COMP473: PATTERN RECOGNITION

Artistic Style Transfer Using Convolutional Neural Networks: A Study and Implementation

GitHub: https://github.com/bhavya6701/comp473-project

Shibin Koshy [40295019] Ruturajsinh Vihol (Rue) [40154693] Bhavya Manjibhai Ruparelia [40164863]

***Abstract*: This report examines the implementation of artistic style transfer, inspired by the seminal work of Gatys et al. (2016), which combines the semantic content of one image with the artistic style of another using Convolutional Neural Networks (CNNs).**[2] **Leveraging the feature extraction capabilities of pre-trained VGG-16 and VGG-19 networks, this implementation demonstrates the optimization process to balance content preservation and style transformation through a weighted loss function.**
The report evaluates the impact of model architecture, content and style layer selection, and hyperparameter tuning on the quality of stylized outputs. Comparative analyses highlight the effectiveness of VGG-19 in capturing intricate style details and preserving structural integrity. The findings validate the potential of neural style transfer in creating visually compelling images, with applications ranging from digital artwork to creative media production.

1. Introduction

Transferring the style of one image onto another can be viewed as a texture transfer problem, where the objective is to synthesize the texture of a source image while maintaining the semantic content of the target image.

Artistic style transfer is a remarkable innovation in computer vision that combines the content of one image with the stylistic elements of another, producing visually compelling outputs. Introduced by Gatys et al. (2016), neural style transfer leverages Convolutional Neural Networks (CNNs) to independently represent and recombine the content and style features of images. [2] This approach marked a significant departure from traditional methods, which relied on statistical techniques or handcrafted features and struggled to achieve the desired balance between content fidelity and stylistic expression.

At the heart of neural style transfer lies the use of pre-trained networks, such as VGG-16 and VGG-19, to extract feature representations. Content features are derived from deeper layers of the network, capturing semantic information, while style is represented using Gram matrices computed from multiple convolutional layers. By optimizing a weighted combination of content and style losses, the algorithm generates a new image that reflects the artistic style of one input while preserving the structural content of another.

This report delves into the implementation of neural style transfer using PyTorch, focusing on the feature extraction capabilities of VGG-16 and VGG-19. It examines critical components of the algorithm, including:

- **Optimization Techniques**: The use of gradient descent with Adam and L-BFGS optimizers to minimize the total loss function. [3]
- **Hyperparameter Selection**: The impact of weighting factors number of iterations, and layer selection on the quality of the stylized output.
- **Evaluation Metrics**: The role of loss functions in achieving a balance between content preservation and stylistic transformation.

Through systematic experimentation, this report evaluates how architectural choices, hyperparameter tuning, and optimization strategies influence the final stylized image. It validates the effectiveness of neural style transfer and highlights its potential for creative applications in digital art, media production, and personalized content generation.

2. Outline

Despite its conceptual elegance, neural style transfer presents several challenges that must be addressed for consistent and high-quality results:

2.1. Content-Style Balance

Achieving an optimal trade-off between preserving the semantic content of the original image and accurately reproducing the desired artistic style. This project uses layer-specific weighting (e.g., layers from VGG-16 and

VGG-19) and hyperparameter tuning to balance content and style fidelity effectively.

2.2. Impact of Model Architecture

Architectural differences between VGG-16, VGG-19, and ResNet-18 are explored. Each model's feature extraction capabilities are analyzed to understand how they affect the stylized output's quality and detail. Pretrained models from Torchvision are utilized, with VGG-16 and VGG-19 serving as the primary architectures for comparison.

2.3. Hyperparameter Optimization

This project employs a set of predefined hyperparameters, such as style and content weights (alpha and beta), learning rates, and iteration counts, which were fine-tuned through experimentation. These parameters are crucial for producing visually appealing results and are explicitly defined in the configuration file.

2.4. Optimization Challenges

Optimization strategies such as Adam and L-BFGS are implemented and compared to determine their strengths and weaknesses in converging to high-quality solutions. The project tracks the total loss over iterations to evaluate convergence behavior.

2.5. Implementation Details

- **External Resources**: Pre-trained models (VGG-16, VGG-19, ResNet-18) were sourced from Torchvision. [6] Image processing utilities from Pillow and PyTorch's tensor operations were integral to feature extraction and loss computation.
- **Algorithmic Approach**: The core algorithm involves extracting content and style representations using convolutional layers, computing Gram matrices for style representation, and optimizing a target image to minimize content and style losses.
- **Outputs**: Generated stylized images and corresponding loss graphs highlight the project's iterative process and outcomes.

This project addresses these challenges by implementing and analyzing the neural style transfer algorithm, providing insights into the factors influencing its success.

3. Why This Approach?

The chosen approach combines state-of-the-art pre-trained convolutional networks and an iterative optimization algorithm to tackle the challenges of neural style transfer. Here's a breakdown of the decisions made and their justification:

3.1. Algorithm and Implementation

The project implements neural style transfer using VGG-based feature extraction, leveraging the models' hierarchical representation of features to capture content and style effectively.

Layer configurations were explicitly selected to focus on the balance between content preservation and stylistic accuracy, as defined in the configuration JSON file.
Optimization algorithms (Adam and L-BFGS) were tested to find a robust strategy for minimizing combined content and style losses.

3.2. What Was Learned

The selection of layers and their weights significantly influences the stylized image's quality. Higher layers focus on more abstract representations, which are critical for content preservation, while lower layers emphasize stylistic details.

