A4: Style Transfer by Pavan Sesha Sai Kasukurti

This report summarizes my implementation of the neural style transfer algorithm based on Gatys et al.'s paper "Image Style Transfer Using Convolutional Neural Networks" (2016). The implementation uses a pre-trained VGG19 network to extract content and style features from images, then optimizes a target image to match both the content representation of one image and the style representation of another. Through this process, I was able to generate images that preserve the content of one image while adopting the artistic style of another.

Implementation Details

The neural style transfer algorithm was implemented in PyTorch following the structure provided in the Style_Transfer_Exercise notebook. I completed several TODOs that were essential to the functioning of the algorithm:

TODO 1: Layer Mapping Implementation

I implemented the layer mapping function to extract features from specific layers of the VGG19 network:

```
def get_features(image, model, layers=None):
    if layers is None:
        layers = {
            '0': 'conv1_1',
            '5': 'conv2_1',
            '10': 'conv3_1',
            '19': 'conv4_1',
            '21': 'conv4_2',  # content representation
            '28': 'conv5_1'
        }

    features = {}
    x = image
    for name, layer in model._modules.items():
        x = layer(x)
        if name in layers:
            features[layers[name]] = x

    return features
```

This mapping follows the original paper's approach, using the second convolution of the fourth layer (`conv4_2`) for content representation, while using the first convolutions of each layer for style representation.

TODO 2: Gram Matrix Calculation

I implemented the Gram matrix calculation function, which is crucial for capturing style information:

```
def gram_matrix(tensor):
    _, d, h, w = tensor.size()
    tensor = tensor.view(d, h * w)
    return torch.mm(tensor, tensor.t())
```

This function reshapes the tensor to multiply the features for each channel, creating a matrix that captures the correlations between different feature maps - essentially representing the style characteristics.

TODO 3: Content Loss Calculation

I implemented the content loss calculation, which measures how well the content of the original image is preserved:

```
target_features = get_features(target, vgg)
content_loss = torch.mean((target_features['conv4_2'] - content_features['conv4_2'])**2)
```

This calculates the mean squared error between the feature representations of the target and content images at the `conv4_2` layer.

TODO 4: Style Loss Calculation

I implemented the style loss calculation, which measures how well the style of the style image is captured:

```
target_gram = gram_matrix(target_feature)
style_gram = style_grams[layer]
layer_style_loss = style_weights[layer] * torch.mean((target_gram - style_gram)**2)
```

This calculates the weighted mean squared error between the Gram matrices of the target and style images for each style layer.

TODO 5: Total Loss Calculation

Finally, I implemented the total loss calculation, which combines content and style losses:

```
total_loss = content_weight * content_loss + style_weight * style_loss
```

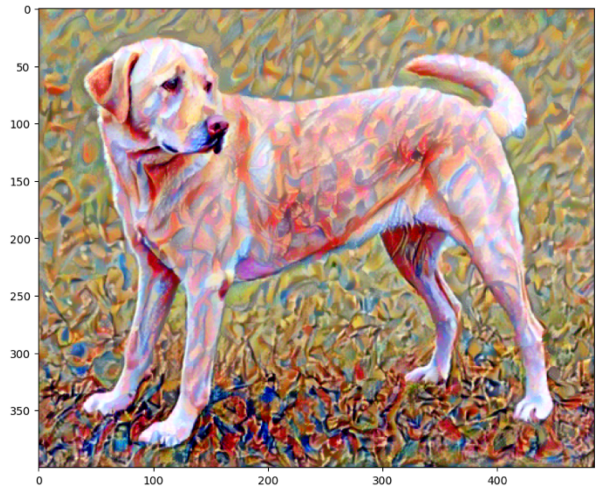
This weighted sum allows us to control the balance between preserving content and transferring style.

Default Results

For my implementation, I used a photograph of a Yellow Labrador(left) as the content image and Kandinsky's Composition 7 as the style image(right).

Initial:


Final Result with default values:

Default parameters:

- Content weight (α): 1
- Style weight (β): 1e6
- Style layer weights: {'conv1_1': 1.0, 'conv2_1': 0.8, 'conv3_1': 0.5, 'conv4_1': 0.3, 'conv5_1': 0.1}
- Learning rate: 0.003
- Iterations: 2000

Hyperparameter Experiments

I conducted several experiments by varying key hyperparameters to understand their impact on the style transfer results.

