Artificial Intelligence - IA2 report on

Various Neural Style Transfer Implementations

Submitted by
Peeth Chowdhary - 16010122034
Rohit Deshpande - 16010122041
Eeshanya Joshi - 16010122074
Shubh Jalui - 16010122072
Ninad Marathe - 16010122106

Class- TY COMPS C

Table of Contents

Introduction

Neural Style Transfer (NST) is a compelling computer vision technique that blends the content of one image with the artistic style of another. This allows for the transformation of photos into artwork that mimics famous painting styles. Despite its creative appeal, NST poses challenges in balancing content preservation and style reproduction, as well as in optimizing computational performance. Alongside backend optimization using PyTorch and VGG19, we also implemented a real-time web interface using TensorFlow Hub and Streamlit to bring this artistic experience to end users.

Objective

To develop a flexible and efficient style transfer system that:

- Preserves the core content structure of input images.
- Applies complex textures from a reference style image.

- Offers a real-time, user-friendly interface for non-technical users.
- Allows experimentation with optimization strategies and hyperparameters to improve output quality and reduce compute time.

Literature Survey

Paper Title	Methodology	Dataset Used	Observation	Pros	Cons	Findings
Neural Style Transfer: A Review (2017)	Surveyed NST methods and categorizations	Various style/content images	Provided historical and technical context	Comprehensive overview	No new method introduced	Established foundations and evolution of NST techniques
Dynamic Neural Style Transfer using VGG19 (2025)	Flexible VGG19-based method with adjustable style weights	Public art and real-world datasets	Enabled faster convergence and better customization	Dynamic weight tuning, improved efficiency	Requires fine-tuning	Showed promising results with lower computation time and better style control

Problem Statement

How can we improve both the performance and accessibility of neural style transfer by leveraging deep learning optimization and interactive deployment technologies?

Dataset

- **Style Images**: Classic artworks (Van Gogh, Picasso, etc.)
- **Content Images**: High-resolution photos, portraits, landscapes
- **Web Interface Input**: User-uploaded images
- **Source**: Public datasets + user inputs for deployed app

Implementation

Backend Optimization (VGG19, PyTorch)

1. **Imported Libraries**: PyTorch, torchvision, PIL, NumPy.
2. **Loaded and Preprocessed Images**: Resized, normalized, and converted to tensors.
3. **Feature Extractor**: Custom module using pre trained VGG19, extracting multi-scale features from selected layers (0, 5, 10, 19, 28).
4. **Loss Computation**:
    - **Content Loss**: MSE between feature maps.
    - **Style Loss**: MSE between Gram matrices of feature maps.
5. **Optimization**:
    - Image pixels optimized via Adam.
    - Tuned learning rates, alpha-beta weights.
6. **Logging & Output**: Intermediate losses and images were saved for evaluation.

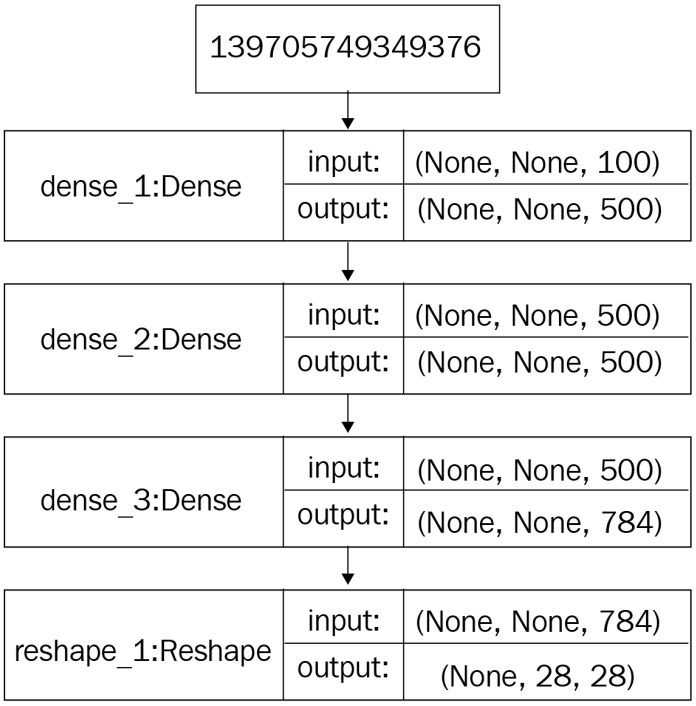
Front-End Deployment (Streamlit + TensorFlow Hub)

1. **Model**: lloaded via TensorFlow Hub.
2. **Framework**: Streamlit-based UI for user interaction.
3. **Image Handling**: Uploaded images resized, normalized, and converted to tensors.
4. **Real-Time Inference**:
    - Stylized output generated instantly using stylize_model(content, style).
5. **Downloadable Output**:
    - Stylized image shown in browser.
    - One-click PNG download button implemented.