Hyperparameter tuning, including balancing style and content weights, plays a crucial role in achieving visually pleasing results.
Adam is faster for iterative updates, while L-BFGS achieves more stable results in fewer iterations.

3.3. Algorithm Capabilities

The implementation successfully generates high-quality stylized images, offering a flexible framework to experiment with various content-style combinations, hyperparameters, and models.

3.4. Implementation and Approach Assessment

The strategy worked effectively, producing visually compelling results. The modular design allowed for seamless switching between architectures (e.g., VGG-16, VGG-19, and ResNet-18).

External resources like pre-trained weights and tensor utilities streamlined development and ensured reproducibility.

3.5. In Hindsight

- **Alternative Models**: Exploring transformer-based models like CLIP or ViT could have provided more context-aware representations, enhancing stylization for complex scenes.

- **Optimization Techniques**: Adopting learning rate schedulers or hybrid optimization strategies might have improved convergence speed and quality.
- **Augmented Features**: Using additional image augmentation or post-processing techniques could further refine results.
- This approach demonstrates a well-rounded exploration of neural style transfer while offering practical insights into its implementation and performance.

4. Methodology

Our implementation closely follows the methodology proposed by Gatys et al., employing the VGG-16 model for feature extraction and combining content and style features through optimization. Key steps include data preparation, feature extraction, loss computation, and optimization.

4.1. Data Preparation

Content and style images are prepared using PyTorch utilities to ensure compatibility with the pre-trained networks.

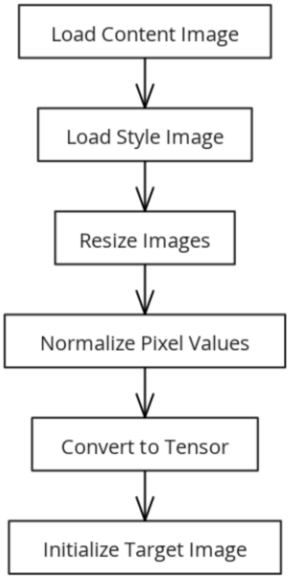

Figure 1: Image Preparation Flowchart

The images are first loaded and resized to 512 x 512 pixels, maintaining a consistent input size for the model. During preprocessing, normalization is applied to align the pixel values with the expected range of the VGG network, ensuring optimal performance during feature extraction. Additionally, the images are converted into tensors, making them suitable for operations within the PyTorch framework.

4.2. Algorithm Design

The algorithm for neural style transfer leverages pre-trained Convolutional Neural Networks (CNNs) for feature extraction, a dual-objective loss function for balancing content and style, and optimization techniques to synthesize a stylized image. Below, we briefly outline the core design decisions and their rationale:

- **Pre-trained Models for Feature Extraction**: The choice of VGG-16, VGG-19 and ResNet-18 models is rooted in their hierarchical feature representation capabilities.[4] Shallow layers capture textures and edges, while deeper layers encode high-level semantics, enabling the effective separation and manipulation of content and style. [6]
- **Content-Style Balance via Loss Function**: The algorithm employs a weighted combination of content and style losses, tuned through hyperparameters (alpha and beta). [2] Content loss ensures structural integrity by comparing the generated image's high-level feature maps to those of the content image. Style loss, based on Gram matrix comparisons, enforces adherence to the artistic style across multiple layers.
- **Optimization Strategy**: Two optimization techniques—Adam and L-BFGS—were tested. Adam offers flexibility and speed in iterative updates, while L-BFGS converges more stably for this problem. The stylized image is generated by minimizing the combined loss over several iterations.
- **Modular and Adaptive Design**: The modular structure allows switching between models and varying hyperparameters. This adaptability ensures that the approach works across diverse content and style combinations, as demonstrated by consistent outputs during testing.

This design effectively synthesizes content and style, balancing algorithmic precision with practical flexibility.

4.2.1. Feature Extraction

Feature extraction is a crucial step in artistic style transfer, where the content and style representations of images are derived from the intermediate layers of a pre-trained convolutional neural network (CNN). These networks, such as VGG-16, VGG-19, and ResNet18, are trained on large-scale datasets like ImageNet, allowing them to learn hierarchical feature representations. In this project, feature extraction leverages the different levels of abstraction encoded in the layers of these networks. Shallow layers capture edges and textures, while deeper

