Neural Style Transfer: Analysis and Implementation

Name: Priyanshu Ranka (NUID: 002305396)
Professor: Prof. Bruce Maxwell
Subject: Pattern Recognition and Computer Vision
Semester: Spring 2025

1. Introduction and Problem Statement

Neural Style Transfer (NST) represents a class of algorithms that manipulate digital images to adopt the visual style of another image while preserving original content. This technique sits at the intersection of *computer vision* and *deep learning*, leveraging *convolutional neural networks* to separate and recombine *content and style* representations.

The fundamental challenge addressed in this project is the algorithmic separation of content and stylistic elements within images—a task that requires sophisticated understanding of how neural networks encode visual information at different levels of abstraction. As established by *Gatys et al. (2016)*, this separation enables the creation of novel artistic imagery through mathematical optimization, essentially solving a texture transfer problem within deep neural network feature spaces.

2. Research Objectives

This project pursues the following objectives:

1. Implementation of a feature-based neural style transfer algorithm using the PyTorch framework and VGG19 architecture
2. Development of a comprehensive experimental framework for systematic evaluation of hyperparameters
3. Quantitative and qualitative analysis of how key parameters (noise ratio, learning rate) affect stylization quality
4. Evaluation of optimization trajectory and loss convergence patterns across parameter combinations
5. Formulation of practical guidelines for hyperparameter selection based on empirical evidence

3. Theoretical Framework and Literature Review

3.1 Neural Style Transfer Foundations

The theoretical foundation of this work rests on the seminal paper by *Gatys et al. (2016)*, *"A Neural Algorithm of Artistic Style,"* which demonstrated that convolutional neural networks (CNNs) implicitly learn to separate content and style. This separation occurs naturally through the network's hierarchical structure:

- Lower layers capture local patterns and textures (style information)
- Higher layers capture semantic content and spatial structure (content information)

As noted by Jing et al. (2019) in their comprehensive survey "Neural Style Transfer: A Review," this separation enables the independent manipulation of content and style through optimization in feature space:

"The key insight was that the correlation between different filter responses at each layer of the network captures the style of an input image, while the content is preserved in the exact filter responses at higher layers."

3.2 Feature Representation and Loss Functions

Following *Gatys'* approach, our implementation utilizes feature representations from the *VGG19* network pre-trained on ImageNet. The content representation is captured from *layer relu4_2*, while style information is extracted from multiple layers *(relu1_1, relu2_1, relu3_1, relu4_1, relu5_1).*

Johnson et al. (2016) in *"Perceptual Losses for Real-Time Style Transfer and Super-Resolution"* formalized the content loss as:

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} (F_{ij}^l(\vec{p}) - F_{ij}^l(\vec{x}))^2$$

Where F^l represents the feature map at layer l, and p and x are the content image and the image being optimized, respectively.

The style is represented through Gram matrices of filter responses, as described by Gatys et al. (2016):

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

The style loss is then calculated as:

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2$$

Where A^l is the Gram matrix of the style image at layer l, and N_l and M_l are the dimensions of the feature maps.

3.3 Optimization Approach

This implementation employs the *Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS)* algorithm for optimization, following *Gatys'* original approach. As noted by *Li et al. (2017) in "Demystifying Neural Style Transfer,"* L-BFGS is particularly well-suited for style transfer optimization due to its efficient handling of the high-dimensional parameter space and its ability to incorporate curvature information.

4. Methodology and Implementation

4.1 System Architecture

The implementation follows a modular design with distinct components for:

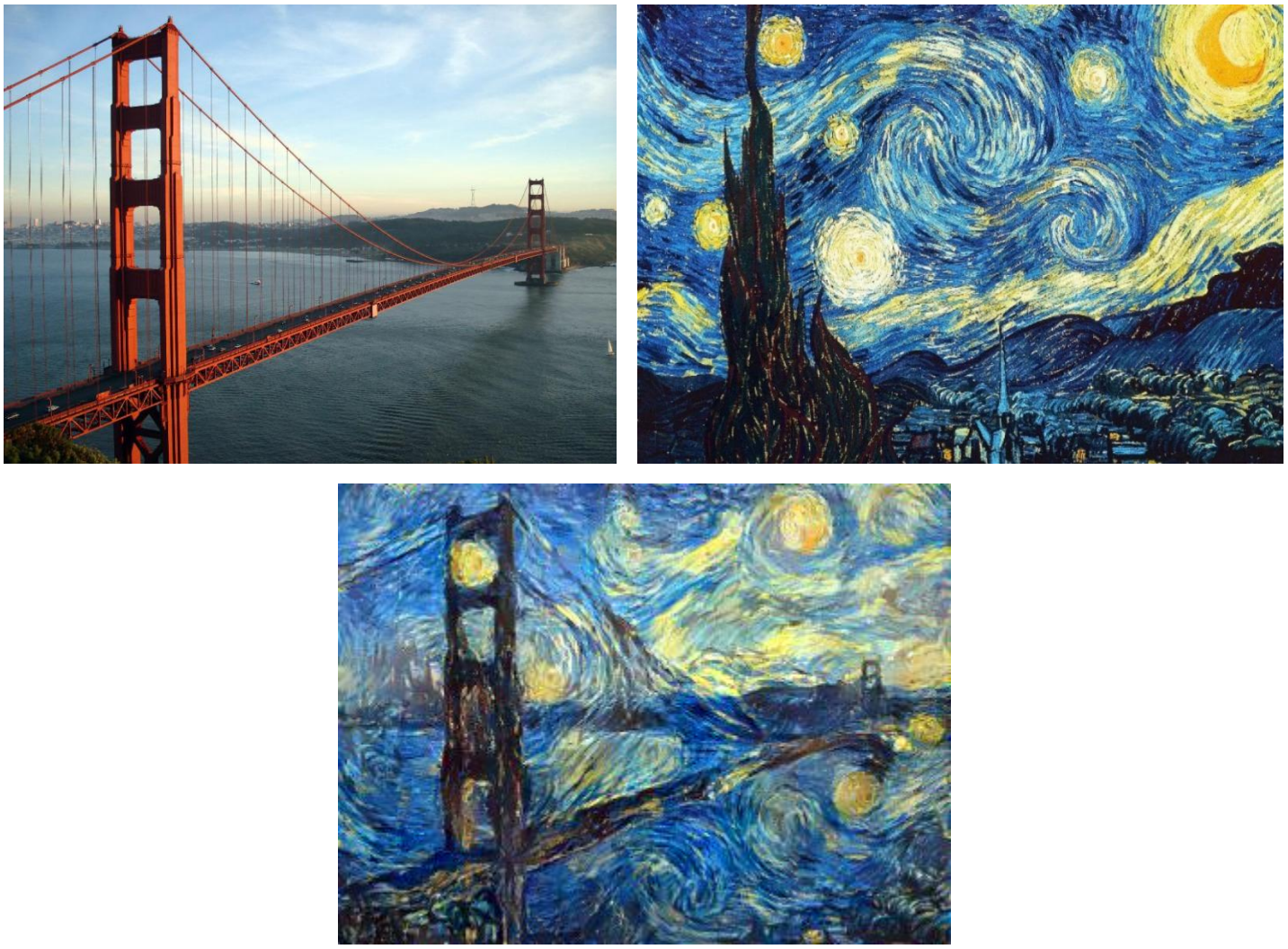
1. **Feature Extraction**: Utilizing the VGG19 network and torchvision's feature extraction API
2. **Loss Calculation**: Computing content, style, and regularization losses
3. **Optimization**: Employing L-BFGS with a closure function for gradient updates
4. **Parameter Experimentation**: Systematic exploration of the parameter space
5. **Analysis**: Tools for visualizing and quantifying results

4.2 Feature Extraction and Representation

The VGG19 network, pretrained on *ImageNet*, serves as the feature extractor. Following the methodology of Gatys et al. (2016), we extract features from specific layers:

```
return_nodes = {'1': 'relu_1_1',
                '6': 'relu_2_1',
                '11': 'relu_3_1',
                '20': 'relu_4_1',
                '22': 'relu_4_2',
                '29': 'relu_5_1'}
```

Content representation is captured from layer *relu_4_2*, while style representation incorporates multiple layers from *relu_1_1* through *relu_5_1*, providing multi-scale style information as recommended by Gatys et al. (2016).



Figures 1-3: Content image (Bridge) + Style image (Starry Night) and the resultant Image created combining the content and style

The same content image can be combined with various styles to create some unimaginable outputs. A few examples of such are as below:

Figures 4 and 5: Content image (Bridge) + Style image (Abstract) and the resultant Image created combining the content and style

The resultant in this case was made using the same Bridge as content image and by changing the style image to Absract style.

Style Image

Content Image

Resultant Image

Figures 6-8: Content image (City) + Style image (Painting) and the resultant Image created combining the content and style

4.3 Loss Function Implementation

The implementation incorporates three key loss components:

4.3.1 Content Loss

```
def calc_content_loss(features, targets, nodes):
    """Calculate Content Loss."""
    content_loss = 0
    for node in nodes:
        content_loss += mse_loss(features[node], targets[node])
    return content_loss
```

This directly implements the *mean squared error* between feature representations, as formalized in the theoretical framework.