Model Architecture

PyTorch-Based Style Transfer

- Pretrained VGG19 encoder
- No decoder – pixels of output image optimized directly
- Tunable hyperparameters for content/style balance



TensorFlow Hub Model

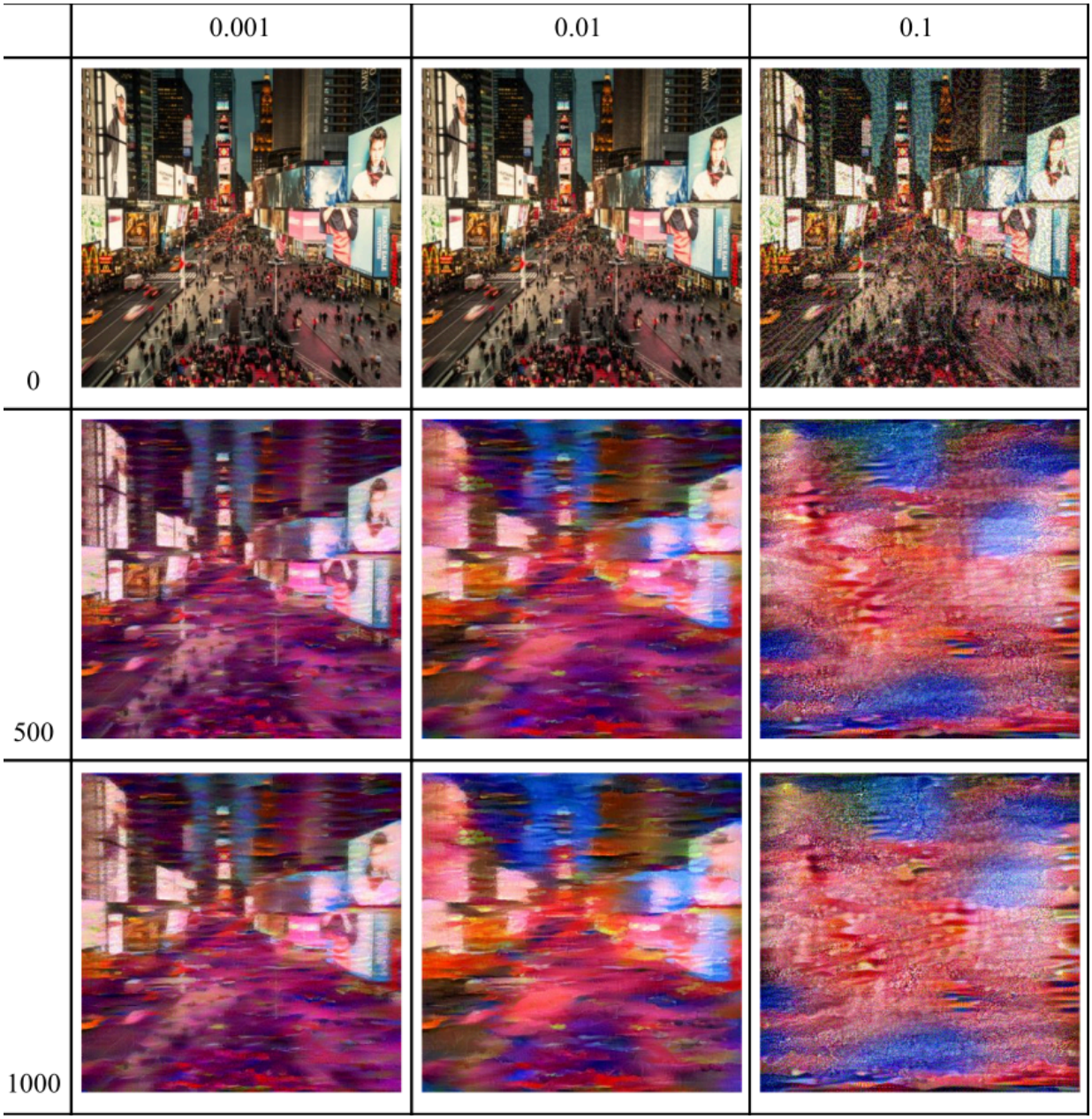
- CNN-based encoder-decoder
- Arbitrary stylization with fixed weights
- Designed for fast, deployable inference