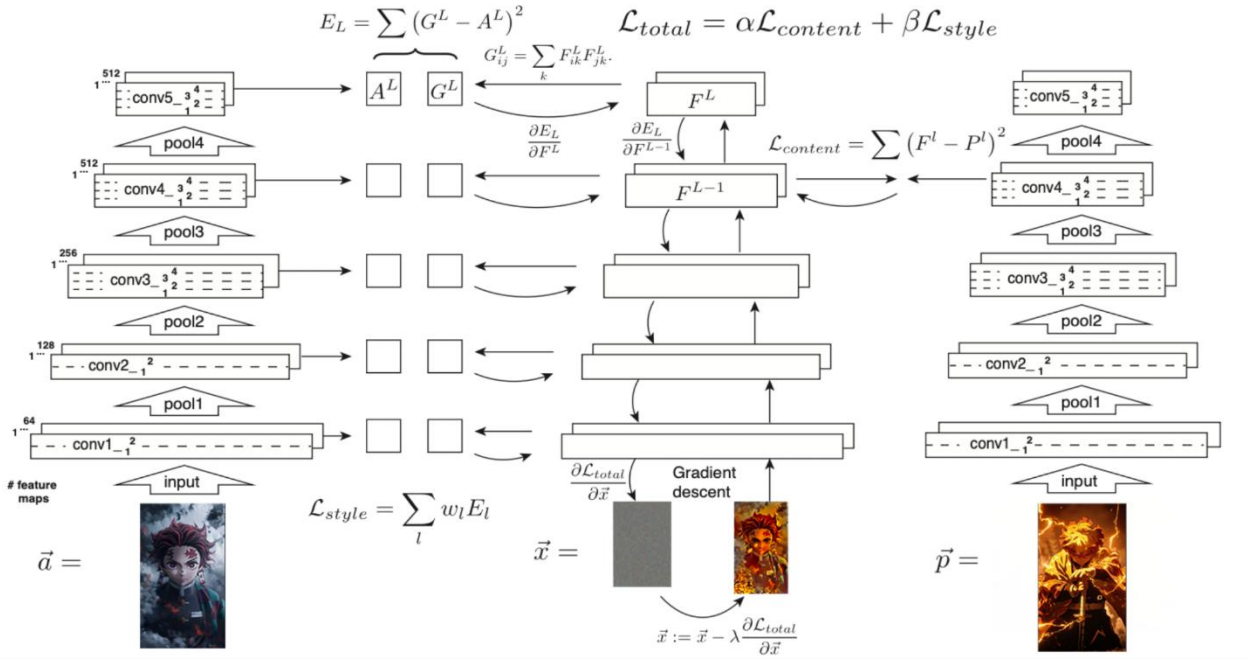

Figure 2: Neural Style Transfer: Architecture and Loss Function Workflow

layers represent complex shapes and high-level semantics.

Feature extraction enables the separation of content and style, which are later recombined to generate a new stylized image. Below, we provide detailed insights into how content and style representations are formulated.

Content Representation: Content representation focuses on the structural and semantic information of an image. High-level features from deeper layers of the CNN are utilized to encode the content. These layers, such as conv4_2 in VGG-19, capture spatial arrangements and relationships among objects within the image. [2] The feature map of the content image serves as the reference for ensuring the structural integrity of the output during optimization. By comparing the feature map of the generated image with that of the content image, the content loss is calculated, which guides the stylization process to preserve the original scene or object structure.

The importance of content representation lies in its ability to retain recognizable elements from the content image, even as the style features are blended. For example, in transferring the style of a painting onto a photograph, the photograph's core layout and primary objects remain intact, ensuring that the output is a meaningful combination of the two inputs.

Style Representation: Style representation captures the artistic aspects of an image, such as texture, color, and patterns. Unlike content features, which depend on spatial arrangements, style features rely on spatially invariant statistical relationships. These relationships are computed using the Gram matrix, a mathematical construct that measures the correlations between different feature maps of the CNN.

By extracting style features from multiple layers (e.g., conv1_1, conv2_1, conv3_1, etc.), the algorithm models both fine-grained textures and broader artistic patterns. The Gram matrix aggregates these features into a representation that embodies the image's artistic essence. During optimization, the style loss quantifies the discrepancy between the Gram matrix of the generated image and that of the style image, ensuring the output adheres to the desired aesthetic.

The multi-layered approach to style representation enables the algorithm to capture both low-level details, such as brush strokes, and high-level stylistic elements, like color palettes. This hierarchical modeling is key to creating outputs that are both visually appealing and faithful to the input style.

4.2.2. Loss Function

The loss function serves as the guiding metric for the optimization process in neural style transfer. It quantifies the differences between the generated image and the target content and style, ensuring that the resulting output maintains the semantic structure of the content image while incorporating the artistic patterns of the style image. The total loss function combines the content and style losses, weighted by user-defined

parameters α and β which controls the relative importance of each component.

Content Loss: This measures the squared error between the feature map of the generated image and the feature map of the content image, ensuring that the generated image retains structural integrity.

$$L_{content} = \frac{1}{2}\Sigma_{i,j}\left(F_{i,j}^{l} - P_{i,j}^{l}\right)^2$$

where F is the feature representation of the generated image, and P is that of the content image at the layer **l**, indexed by **i** and **j**.

Style Loss: The style loss measures the stylistic differences between the generated image and the style image by comparing their Gram matrices. Gram matrices capture the correlations between different feature maps of the convolutional layers, encoding the texture and patterns present in the style image. The style loss is computed as:

$$L_{style} = \frac{1}{4N^2M^2}\Sigma_{i,j}\left(G_{i,j}^{l} - A_{i,j}^{l}\right)^2$$

Here, G and A are the Gram matrices for the generated and style images between filters **i** and **j** in layer **l**, respectively.

Total Loss: The total loss combines the content loss and style loss to achieve a balance between preserving the content's structure and transferring the style's texture and patterns.

$$L_{total} = \alpha \cdot L_{content} + \beta \cdot L_{style}$$

Here, α: is the content weight that controls the importance of content preservation and β: is the style weight that emphasizes the importance of stylistic transformation.

By adjusting these weights, users can fine-tune the trade-off between maintaining the content image's structure and incorporating the style image's artistic features. For instance, a higher α relative to β prioritizes content preservation, while a higher result in a more stylized output.

4.2.3. Optimizers

In the context of artistic style transfer, the choice of optimizer significantly impacts both the convergence process and the visual qualities of the resulting image. The project utilized two optimizers, **L-BFGS** and **Adam**, each with distinct characteristics and initialization strategies.