4.3.2 Style Loss with Gram Matrix Calculation

```
"""Transfer a feature to gram matrix."""
def gram(x):
    b, c, h, w = x.size()
    f = x.flatten(2)
    g = torch.bmm(f, f.transpose(1, 2))
    return g.div(h*w)


"""Calculate Gram Loss."""
def calc_gram_loss(features, targets, nodes):
    gram_loss = 0
    for node in nodes:
        gram_loss += mse_loss(gram(features[node]), gram(targets[node]))
    return gram_loss
```

The Gram matrix calculation *captures style information* by measuring feature correlations, following Gatys' original approach. This implementation efficiently computes the Gram matrix using batch matrix multiplication.

4.3.3 Total Variation Loss

```
def calc_tv_loss(x):
    """Calculate Total Variation Loss."""
    tv_loss = torch.mean(torch.abs(x[:, :, :, :-1] - x[:, :, :, 1:]))
    tv_loss += torch.mean(torch.abs(x[:, :, :-1, :] - x[:, :, 1:, :]))
```

```
    return tv_loss
```

This implements the total variation *regularization term* that promotes *spatial smoothness* by penalizing large differences between adjacent pixels.

4.4 Optimization Process

The optimization employs L-BFGS with a closure function, following the approach described by Gatys et al. (2016):

```
optimizer = torch.optim.LBFGS([input_image], lr=args.learning_rate)
for i in tqdm(range(args.iteration), desc='Stylization'):
    def closure():
        optimizer.zero_grad()

        # Extract features from input image
        input_features = feature_extractor(input_image)

        # Calculate losses
        content_loss = calc_content_loss(input_features, content_features, content_nodes)
        style_loss = calc_gram_loss(input_features, style_features, style_nodes)
        tv_loss = calc_tv_loss(input_image)

        # Weighted total loss
        total_loss = content_loss * args.content_loss_weight \
            + style_loss * args.style_loss_weight \
            + tv_loss * args.tv_loss_weight

        total_loss.backward()

        return total_loss

    optimizer.step(closure)
```

The closure function enables the L-BFGS optimizer to evaluate both the *loss and its gradient* efficiently during each iteration.

4.5 Hyperparameter Experimentation Framework

The implementation includes a systematic approach to hyperparameter experimentation:

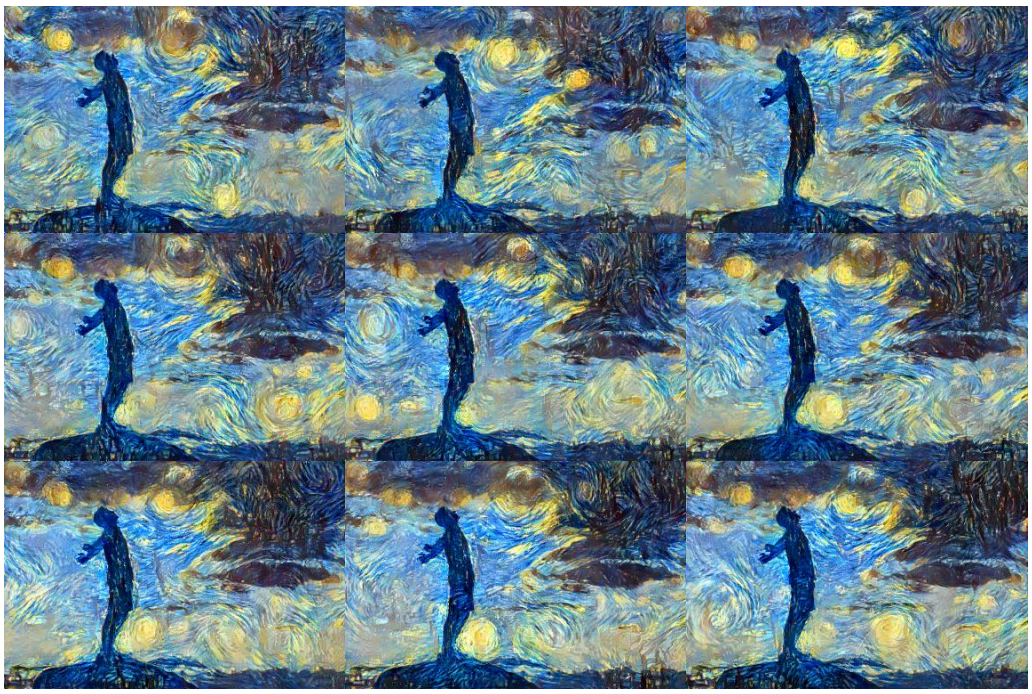
```
# Define parameter combinations to test
noise_ratios = [0.3, 0.5, 0.7]
learning_rates = [0.4, 0.7, 1.0]

# Run all combinations
for nr in noise_ratios:
    for lr in learning_rates:
        print(f"\nRun {run_count}/{total_runs}: noise_ratio={nr}, learning_rate={lr}")
```

