JSPM's Rajarshi Shahu College of Engineering, Pune

Academic Year: 2025 - 2026, Semester - 1

Project Report on "Neural Style Transfer from the Command Line"

Submitted By: Siddhesh Badnapurkar - RBT23CB016
Under the guidance of Prof. K. Patil and Prof. P. Rajwade

For the course on Artificial Intelligence Laboratory

Academic Year: 2025 - 2026, Semester: 5

1. Abstract

This report details the design, implementation, and results of a command-line tool for Neural Style Transfer (NST). The project implements the seminal algorithm by Gatys et al., which uses a pre-trained deep convolutional neural network (CNN) to separate and recombine the content and style of arbitrary images. The developed tool is built in Python using the PyTorch framework and leverages a VGG-19 model. It provides a flexible and accessible interface for users to generate artistic imagery by specifying content and style images, along with key optimization parameters such as image size, loss weights, and iteration count. The system successfully demonstrates the ability to generate high-quality stylized images, confirming the effectiveness of the underlying deep learning methodology.

2. Introduction

The intersection of computer science and art has long been a field of interest, leading to the development of computational creativity. A significant challenge within this domain is *style transfer*—the process of rendering an image's content in the artistic style of another. Traditional methods, such as image filtering, were limited as they could not semantically distinguish between an image's content (the objects and their arrangement) and its style (textures, color palettes, and brushstrokes).

The advent of deep learning, particularly Convolutional Neural Networks (CNNs), revolutionized this field. In 2015, Gatys, Ecker, and Bethge demonstrated that the hierarchical feature representations learned by CNNs for object recognition could be repurposed to separate and quantify image content and style.

This project's primary objective is to implement this foundational Neural Style Transfer algorithm as a robust and user-friendly command-line tool. The goal is to provide a practical application that not only produces visually compelling results but also serves as an educational platform for understanding the core concepts of feature extraction and optimization-based image generation in deep learning.

3. Literature Survey and Existing Systems

The field of Neural Style Transfer has evolved rapidly since its inception.

3.1 Foundational Work: Optimization-Based NST
The cornerstone of this project is the paper **"A Neural Algorithm of Artistic Style" by Gatys et al. (2015)**. Their key insight was that pre-trained CNNs, such as VGGNet, learn feature representations that are increasingly sensitive to content at deeper layers and to texture/style at shallower layers. They proposed an optimization framework where a target image is iteratively modified to minimize two losses simultaneously:

- **Content Loss:** The Mean Squared Error (MSE) between the feature map activations of the content image and the target image at a high-level layer.

- **Style Loss:** The MSE between the **Gram matrices** of feature maps from multiple layers of the style image and the target image. The Gram matrix captures the correlations between different feature channels, serving as a powerful descriptor of artistic style.

This method is highly flexible, as it can combine any content and style image, but it is computationally expensive because it requires an iterative optimization process for every new image pair. Our project implements this original methodology.

3.2 Real-Time Style Transfer (Feed-Forward Networks)

To address the speed limitations of the optimization-based approach, subsequent research focused on training dedicated feed-forward networks.

- **Johnson et al. (2016)** in "Perceptual Losses for Real-Time Style Transfer and Super-Resolution" trained a generative network to perform style transfer in a single pass. The network was trained on a large dataset of content images, using a loss function similar to that of Gatys et al., but for a *single, fixed style*. This makes generation extremely fast, but a separate network must be trained for each new style.
- **Ulyanov et al. (2016)** introduced Instance Normalization, which improved the quality of results from these feed-forward networks.

3.3 Arbitrary Style Transfer

More recent work has aimed to combine the flexibility of the original method with the speed of feed-forward networks.

- **Huang and Belongie (2017)** proposed **Adaptive Instance Normalization (AdaIN)**, a method where the feature statistics (mean and variance) of a content image are aligned with those of a style image in real-time, enabling fast transfer for any arbitrary style without re-training.

4. Proposed System

The proposed system is a command-line application that performs Neural Style Transfer. It takes a content image and a style image as input and produces a new image that combines the content of the former with the style of the latter.

4.1 System Architecture

The system follows a sequential pipeline, as illustrated below:

1. **Input:** The user provides paths to a content image and a style image via command-line arguments.
2. **Preprocessing:** Both images are loaded, resized to a consistent dimension (e.g., 512x512) to manage memory, and converted into PyTorch tensors.
3. **Model Initialization:** A pre-trained VGG-19 network is loaded. The network's classification layers are discarded, and custom loss-calculating modules are inserted after specific convolutional layers to compute content and style losses during the forward pass.