L-BFGS is a quasi-Newton optimization algorithm well-suited for style transfer due to its stability and fast convergence. This optimizer typically begins with a random noise image, ensuring that the generated image blends feature from both the content and style inputs during the early iterations. This blending often results in intermediate outputs that exhibit an artistic fusion of both images, though they may initially appear less coherent compared to the content image. The results, however, tend to be more polished as L-BFGS efficiently minimizes the total loss.

In contrast, **Adam** is an adaptive gradient-based optimizer that allows the optimization process to start directly from the content image. This initialization strategy ensures that the generated image retains a strong resemblance to the content image throughout the process. Consequently, Adam's intermediate results often appear more visually appealing to the observer, especially during early iterations.

The noticeable difference lies in the progression and nature of intermediate results: while Adam emphasizes content preservation early on, L-BFGS facilitates a gradual stylistic fusion that leads to a more balanced final output.

4.3. User Manual

4.3.1. Choice of Tools

- **Hardware**: Minimum 8GB RAM. For optimal performance, an NVIDIA GPU (e.g., RTX 3060 or higher) is recommended to speed up computations.
- **Software**: Python 3.10, PyTorch 2.0, and supporting libraries.
- **Execution Time**: Approximately 30 seconds per image pair at 400x400 resolution for 500 iterations on a GPU.

This project utilizes tools like PyTorch for neural network implementation and pre-trained models, Jupyter Notebook for experimentation and visualization, and Visual Studio Code for code development. [5]

4.3.2. Compilation and Execution

The project's GitHub repository provides step-by-step instructions for installing dependencies, setting up the environment, and running the code.

- README.md
- requirements.txt

Please refer to these links for detailed guidance.

4.4. VGG and ResNet Models

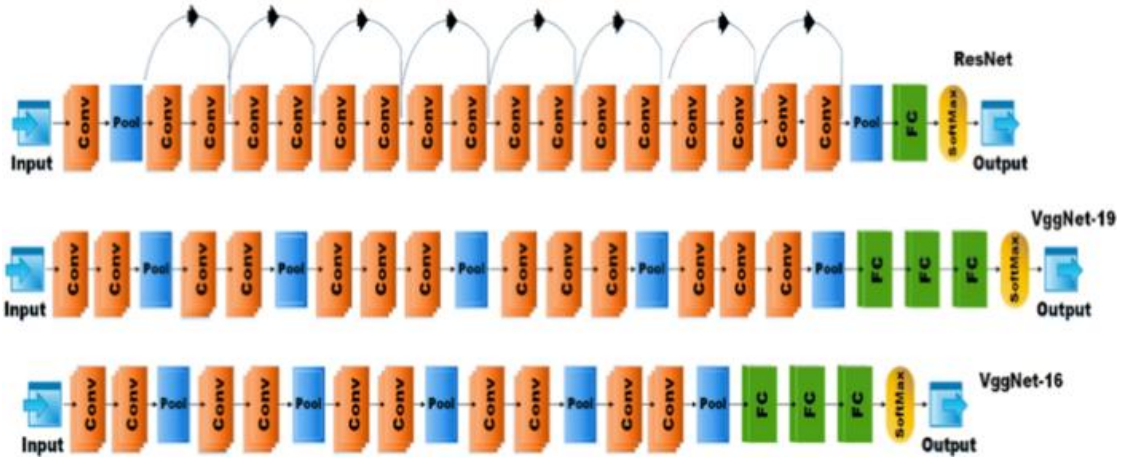

Figure 3: Network architecture of ResNet-18, VggNet-19, and VggNet-16

The choice between VGG-16, VGG-19 and ResNet-18 was driven by the need to balance computational efficiency with the quality of the generated images.

VGG-16: Preferred for this implementation due to its lower computational requirements, making it suitable for systems with limited resources. Its architecture effectively balances content preservation and style transfer, providing high-quality results without excessive resource consumption.

VGG-19: While VGG-19 offers finer detail extraction due to its deeper architecture, its additional layers introduce higher computational demands with diminishing returns in visual quality. Resource constraints prevented its use in this implementation, prioritizing a more efficient approach with VGG-16.

ResNet-18: ResNet-18, with 18 layers and residual connections, ensures efficient training and hierarchical feature extraction. Style features are drawn from layer1 to layer4, with content from layer3. While ResNet-18 is configured, it wasn't actively used in this project, as its behavior differs from the well-established VGG-based approaches for style transfer.

VGG-16 offers the best trade-off between quality and computational efficiency, making it the preferred choice. VGG-19 delivers finer details but incurs higher costs, while ResNet-18 remains a modern alternative with limited adoption for this task. VGG-16 was selected for its practicality and balanced performance.

4.5. Hyperparameter Tuning

The hyperparameters α and β control the relative importance of content and style in the generated image:

α: Assigns equal weight to the content loss, emphasizing structural preservation from the content image.

β: Heavily prioritizes style transfer by assigning a significantly larger weight to the style loss.

This balance ensures the stylized image effectively integrates the artistic patterns of the style image while retaining the semantic structure of the content image.

Figure 4: Low α/β Ratio (1e-8)

Figure 5: High α/β Ratio (1e-4)

4.5.1. Learning Rate

The **learning rate**, set to 0.1, plays an essential role in controlling the step size during optimization. A moderate learning rate ensures stable convergence without overshooting the optimal solution. This value is particularly effective when used with the Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-

BFGS) optimizer, which refines the generated image iteratively.