This grid search approach enables *comprehensive evaluation* of how different parameter combinations affect *stylization quality* and *convergence behavior*.

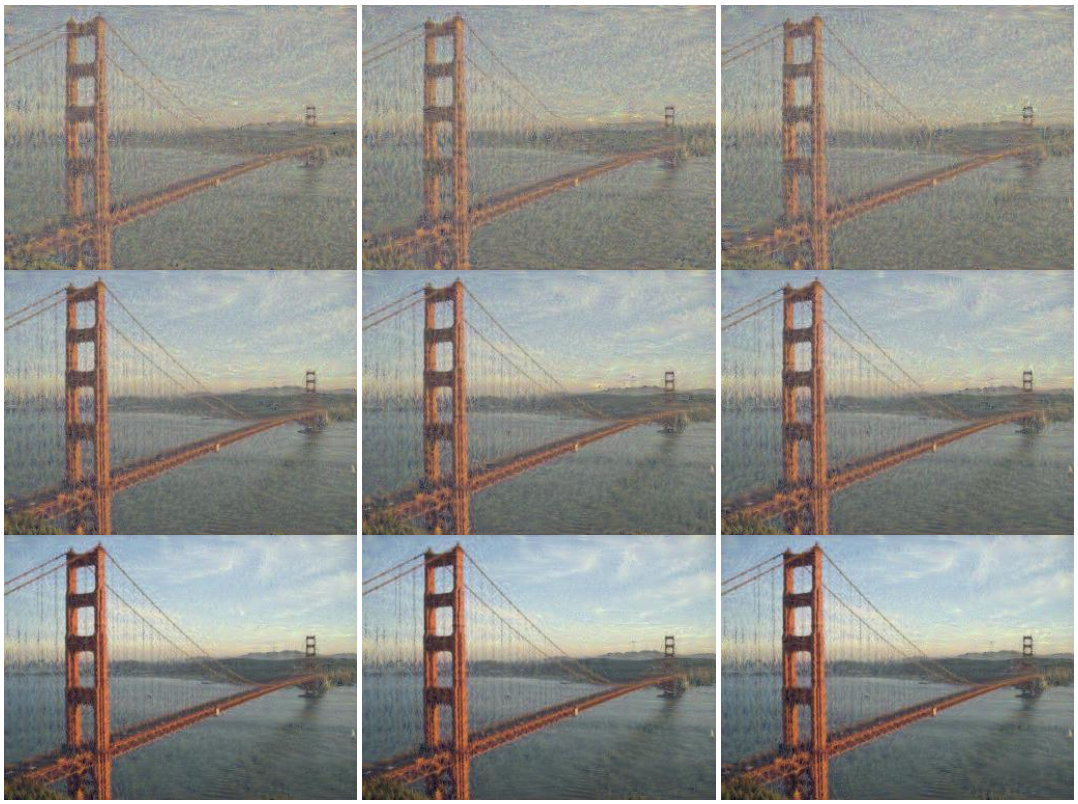
5. Experimental Results and Analysis

5.1 Comparative Visual Analysis

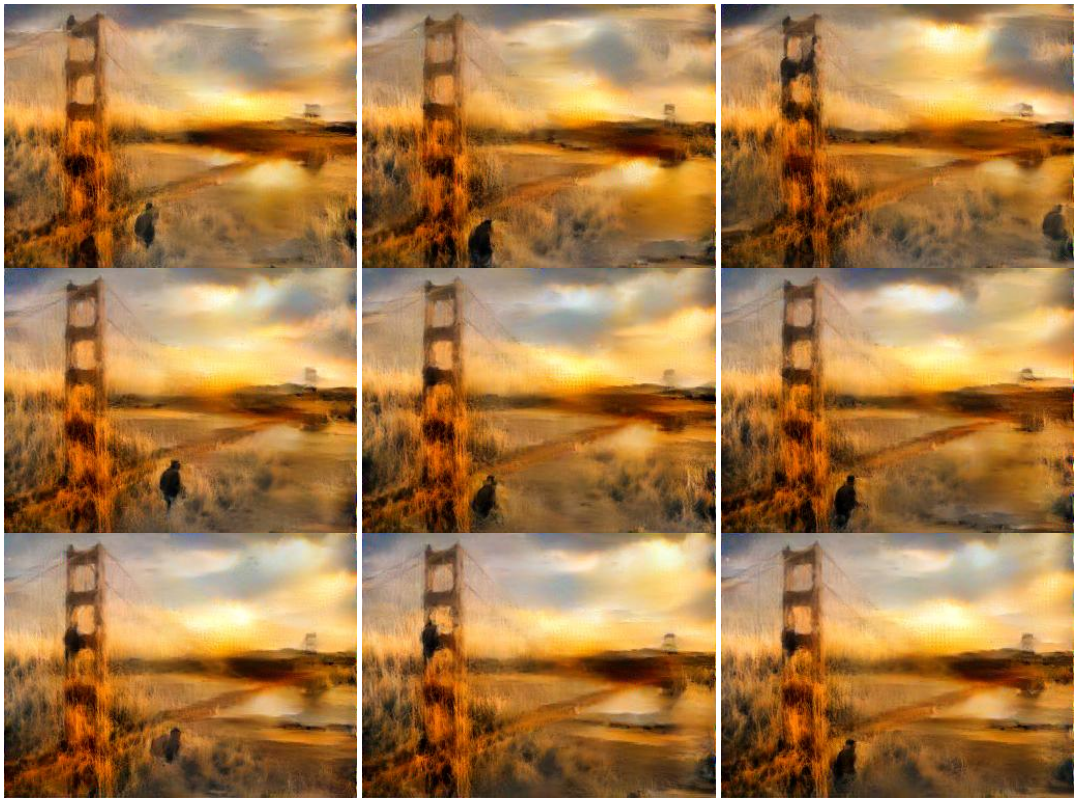
Figures 9-17: Feature image Increasing Learning Rate (from Left to Right) and