Results

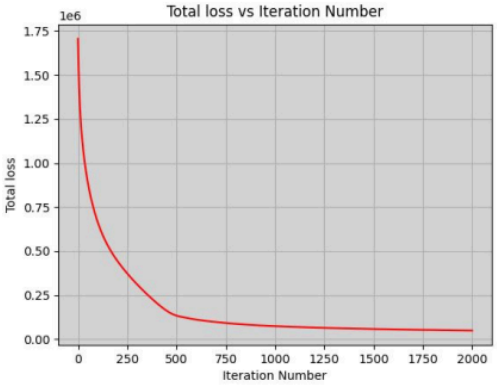
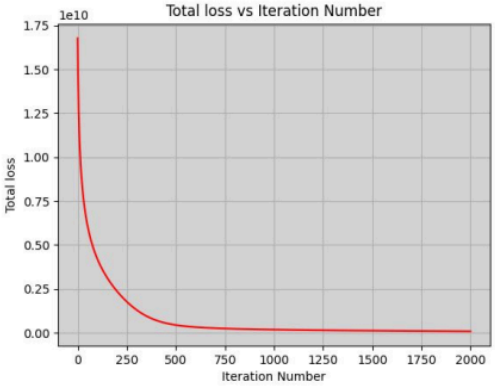
Test Case	Method	Time (s)	Stylization Quality	Editable Parameters
Notebook (VGG19)	PyTorch	~10 mins (1000 iter)	High, with control	Yes (α, β, lr, loss type)

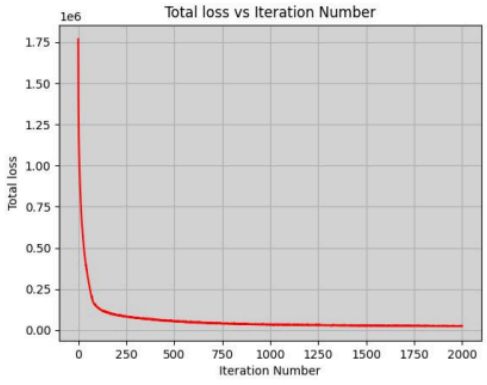
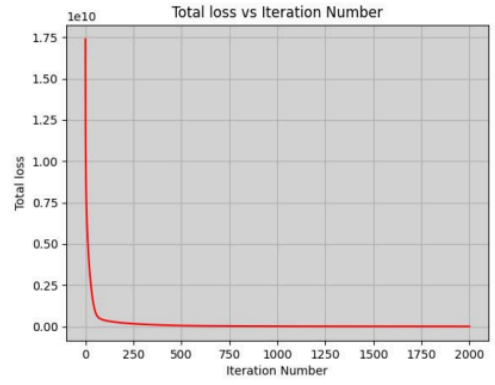
Web App	TF Hub + Streamlit	~3–5s	Good, stylized textures preserved	No (fixed pretrained model)

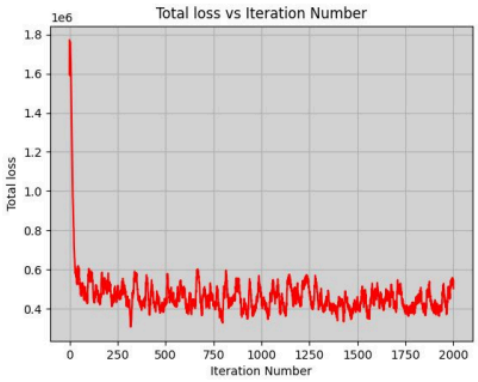
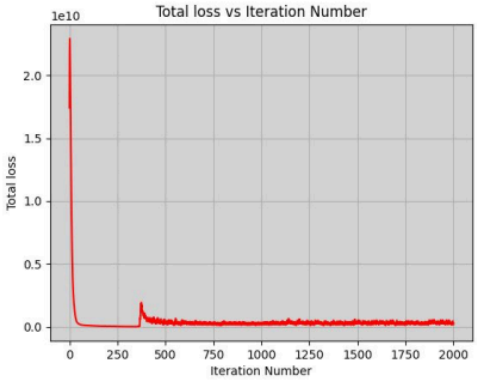
- **Learning Rate Tuning**: Optimal around 0.001–0.01 for stability.