4.5.2. Iterations

The number of **iterations** is set to 250, balancing computational efficiency with the refinement of the stylized image. This ensures that the target image undergoes sufficient optimization to integrate style patterns without excessive computational overhead.

4.5.3. Style Layer Weights

Layer-specific weights are assigned to capture multi-scale stylistic features effectively. Layers such as conv1_1 through conv5_1 are weighted to balance fine details and broader textures. Earlier layers like conv1_1 and conv2_1 are assigned higher weights (e.g., 1.0 and 0.8, respectively) to emphasize fine textures, while deeper layers such as conv4_1 and conv5_1 are weighted lower (e.g., 0.3 and 0.2) to focus on global patterns. This hierarchical weighting scheme ensures that textures across multiple scales contribute proportionately to the final output.

By fine-tuning these hyperparameters—content and style weights, learning rate, iterations, and style layer contributions—the implementation achieves a stylized output that balances artistic expression with structural coherence. These carefully calibrated values make the algorithm both robust and adaptable to a variety of artistic styles and computational settings.

4.6. Content Layer Comparison

The choice of content layer significantly impacts the results in neural style transfer, as different layers of a convolutional neural network capture varying levels of abstraction. To evaluate this effect, we experimented with multiple content layers, such as conv1_1, conv2_1, conv3_1, conv4_1 and conv5_1 using VGG-19, when calculating content loss, and compared the resulting stylized images.

Shallow layers like conv1_1 primarily capture low-level features such as edges and textures. Using these layers for content loss resulted in outputs that retained finer spatial details but appeared overly influenced by the style image, often leading to excessive texturing. On the other hand, deeper layers like conv4_2 encode high-level semantic information, focusing on the overall structure of the content image while discarding less relevant details.

4.7. Variation in Results with L-BFGS

The L-BFGS optimizer is deterministic for a given starting point, but in neural style transfer, results vary each time due to the random initialization of the target image with noise. This randomness introduces unique starting conditions for the optimization process, leading the algorithm to converge to different local minima in the non-convex loss space.

Despite variations, outputs consistently balance content structure and style patterns. The differences are reflected in subtle changes to texture, color balance, and stylistic pattern placement, showcasing the flexibility and artistic diversity of the process. This randomness can be an advantage in creative applications by generating a range of outputs from the same inputs.

5. Description of Program Execution and Results

5.1. Testing Methodology

Validation: To evaluate the program's robustness, diverse content and style images were used as inputs. These included vibrant abstract artworks for style images and noisy grayscale photographs for content images. The variety ensured that the program could adapt to different visual properties.

The impact of content and style weights (α and β) was extensively tested to explore the balance between content preservation and stylistic transformation. Lower α/β ratios led to outputs dominated by the style image, replicating textures and patterns effectively but with reduced content fidelity. Conversely, higher α/β ratios resulted in images that retained more structural details of the content at the expense of stylistic richness.

Edge Cases: The algorithm was tested on challenging scenarios to evaluate its adaptability:

- **Grayscale Content with Vibrant Styles**: The program successfully applied colorful, textured styles to grayscale content images, showcasing its ability to transfer stylistic elements without relying on color similarity.
- **Extreme α/β Ratios**: Testing extreme values revealed the program's boundaries. Extremely low α/β ratios resulted in excessive stylization with diminished content structure, while extremely high ratios produced outputs heavily dominated by content, with minimal stylistic integration.

5.2. Program Verification

The program was tested extensively to ensure correctness and consistency:

Feature Map Validation: The extracted content and style features were visually inspected at key layers to confirm proper feature extraction.

Loss Function Behavior: The content and style losses were monitored during optimization to ensure convergence. Plots of loss values over iterations validated the algorithm's stability and effectiveness as shown in the plot below for the total loss after each iteration while running the algorithm on VGG-19.

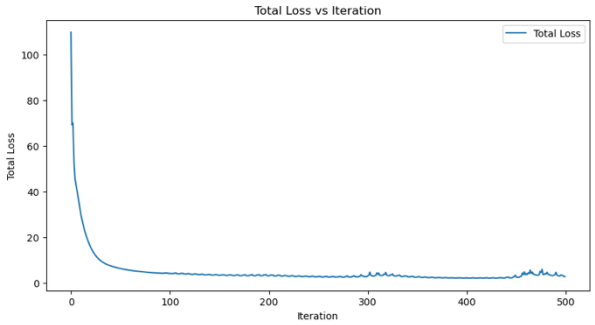

Figure 6: Total loss vs iteration graph

Comparison Across Models: Outputs from different architectures (e.g., VGG-16, VGG-19, and ResNet-18) were compared to identify performance variations, with VGG-19 producing the most intricate stylizations.

Target Image Progression: Intermediate results were saved at regular intervals during optimization. These snapshots revealed the progressive integration of style into the content image and provided a visual confirmation of loss reduction. The evolving stylization showcased how the algorithm effectively balanced content and style over iterations.

5.3 Program Demonstration

- **Content and Style Inputs**: Input images prepared for stylization (e.g., a photograph as content and a Van Gogh painting as style).
- **Stylized Outputs**: Outputs generated after 500 iterations, demonstrating a balance between content preservation and style replication for different α/β ratios.
- **Loss Convergence**: Graphs showing the reduction of content and style loss over iterations, highlighting the optimization process.
- **Extreme Cases**: Examples of extreme α/β ratios showing highly stylized versus content-dominated outputs.