Increasing Noise Ratio (from Top to Bottom), Learning Rate = [0.4, 0.7, 1.0] and Noise Ratio = [0.3, 0.5, 0.7]



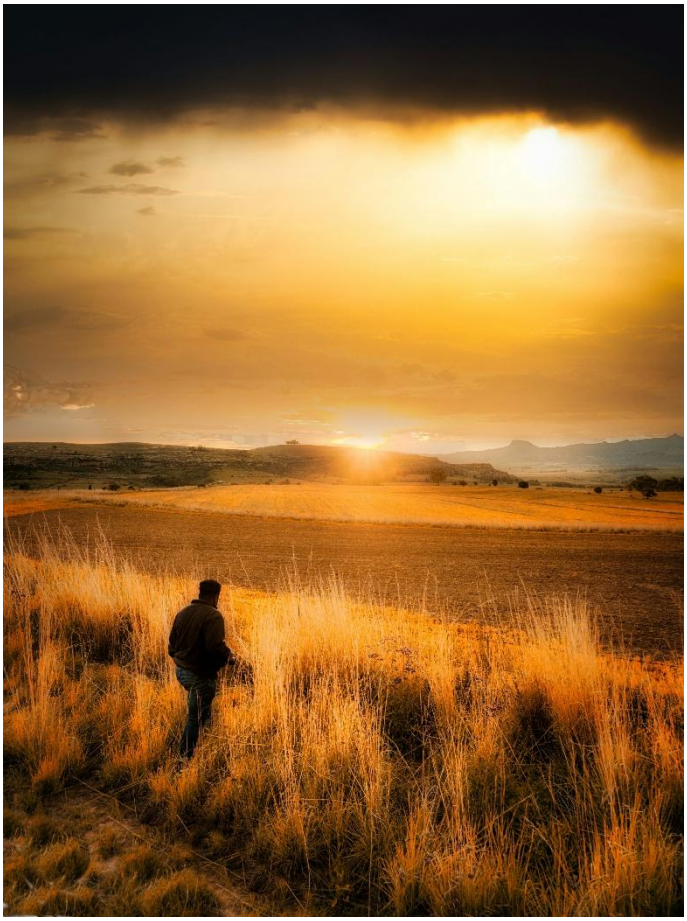
Figures 18-26: Resultant stylized images



Figures 27-35: Feature image Increasing Learning Rate (from Left to Right) and Increasing Noise Ratio (from Top to Bottom), Learning Rate = [0.4, 0.7, 1.0] and Noise Ratio = [0.3, 0.5, 0.7]