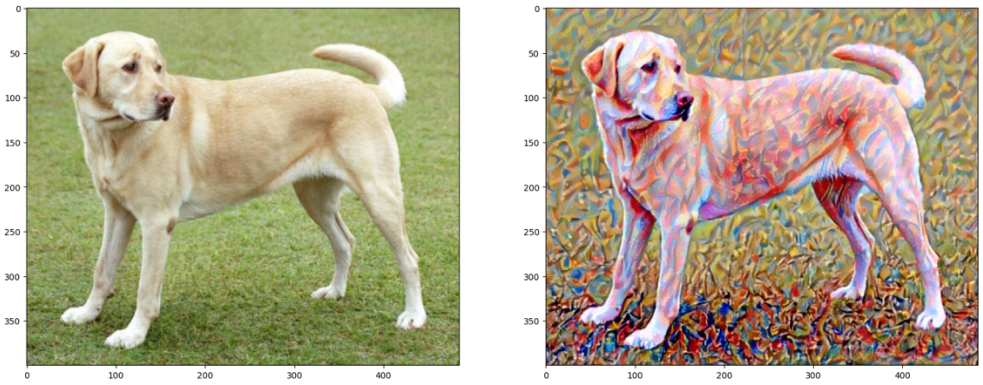
Experiment 1: Style Weight Variation

The style weight (β) controls the emphasis on style versus content:

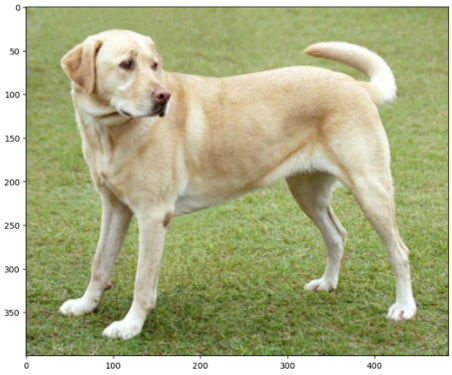
Style Weight	Observation
1e4	Minimal style transfer, content clearly preserved
1e6 (default)	Good balance between content and style
1e8	Heavy stylization, content structure barely visible

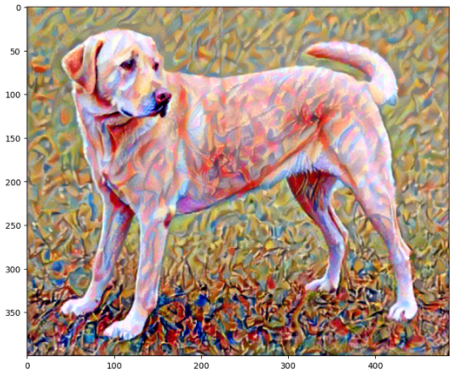
At lower style weights (1e4), the output closely resembled the content image with subtle style elements. The default value (1e6) provided a good balance, while at very high values (1e8), the artistic elements became so pronounced that they overwhelmed the content structure.

1e4:

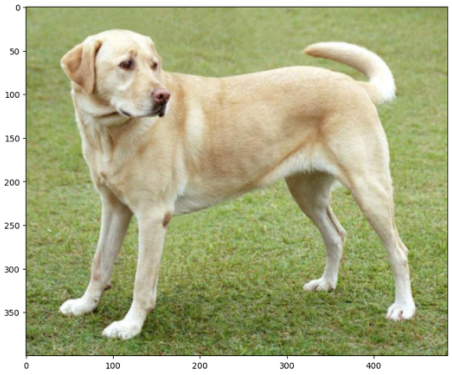


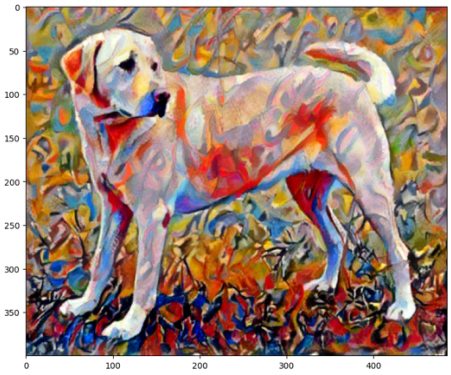
1e6:





1e8:





