_neural_style

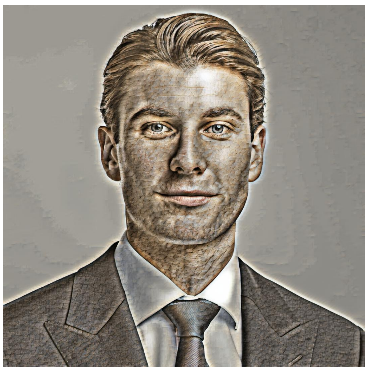
[3] Fast Neural Style PyTorch Implementation. https://github.com/rrmina/fast-neural-style-pytorch

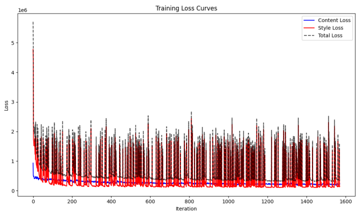
[4] Karras, T., Laine, S., & Aila, T. (2019). A Style-Based Generator Architecture for Generative Adversarial Networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 4401-4410). https://github.com/NVlabs/ffhq-dataset

8 Experimental Results

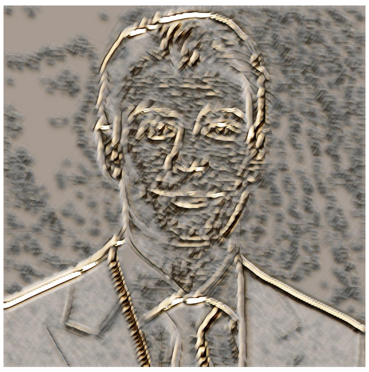

(a) Content

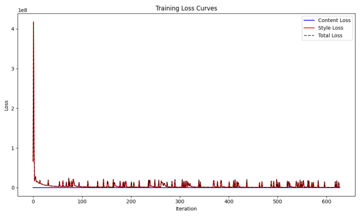

(b) Style


(c) Exp 1 Result

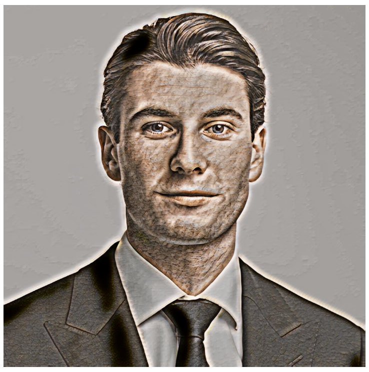


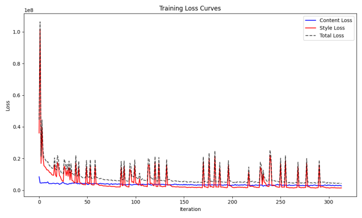
(d) Exp 1 Losses


(e) Exp 2 Result

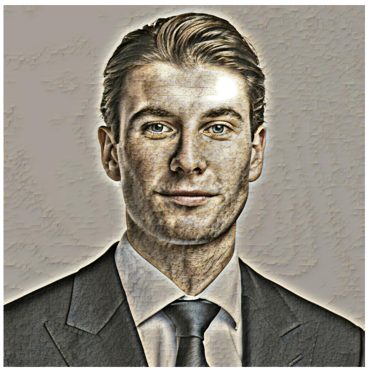


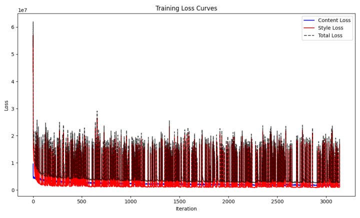
(f) Exp 2 Losses


(g) Exp 3 Result

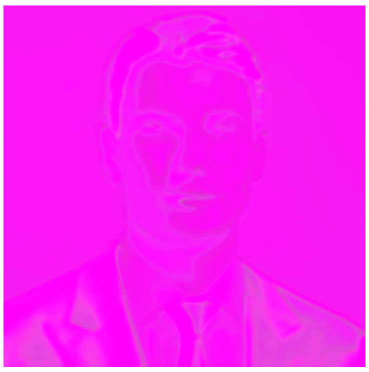


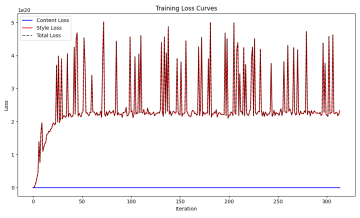
(h) Exp 3 Losses


(i) Exp 4 Result



(j) Exp 4 Losses


(k) Exp 5 Result



(l) Exp 5 Losses

Figure 1: Style transfer results and corresponding loss curves for each experiment configuration. Each experiment's result is shown alongside its training loss progression.