Figures 36-44: Feature image Increasing Learning Rate (from Left to Right) and Increasing Noise Ratio (from Top to Bottom), Learning Rate = [0.4, 0.7, 1.0] and Noise Ratio = [0.3, 0.5, 0.7]



Figures 45&46 and 47&48: The content image and style image pair for the Figures 27-35 and 36-44 respectively

The visual comparison demonstrates:

1. **Textural Variations**: Lower noise ratios produce more pronounced texture transfer, particularly visible in background regions.
2. **Color Fidelity**: Higher noise ratios tend to preserve the color palette of the content image more faithfully, while lower noise ratios adopt more of the style image's color scheme.

3. **Edge Preservation**: The learning rate significantly impacts edge preservation, with lower rates maintaining more precise edges and object boundaries.
4. **Artifact Presence**: Higher learning rates occasionally introduce artifacts, particularly in regions with sharp color transitions or fine details.

5.2 Effect of Noise Ratio on Stylization

The *noise ratio parameter* controls the initialization of the input image, ranging from *pure random noise (0.0)* to the *original content image (1.0).* Our experiments systematically evaluated noise ratios of 0.3, 0.5, and 0.7 while keeping other parameters constant.

5.2.1 Qualitative Analysis

Low Noise Ratio (0.3):

- Produces more *creative* and *distinctive* stylizations
- Shows greater *textural variety* and *artistic* influence
- May compromise content structure in some cases
- Typically requires more iterations to converge

Medium Noise Ratio (0.5):

- Offers balanced preservation of content and style elements
- Provides stable convergence in most cases
- Results in natural-looking stylistic integration
- Often represents an optimal compromise for general use

High Noise Ratio (0.7):

- Strongly preserves content structure and details
- Results in more conservative stylizations
- Converges more quickly and reliably
- May under-emphasize subtle stylistic elements

5.2.2 Quantitative Analysis (Noise Ratios)

Our analysis of loss trajectories across noise ratios reveals:

1. **Content Loss**: Higher noise ratios (0.7) start with *lower content loss* values and converge more quickly due to the *stronger initialization* bias toward the content image.
2. **Style Loss**: Lower noise ratios (0.3) typically achieve *lower final style loss* values, indicating stronger style transfer, but require more iterations to reach convergence.
3. **Convergence Behavior**: The noise ratio significantly impacts the optimization trajectory, with higher noise ratios following smoother convergence paths compared to the more exploratory behavior seen with lower noise ratios.

5.3 Effect of Learning Rate on Stylization

The learning rate parameter controls the step size during optimization. Our experiments evaluated learning rates of 0.4, 0.7, and 1.0 across different noise ratio settings.

5.3.1 Qualitative Analysis (Learning Rates)

Low Learning Rate (0.4):

- Produces *smoother*, *more refined results*
- Demonstrates better *handling of fine details*
- Shows *slower* but *more stable convergence*
- May *not fully explore the style space* within iteration constraints

Medium Learning Rate (0.7):

- *Balances exploration and refinement*
- Achieves *good stylization quality* within reasonable iteration counts
- Shows *robust performance* across different content-style pairs
- Represents a reliable default setting

High Learning Rate (1.0):

- Achieves *faster initial stylization*
- May introduce artifacts or oversaturated regions
- Shows *less stable convergence behavior*
- Can occasionally produce more *striking or dramatic results*

5.3.2 Quantitative Analysis

Analysis of the optimization trajectories across learning rates reveals:

1. **Convergence Speed**: *Higher learning rates* (1.0) show *faster initial loss reduction* but may plateau or oscillate in later iterations. In contrast, *lower learning rates* (0.4) show more *gradual but steady improvement* throughout the optimization process.
2. **Final Loss Values**: Interestingly, *moderate learning rates* (0.7) often achieve the *lowest final total loss* values, suggesting an *optimal balance* between exploration and refinement.
3. **Loss Component Behaviors**: The content loss tends to be *more sensitive to learning rate variations* than style loss, particularly in early iterations.