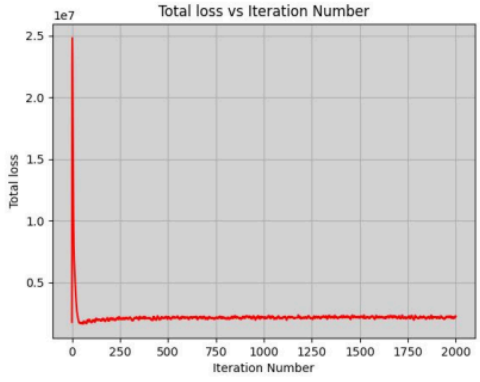
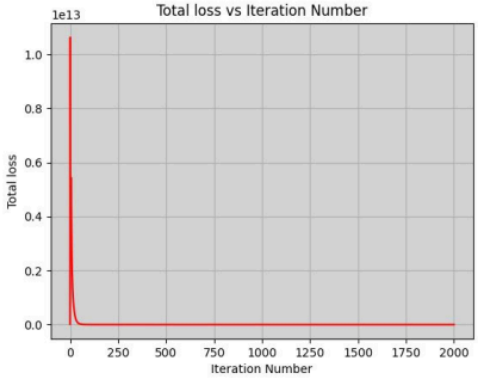


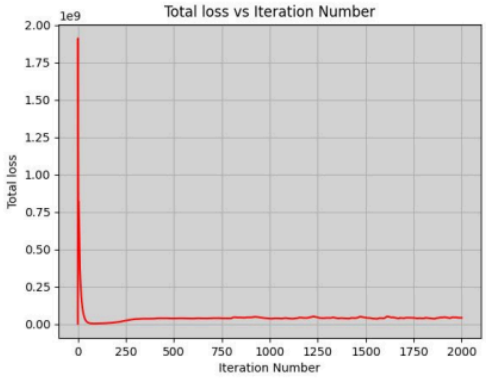
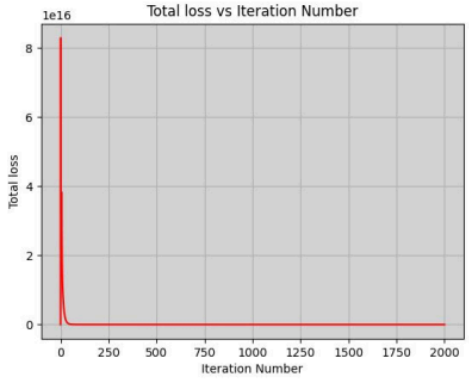
- **Loss Functions**: RMS preserved better texture; MAE was faster but less expressive.