Experiment 2: Layer Weights Distribution

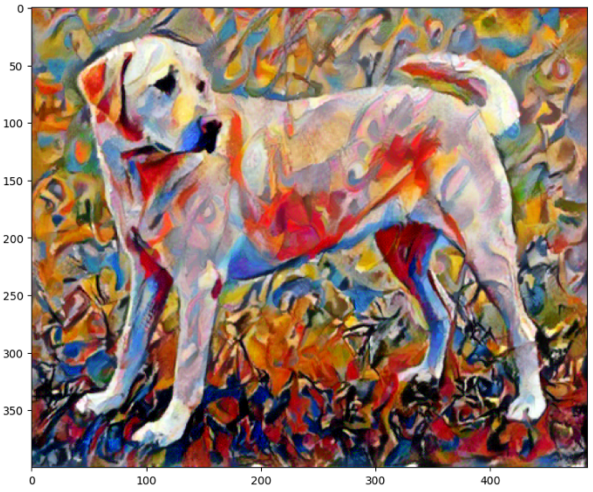
I experimented with different distributions of weights across style layers:

Layer Weight Distribution	Effect
Higher weights on early layers	Larger style elements (brush strokes, color patterns)
Higher weights on later layers	Finer details and textures
Uniform weights	Balanced style transfer across scales

When early layers (conv1_1, conv2_1) received higher weights, the result showed larger style artifacts like broad brush strokes and color schemes. Emphasizing later layers produced more detailed textures while preserving the overall structure.

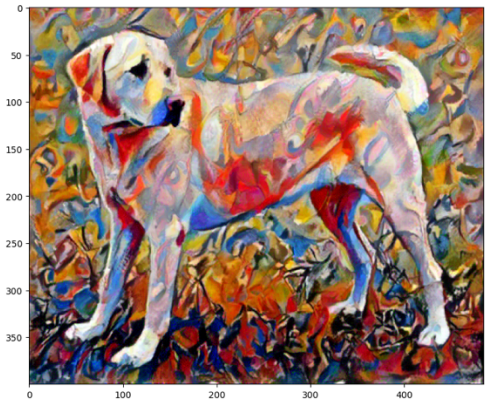
Higher weights on early layers:

```
style_weights = {'conv1_1': 1.5,
                 'conv2_1': 1.0,
                 'conv3_1': 0.5,
                 'conv4_1': 0.2,
                 'conv5_1': 0.1}
```



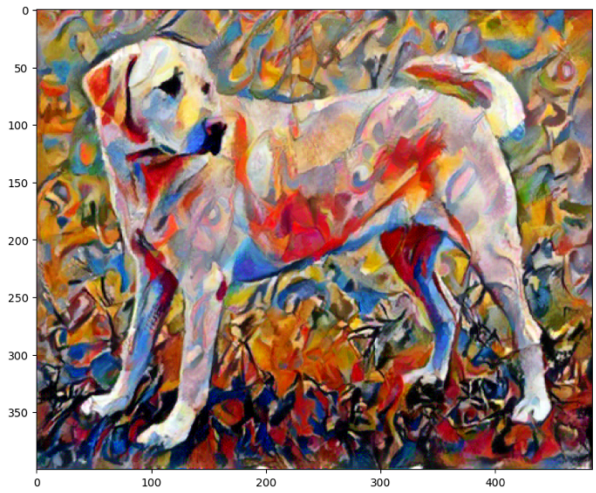
Higher weights on later layers:

```
style_weights = {'conv1_1': 0.1,
                 'conv2_1': 0.3,
                 'conv3_1': 0.5,
                 'conv4_1': 0.8,
                 'conv5_1': 1.0}
```



Uniform Weights across all layers:

```
style_weights = {'conv1_1': 0.5,
                 'conv2_1': 0.5,
                 'conv3_1': 0.5,
                 'conv4_1': 0.5,
                 'conv5_1': 0.5}
```



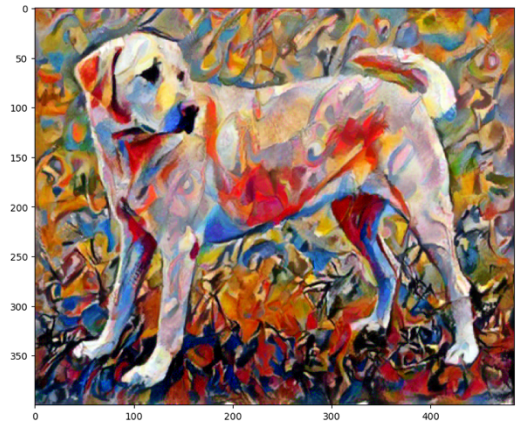
Experiment 3: Number of Iterations

I varied the number of optimization steps:

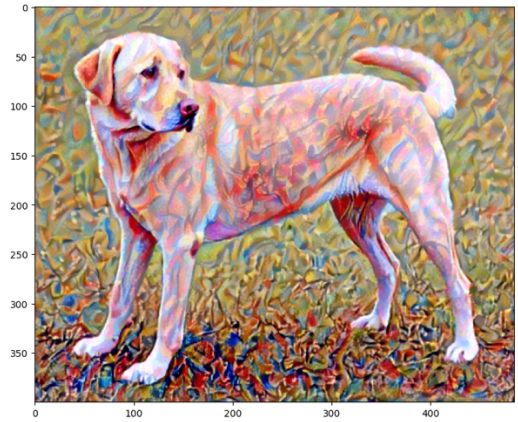
Iterations	Result
500	Rough transfer, incomplete stylization
2000 (default)	Good balance of quality and computation time
5000	More refined details, diminishing returns

With fewer iterations (500), the style transfer appeared incomplete. At 2000 iterations, most style elements were well-integrated. Beyond this point (5000), improvements became more subtle, with diminishing returns for the additional computation time.

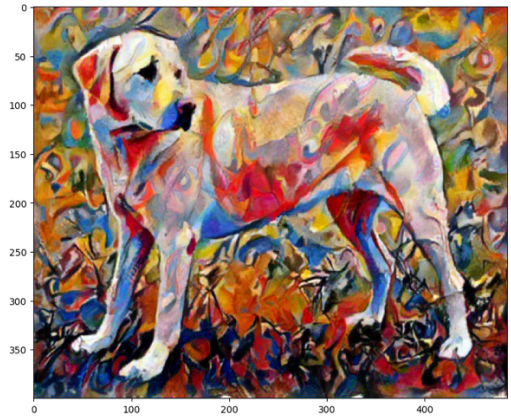
500 Iterations result:

2000 iterations result:

5000 iterations result:

Experiment 4: Learning Rate

I tested different learning rates for the optimizer:

Learning Rate	Observation
0.001	Slower convergence, more stable results
0.003 (default)	Good balance of speed and stability
0.01	Faster initial stylization, sometimes unstable

Lower learning rates produced more stable but slower convergence, while higher rates accelerated the process but occasionally introduced artifacts or instability in the optimization.

Experiment 5: Initial Image

I experimented with different starting points for the optimization:

Initial Image	Result
Content image (default)	Consistent results, preserves content structure
Random noise	More diverse results, less content preservation
Blend of content and style	Interesting hybrid results

Starting with the content image produced the most reliable results. Random noise initialization led to more unpredictable outcomes with less content preservation, while a blend offered an interesting middle ground.

Analysis and Findings

The most significant findings from my experiments include:

1. **Style Weight Impact**: The style weight is the most influential parameter, dramatically affecting the balance between content preservation and style transfer. Too low, and the style is barely noticeable; too high, and the content becomes unrecognizable.

2. **Layer Weight Distribution**: The distribution of weights across different layers allows fine control over which aspects of the style are emphasized. Early layers capture broader style elements like color schemes and large brush strokes, while later layers capture finer textures and patterns.

3. **Iteration Efficiency**: While more iterations generally improve results, there's a point of diminishing returns. For most applications, 2000 iterations provide a good balance between quality and computational efficiency.

4. **Learning Rate Balance**: The learning rate affects both the speed and stability of convergence. A rate of 0.003 offers a good compromise, though lower rates may be preferable for more complex or detailed style transfers.

5. **Initial Image Influence**: The choice of initial image affects both the convergence path and final result. Starting with the content image provides the most reliable content preservation.

Conclusions

This implementation successfully reproduced the neural style transfer algorithm from Gatys et al.'s paper. The experiments revealed that neural style transfer is highly sensitive to hyperparameter choices, with each parameter controlling different aspects of the stylization process.

The algorithm demonstrates how deep neural networks can separate and recombine content and style representations of images, creating novel artistic compositions that blend the semantic content of one image with the stylistic elements of another.

The most effective results were achieved with a moderate style weight (1e6), a distribution of layer weights that emphasized earlier layers slightly more than later ones, approximately 2000 iterations, and a learning rate of 0.003.

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).