5.4 Parameter Interaction Analysis

A key finding from our experiments is the significant interaction between noise ratio and learning rate parameters:

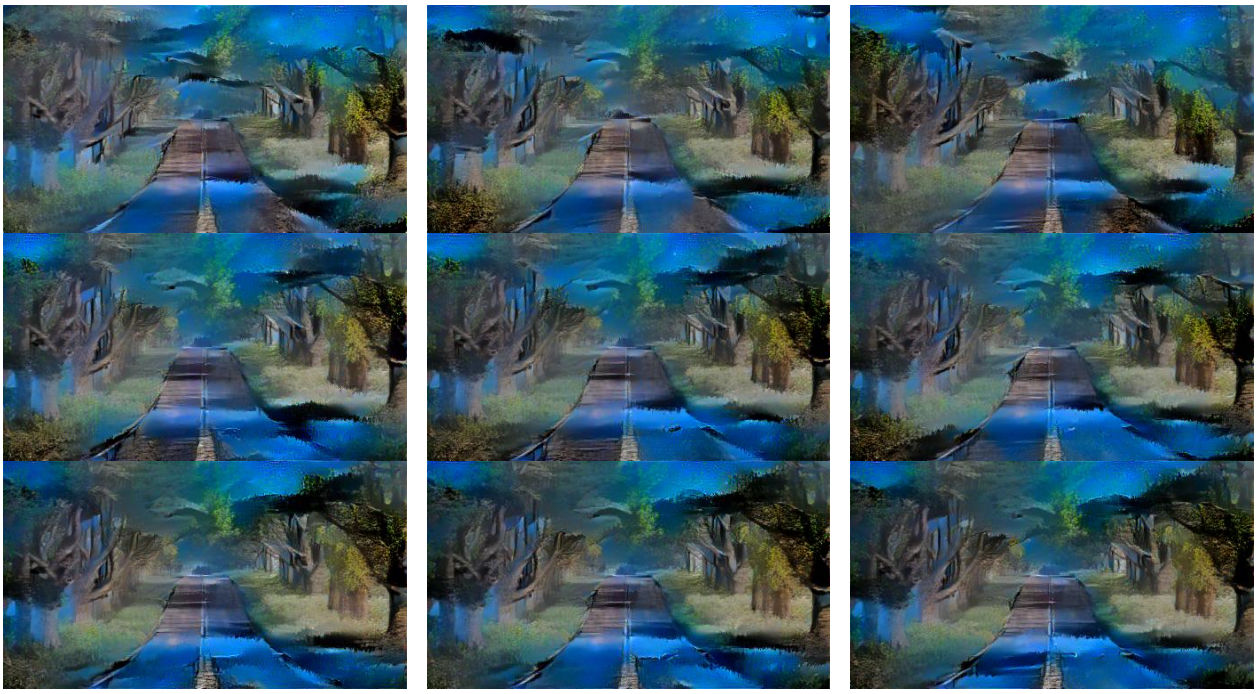
1. **Complementary Effects**: *Lower noise ratios* (0.3) often perform best with *moderate learning rates* (0.7), while higher noise ratios (0.7) can benefit from higher learning rates (1.0) to counteract the strong content bias.
2. **Stability Considerations**: The combination of *low noise ratio* (0.3) and *high learning rate* (1.0) shows the *least stable behavior*, occasionally leading to divergence or artifact-rich results.

3. **Optimal Combinations**: The parameter combination of *noise ratio 0.5 and learning rate 0.7* consistently produces *balanced results* across different content-style pairs, suggesting a robust default configuration.

5.5 Unexpected and Unsuccessful Results

Figures 49 and 50: The style and content image for the results below

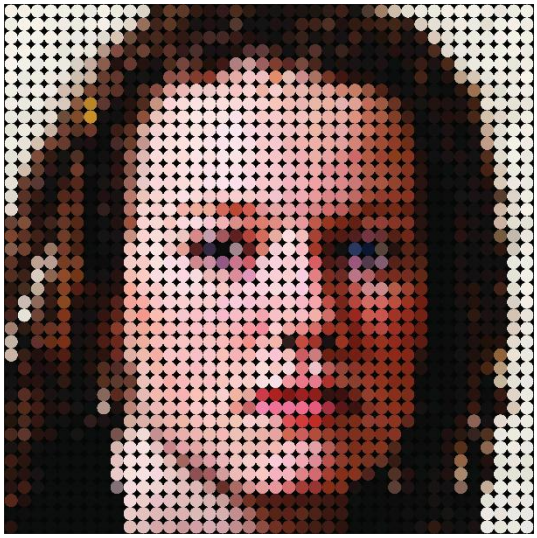
Figures 51-59: Feature and content extraction for increasing learning rates and noise ratios from left to right and top to bottom respectively.