5.4 Outcome Description

5.4.1. Outcome of the Project

- The program successfully performs neural style transfer, generating high-quality stylized images.
- VGG-19 consistently produced the best results, while ResNet-18 struggled to capture detailed styles.
- The algorithm demonstrated flexibility across diverse content and style combinations, providing robust and visually appealing outputs.

5.4.2. Program Success

- The implementation strategy worked effectively, as evidenced by consistent convergence and high-quality stylized outputs.
- Testing with diverse inputs and edge cases confirmed the program's robustness and adaptability.

6. Results and Analysis

This section evaluates the performance of the style transfer implementation across different architectural choices, hyperparameter settings, content layer selections, and the effect of random initialization with L-BFGS. The results are analyzed for their impact on content preservation and stylistic fidelity.

6.1. Model Comparison

The choice of architecture significantly influenced the quality of the stylized outputs:

VGG-16: Provided consistent results with balanced style and content representation. However, due to fewer layers compared to VGG-19, it sometimes missed intricate stylistic details, particularly in complex styles.

VGG-19: Delivered superior results with enhanced stylistic richness, owing to its deeper architecture. It captured finer textures and patterns from the style image, making it the most effective model for artistic style transfer in this project. The trade-off was higher computational requirements and longer processing times.

ResNet18: While computationally efficient due to residual connections, the outputs were often less stylistically cohesive compared to the VGG models. ResNet18 required additional tuning to achieve comparable results, suggesting that its architecture might be less suited for style transfer tasks.

It is observed that the VGG-16 and VGG-19 models out-perform ResNet-18 in terms of style transfer. Figures 8 and 10 show the input image and the result image from each model.