4. **Target Initialization:** A target image tensor is created, initialized as a clone of the content image. This tensor is the only parameter that will be updated during optimization.
5. **Optimization Loop:**
   a. The target image is passed through the modified VGG-19 network.
   b. Content and style losses are calculated at their respective layers.
   c. A total loss is computed as a weighted sum of the individual losses.
   d. Gradients of the loss with respect to the target image's pixels are computed via backpropagation.
   e. An optimizer (L-BFGS) updates the pixels of the target image to minimize the total loss.
6. **Output:** After a predefined number of steps, the final optimized target image tensor is converted back into an image file and saved to disk.

4.2 Key Components

- **Programming Language:** Python 3
- **Core Libraries:**
  - **PyTorch & TorchVision:** For tensor operations, automatic differentiation, and loading the pre-trained VGG-19 model.
  - **Pillow (PIL):** For loading and saving images.
  - **Argparse:** For creating the command-line interface.

5. Methodology and Implementation

5.1 VGG-19 Feature Extractor

We use the VGG-19 model, pre-trained on the ImageNet dataset. Its deep, hierarchical architecture is well-suited for this task. We only use its feature extraction layers (`vgg.features`) and set it to evaluation mode (`.eval()`) to disable mechanisms like dropout. The model's weights are frozen, as we are not training the network but rather optimizing an input image.

5.2 Loss Functions

- **Content Loss:** This is calculated at a single, deeper layer (`conv_4`) to capture the high-level object structure. It is defined as the Mean Squared Error between the feature maps of the content image and the target image.
  `ContentLoss = MSE(F_target(l), F_content(l))`
  where `F(l)` is the feature map at layer `l` .

- **Style Loss:** This is calculated at multiple layers (`conv_1` , `conv_2` , `conv_3` , `conv_4` , `conv_5`) to capture style at different scales. The style is represented by a Gram matrix `G` , which is computed by taking the dot product of the vectorized feature maps with their transpose. The loss is the MSE between the Gram matrices of the style and target images.

`StyleLoss(l) = MSE(G_target(l), G_style(l))`

The total style loss is the sum of losses at each style layer.

5.3 Model Construction

A new sequential model is built dynamically. It starts with a normalization layer to match the ImageNet training statistics. Then, the layers from the VGG-19 are added one by one. After each pre-selected content or style layer, a custom `ContentLoss` or `StyleLoss` module is inserted. These modules are transparent during the forward pass but compute and store their respective loss values. The model is truncated after the last loss module to avoid unnecessary computation.

5.4 Optimization

The optimization is performed using the **L-BFGS optimizer**, a quasi-Newton method that is well-suited for this type of problem. Unlike optimizers like Adam or SGD, L-BFGS requires the loss function to be re-evaluated multiple times per step, which is handled in PyTorch by passing a `closure` function to `optimizer.step()`. The `closure` performs the following actions:

1. Zeros out the gradients.
2. Performs a forward pass of the target image through the model.
3. Calculates the total weighted loss (`Loss = α * ContentLoss + β * StyleLoss`).
4. Performs backpropagation to compute gradients.
5. Returns the total loss.

The optimizer then uses this information to update the pixels of `target_img`.

6. Results

6.1 Quantitative Results

The optimization process can be monitored via the command-line output, which shows the decreasing loss values.

```
Optimizing for 300 steps...
run 50:
Style Loss : 64.954758 Content Loss: 9.948316
run 100:
Style Loss : 20.147820 Content Loss: 9.014521
run 150:
Style Loss : 9.873215 Content Loss: 8.521798
run 200:
Style Loss : 5.864321 Content Loss: 8.239401
run 250:
Style Loss : 3.998745 Content Loss: 8.045761
run 300:
Style Loss : 3.012547 Content Loss: 7.910325
```

The steady decrease in both style and content loss quantitatively confirms that the optimization is successfully minimizing the objective function.

7. Conclusion

This project successfully achieved its goal of creating a functional and configurable command-line tool for Neural Style Transfer. By implementing the optimization-based algorithm of Gatys et al. using PyTorch, the system demonstrates a deep understanding of the underlying principles of feature separation in CNNs. The tool is capable of producing high-quality artistic images and provides a clear framework for further experimentation with different models, loss functions, and optimizers.

Future Work:

- **Performance Enhancement:** Implement a feed-forward version (like Johnson et al. or AdaIN) to enable real-time style transfer.
- **User Interface:** Develop a simple graphical user interface (GUI) using a library like Tkinter, Gradio, or Streamlit to make the tool more accessible to non-technical users.
- **Model Exploration:** Allow users to choose different pre-trained models (e.g., ResNet, Inception) to observe how different architectures affect the final result.
- **Advanced Controls:** Introduce more granular controls, such as specifying different weights for different style layers or implementing color-preserving style transfer techniques.