Figures 60-68: The resultant images

Figures 69 and 70: The style and content image for the results below

Figure 71: Resultant Image



Figures 72 and 73: Input Images – Circles (Style image) and Woman (Content image)

Figures 74-79: The results after 1, 200, 400, 600, 800 and 1000 iterations

6. Discussion

6.1 Optimal Parameter Selection Guidelines

Based on our experimental results, we propose the following guidelines for parameter selection:

1. **For Maximum Content Preservation**: Use noise ratio 0.7 with learning rate 0.7-1.0, and consider increasing the content loss weight.
2. **For Strong Stylistic Effect**: Use noise ratio 0.3-0.5 with learning rate 0.4-0.7, and consider increasing the style loss weight.
3. **For Balanced Results**: Use noise ratio 0.5 with learning rate 0.7, maintaining equal emphasis on content and style loss weights.
4. **For Challenging Style-Content Pairs**: When content and style images have dramatically different characteristics (e.g., photo vs. abstract or painting), start with noise ratio 0.5 and learning rate 0.4 for more controlled stylization.

These guidelines align with observations by *Jing et al. (2019)*, who noted that "the parameter settings significantly affect the quality and characteristics of the generated image, making them critical factors in achieving desired artistic effects."

6.2 Comparison with Alternative Approaches

While this implementation follows the original optimization-based approach of *Gatys* et al. (2016), it's worth noting alternative approaches in the literature:

1. **Feed-forward Networks**: As demonstrated by *Johnson et al. (2016),* feed-forward networks can achieve real-time style transfer after training, trading flexibility for speed.
2. **Adaptive Instance Normalization**: The AdaIN approach by *Huang and Belongie (2017)* enables arbitrary style transfer through a single feed-forward network by aligning content feature statistics with style features.
3. **Style Attentional Networks**: As proposed by *Park and Lee (2019),* attention mechanisms can enhance style transfer by focusing on semantically meaningful regions.

Our optimization-based approach provides *greater flexibility* and *control over stylization parameters* but at the cost of *computational efficiency* compared to these alternative methods.

6.3 Limitations and Future Work

Several limitations of the current implementation offer opportunities for future enhancement:

1. **Computational Efficiency**: The optimization process remains computationally intensive. Future work could explore *optimization acceleration techniques* or *hybrid approaches combining optimization-based and feed-forward methods*.
2. **Semantic Awareness**: The current approach treats all image regions equally, ignoring semantic content. Integration of semantic segmentation could enable region-specific stylization.
3. **Color Preservation**: The implemented method sometimes produces significant color shifts. Implementing color histogram matching or other color preservation techniques could address this limitation.
4. **Evaluation Metrics**: Development of quantitative metrics for style transfer quality would enable more objective comparison across parameter settings and algorithms.

7. Conclusion

This project has implemented and analyzed a feature-based neural style transfer system using the PyTorch framework. Through systematic experimentation with key parameters (noise ratio and learning rate), we have demonstrated the significant impact of these variables on stylization quality and identified optimal parameter combinations for different stylization objectives.

The analysis reveals the complex interplay between initialization strategy (noise ratio) and optimization dynamics (learning rate), providing insights into the fundamental trade-offs in neural style transfer between content preservation and style adoption. The developed experimental framework offers a valuable tool for future research in this domain, enabling systematic evaluation of additional parameters and algorithmic variations.

By situating this work within the broader context of neural style transfer research, we have contributed to the ongoing refinement of techniques for computational artistic creation, demonstrating both the power and limitations of current approaches in this rapidly evolving field.

8. References

1. *Gatys, L. A., Ecker, A. S., & Bethge, M. (2016). Image style transfer using convolutional neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2414-2423).*
2. *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). Springer, Cham.*
3. *Jing, Y., Yang, Y., Feng, Z., Ye, J., Yu, Y., & Song, M. (2019). Neural style transfer: A review. IEEE transactions on visualization and computer graphics, 26(11), 3365-3385.*
4. *Mahendran, A., & Vedaldi, A. (2015). Understanding deep image representations by inverting them. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 5188-5196).*
5. *Li, Y., Wang, N., Liu, J., & Hou, X. (2017). Demystifying neural style transfer. In Proceedings of the 26th International Joint Conference on Artificial Intelligence (pp. 2230-2236).*
6. *Huang, X., & Belongie, S. (2017). Arbitrary style transfer in real-time with adaptive instance normalization. In Proceedings of the IEEE International Conference on Computer Vision (pp. 1501-1510).*
7. *Park, D. Y., & Lee, K. H. (2019). Arbitrary style transfer with style-attentional networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 5880-5888).*
8. *Gatys, L. A., Ecker, A. S., Bethge, M., Hertzmann, A., & Shechtman, E. (2017). Controlling perceptual factors in neural style transfer. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 3985-3993).*