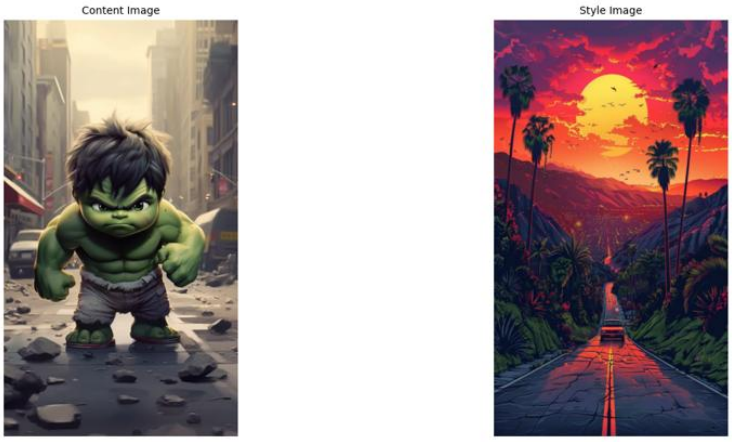

Figure 8: Content and Style Images Used for Neural Style Transfer

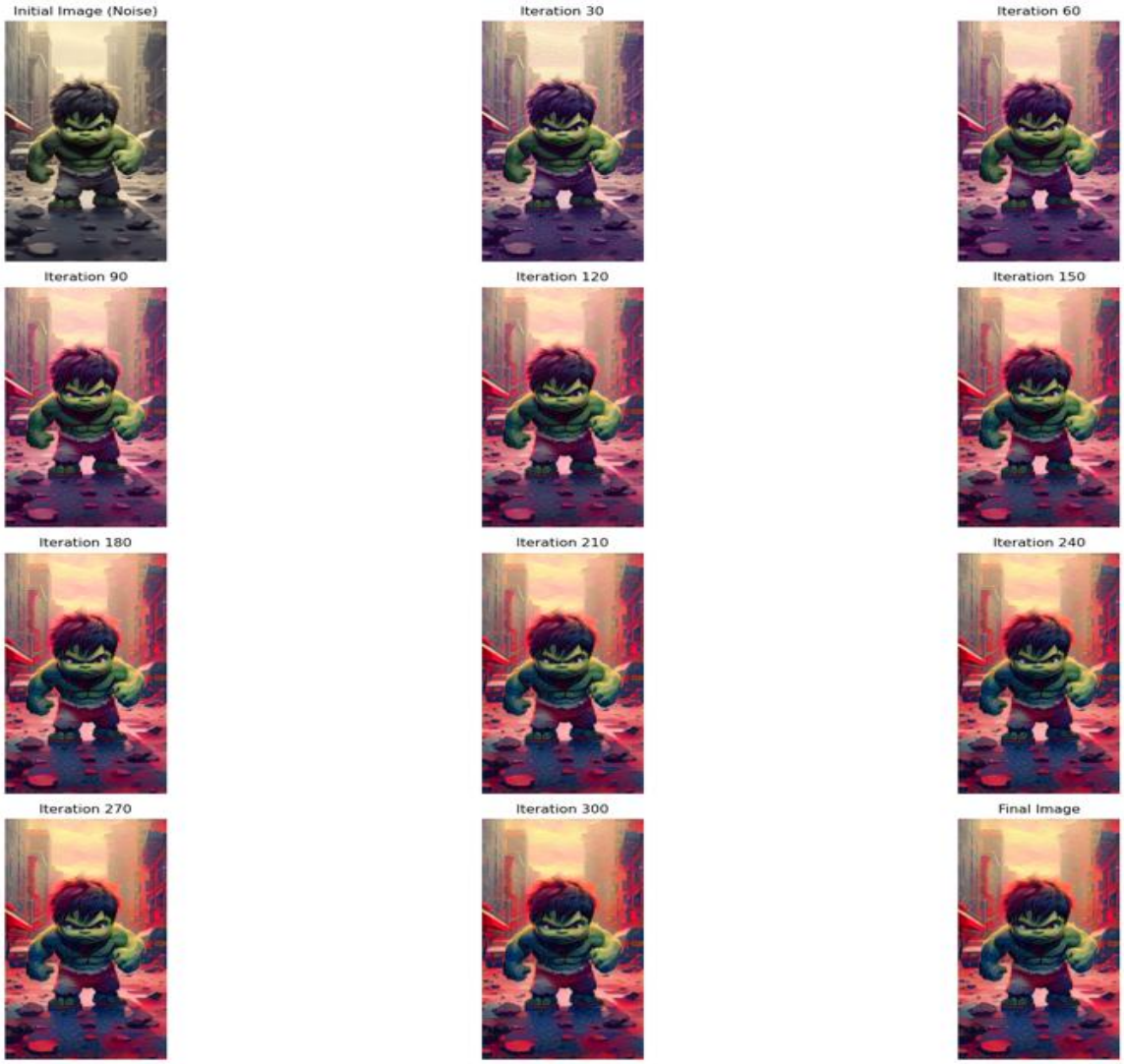

Figure 7: Stylization Progression: Iterative Transformation from Noise to Final Output Over 300 Iterations

Figure 9: Comparison of Final Stylized Images Using VGG-16, VGG-19, and ResNet-18 Architectures

6.2. Hyperparameter Tuning

Hyperparameter adjustments played a critical role in achieving desired results:

Content and Style Weights (α and β): Lower the α/β ratio (α=1, β=1e8), more emphasis on stylistic features at the expense of content, resulting in vibrant outputs dominated by textures and patterns. Conversely, higher α/β ratio (α=1, β=1e2) prioritizes content preservation, yielding outputs closer to the structure of the content image. An intermediate balance (α=1, β=1e5) was optimal for most cases. This is depicted in figure 10, where the results of different α/β ratios are compared.

Iterations: Around 300-1000 iterations were sufficient to generate visually appealing results without overfitting or excessive computation. Fewer iterations often resulted in incomplete style application.

Learning Rate: A stable learning rate between 0.01 to 0.5 using Adam optimizer and 1 for L-BFGS optimizer ensured smooth convergence. Overly high values caused instability, while very low values prolonged optimization.

6.3 Content Layer Comparison

The choice of content layer influenced how the structure of the content image was preserved in the stylized outputs (as shown in the figure 11):

Deeper Layers (e.g., conv4_2): Captured high-level semantic information, ensuring the overall structure and recognizable features of the content image were retained. This layer provided the best results for content fidelity.

Shallower Layers (e.g., conv1_1): Focused on fine-grained spatial details, often resulting in outputs that overemphasized textural features of the style at the expense of content structure. While visually interesting, these results were less suitable for tasks prioritizing content preservation.

Figure 10: Stylized Image Results with Varying α/β Ratios

Figure 11: Results using different content layer for calculating content loss.

Figure 12: Using L-BFGS optimization and starting from noise image, different results can be achieved where each image has a slight style variation.

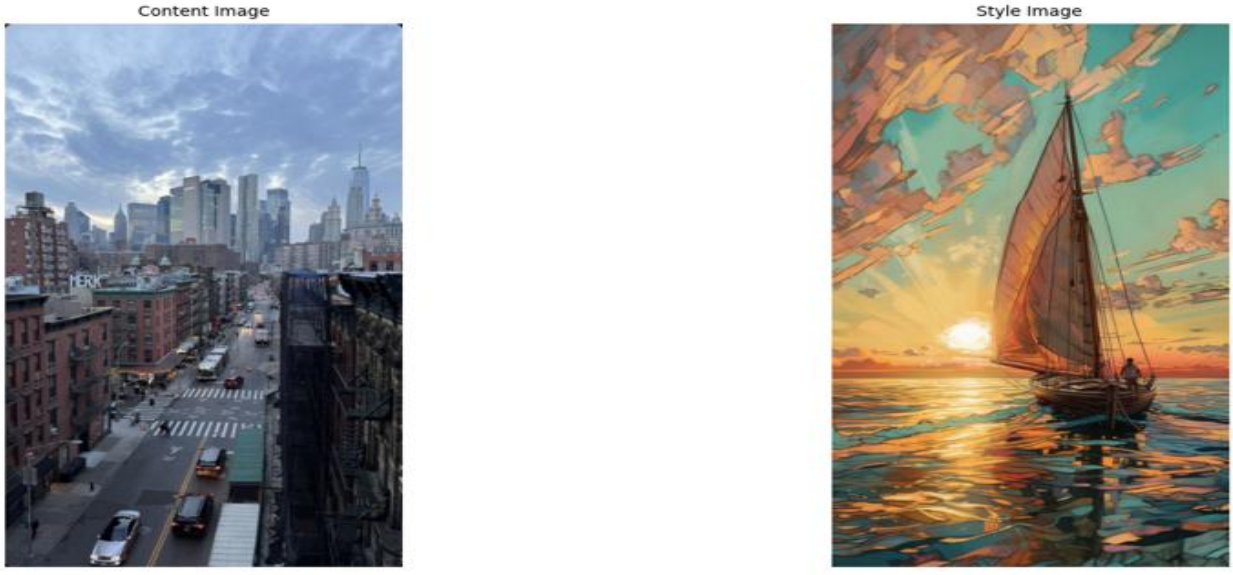

Figure 13: Example - Neural Style Transfer (Content and Style Image)

Figure 14: Neural Style Transfer Example – Resulting images from VGG-16, VGG-19, and ResNet-18 models. Each taking 300 iterations with learning rate of 0.1 and α/β ratio of 1e-7.