Learning Rate = 0.001	
Mean Absolute Error	Mean Squared Error
	

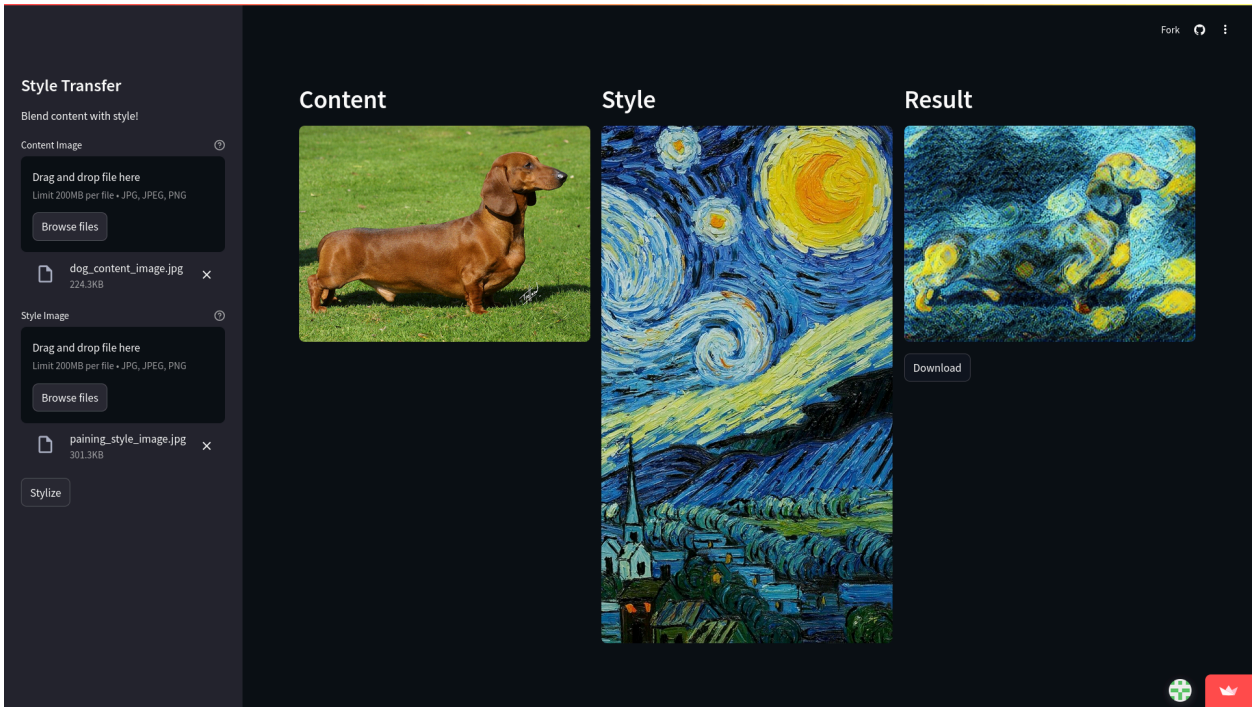
Learning Rate = 0.01	
Mean Absolute Error	Mean Squared Error
	

Learning Rate = 0.1	
Mean Absolute Error	Mean Squared Error
	

Learning Rate = 1	
Mean Absolute Error	Mean Squared Error
	

<table>
<tr><th colspan="2">Learning Rate = 10</th></tr>
<tr><th>Mean Absolute Error</th><th>Mean Squared Error</th></tr>
<tr><td> 
</td><td> 
</td></tr>
</table>

- **Alpha-Beta Trade-offs**: Higher beta added more texture, higher alpha retained structure.
- **Web App**: Provided fast, accessible stylization for anyone via browser.



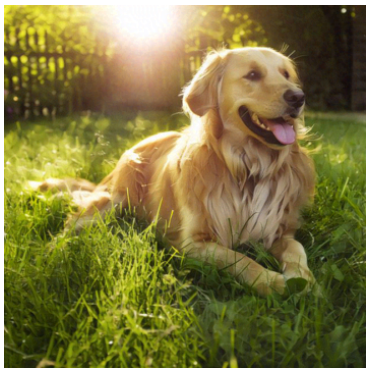
- **Differences in generated outputs:**

  Different kinds of styles are better displayed on the content images through the two different implementations. Specifically, abstract styles are better handled by our VGG19

implementation. In contrast, our arbitrary-image-stylizaiton-v1 implementation will generate better results with painting as style images.

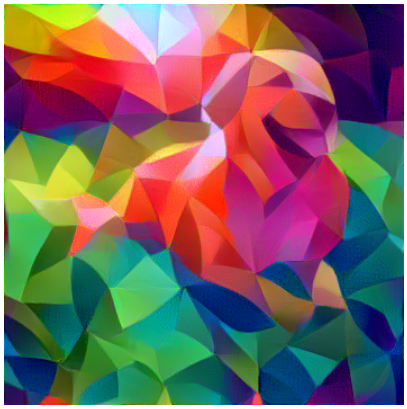
Example

Content:

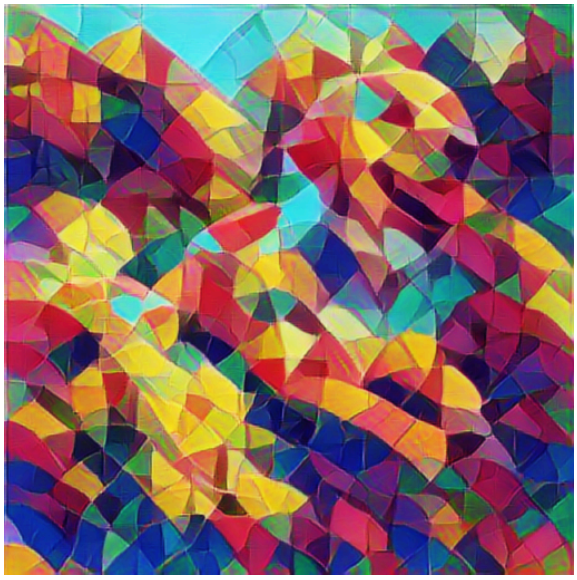


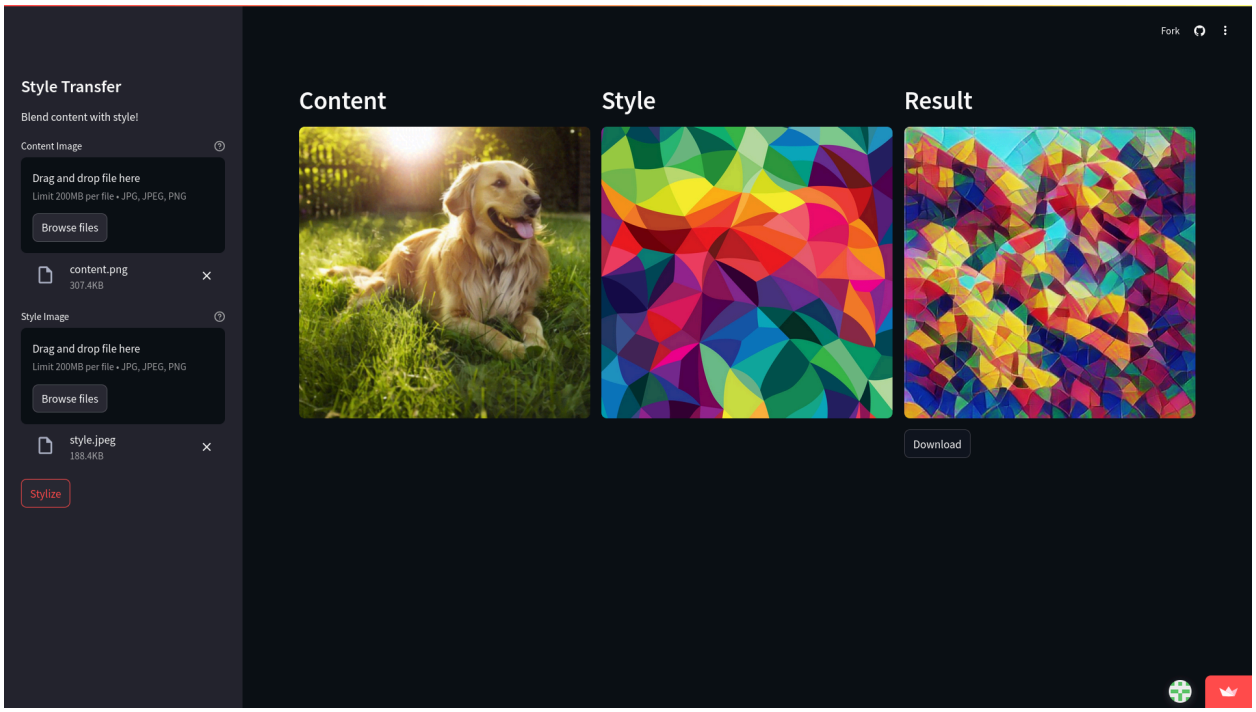
Style:

Generation by VGG19 implementation:



Generated by arbitrary-image-stylizaiton-v1:





Extended Scope

As a final note, an interesting variation of the experiments replaced learning rate and loss function adjustments with alpha and beta tuning. Fixing the optimizer and loss type, this approach manipulates the ratio between content and style, offering more direct control over artistic effect and structure retention.

Conclusion

This study demonstrated two complementary approaches to neural style transfer:

1. A **customizable and experimental backend** for deep learning researchers using VGG19 and PyTorch.

2. A **real-time deployable frontend** for general users via TensorFlow Hub and Streamlit.

Together, they provided a full-stack solution from model training to user interaction. This not only improved the technical understanding of NST but also democratized its use through web deployment.

Future Scope

- **Real-time Style Transfer on Mobile** using TensorFlow Lite or ONNX.
- **Video Stylization** with frame coherence via optical flow.
- **Interactive Controls** in the Streamlit app to adjust style strength.
- **Multiple Style Blending** with sliders to mix more than one style.
- **GAN-Based Style Transfer** for richer stylizations and abstract art generation.

References

Neural Style Transfer: A Review : https://arxiv.org/abs/1705.04058v7

Dynamic Neural Style Transfer using VGG19 https://arxiv.org/abs/2501.09420