6.4. Variation in Results with L-BFGS

The use of L-BFGS with random noise initialization led to variations in the final outputs across different runs.

Random Initialization: Each run started with a different noise image, resulting in unique stylistic outputs due to differences in the optimization trajectory. This variability is attributed to the non-convex nature of the loss function, where different initializations lead to convergence at different local minima. Example shown in figure 12.

Observations: While all outputs effectively balanced content and style, subtle differences were observed in texture placement, color blending, and pattern arrangement. This randomness introduced creative diversity into the results, making the method valuable for artistic applications.

Robustness: Despite variations, the overall quality of the outputs was consistently high, demonstrating the robustness of the algorithm.

6.5. Computational Challenges

The optimization process was computationally intensive, requiring significant GPU resources to achieve acceptable runtimes. Image resolution and the number of optimization iterations also played a critical role in determining the perceptual quality of the results.

6.6. Key Takeaways

- VGG models are highly effective for feature extraction in style transfer tasks, with VGG19 achieving the best results in balancing content and style.
- Adam outperformed L-BFGS in performance, efficiency, and visually coherent outputs though L-BFGS offered stylization fidelity. Adam is 10 or more times faster than L-BFGS with the same number of iterations. Our approach (Adam's approach) is faster to execute and takes a few minutes which the L-BFGS approach from the paper is computationally heavy and takes around 1 hour to get to 300 iterations and 4 hours+ to achieve results.
- The results reaffirm the adaptability of CNNs for combining artistic styles with meaningful content, paving the way for more advanced applications in creative AI.
- This below is an example where we start with the target/content image and combine with style image and the results below are qualitative comparison between, VGG16, VGG19 and Resnet-18 model in which we use the Adam's approach.

7. Insights and Learnings

Understanding CNNs: Higher layers in CNNs capture abstract semantics, such as object shapes and structures, while lower layers focus on fine details, like edges and textures. [2] This hierarchy is key to effective content and style separation.

Challenges in Optimization: Balancing content and style is complex and heavily influenced by hyperparameters such as α/β ratios. Achieving the right trade-off requires careful experimentation and tuning.

Practical Insights: Computational efficiency is crucial for real-world applications, such as deploying neural style transfer on mobile devices or enabling real-time image editing. Techniques like real-time processing or model pruning could significantly improve usability.

8. Related Work

The foundation of this project builds on the seminal work of Gatys et al. (2016), which introduced a neural algorithm to decouple content and style using CNNs. Key advancements inspired by this foundational work include:

- **Real-Time Style Transfer:** Johnson et al. (2016) proposed feed-forward networks for faster processing, eliminating the computational overhead of iterative optimization. [1]
- **Adaptive Instance Normalization (AdaIN):** Huang and Belongie (2017) introduced AdaIN to align feature statistics between content and style images, providing greater flexibility.
- **GAN-Based Approaches:** Techniques like CycleGAN and StyleGAN extended style transfer to unpaired datasets and unseen domains, enabling new creative possibilities. [7]
- **Diffusion Models:** Recent innovations use diffusion models to improve stylization fidelity and flexibility, ensuring better content preservation while allowing diverse artistic transformations.

9. Future Work

- Explore transformer-based architectures like Vision Transformers (ViTs) or diffusion models for better stylization fidelity.
- Implement real-time optimization techniques, such as feed-forward networks, to enhance computational efficiency.
- Investigate style transfer with unpaired datasets using advanced GAN techniques to broaden applicability.

10. Conclusion

This project reinforced our understanding of deep neural networks and their capacity to model complex visual phenomena.

- **Content and Style Separation**: CNNs can effectively disentangle content and style, enabling flexible artistic manipulation of images.
- **Role of Gram Matrices**: Capturing pairwise correlations across feature maps is a robust method for representing texture and style.
- **Optimization and Trade-offs**: Balancing content and style requires careful tuning of loss weights and layer selection.

Future work could explore techniques to accelerate optimization, enhance photorealism, or apply style transfer to video.

11. References

1. Johnson, J. A., Alahi, A., & Fei-Fei, L. (2016). Perceptual Losses for Real-Time Style Transfer and Super-Resolution. *In Proceedings of the European Conference on Computer Vision (ECCV)*, 694–711. https://doi.org/10.1007/978-3-319-46475-6_43
2. Gatys, L. A., Ecker, A. S., & Bethge, M. (2016). Image Style Transfer Using Convolutional Neural Networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
3. Kingma, D. P., & Ba, J. (2015). Adam: A Method for Stochastic Optimization. *In International Conference on Learning Representations (ICLR)*, arXiv:1412.6980.
4. Simonyan, K., & Zisserman, A. (2016). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv*, arXiv:1409.1556.
5. Paszke, A., Gross, S., Massa, F., et al. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library.
6. TorchVision. (2023). Models. [Online]. Available: https://pytorch.org/vision/stable/models.html
7. ResearchGate. (n.d.). Basic Network Architecture of AlexNet, ResNet-18, VGGNet-19, and VGGNet-16Fastai. [Online]. Available: https://www.researchgate.net/figure/Basic-Network-architecture-of-AlexNet-ResNet-18-VggNet-19-and-VggNet-16Fastai_fig3_363798567