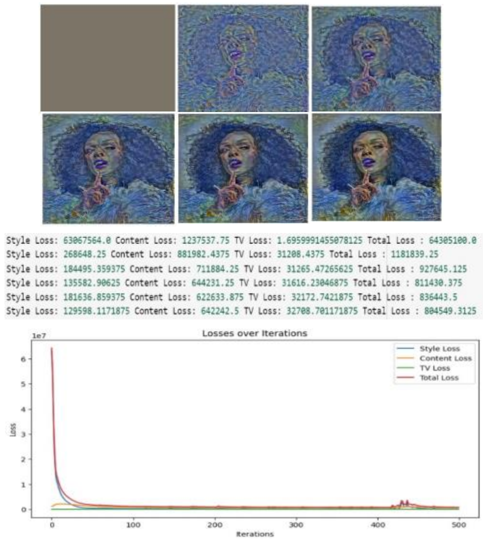
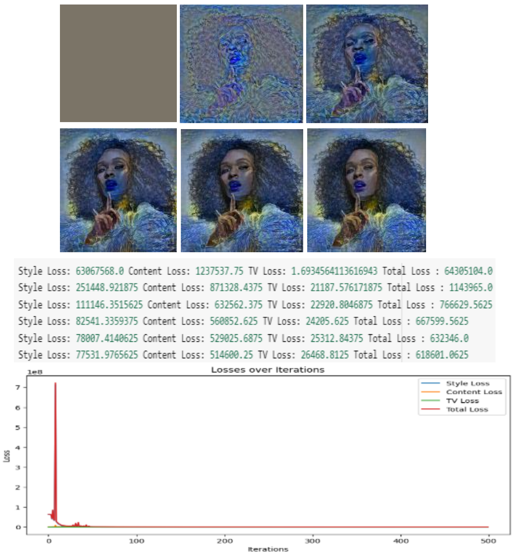
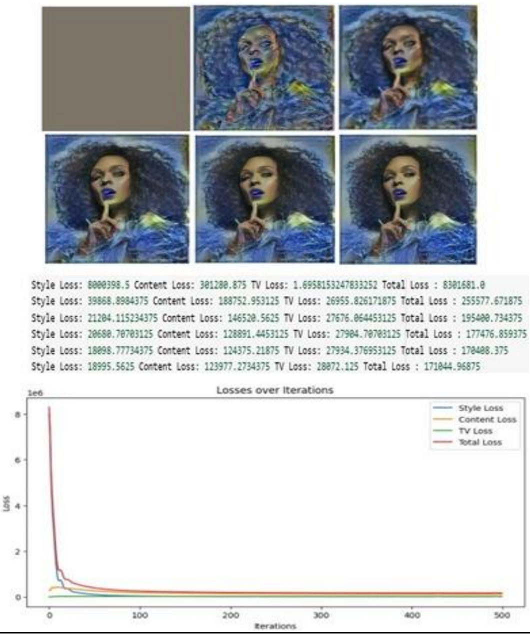
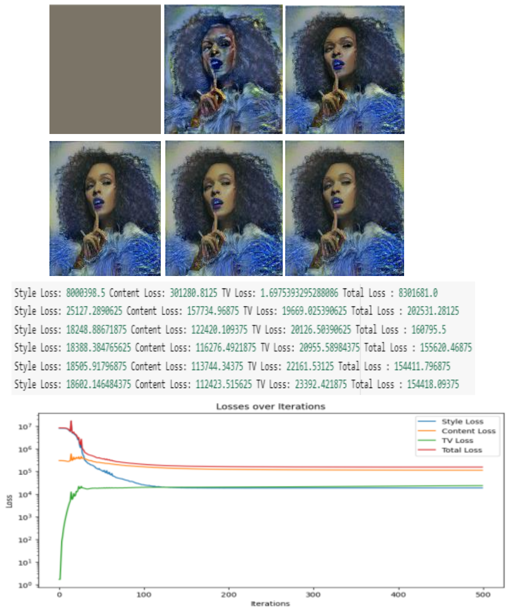
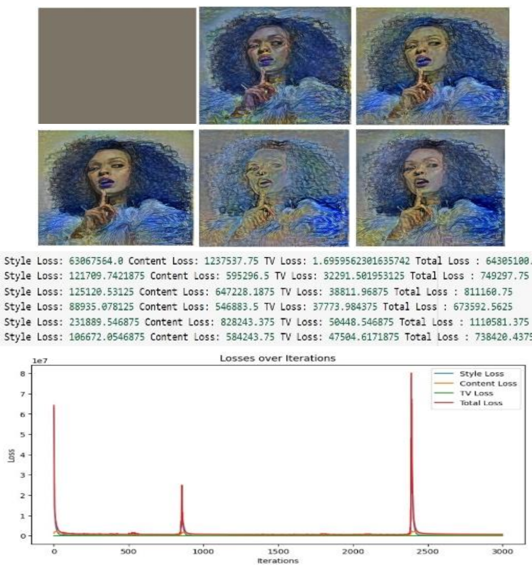
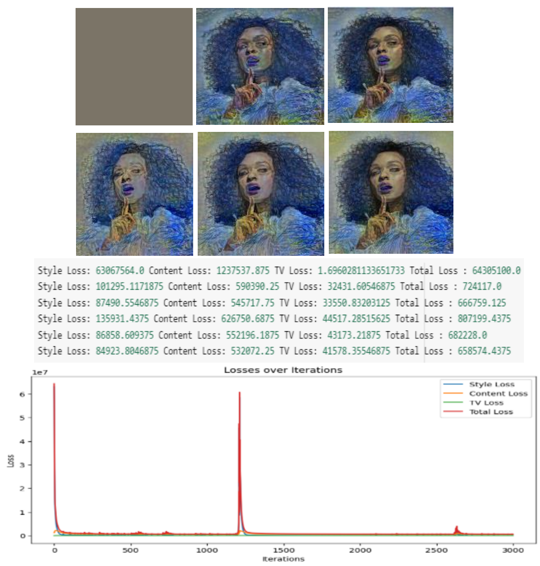
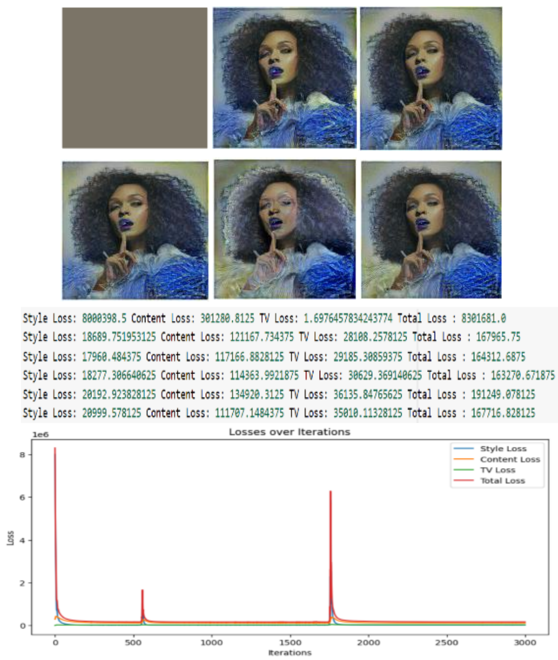
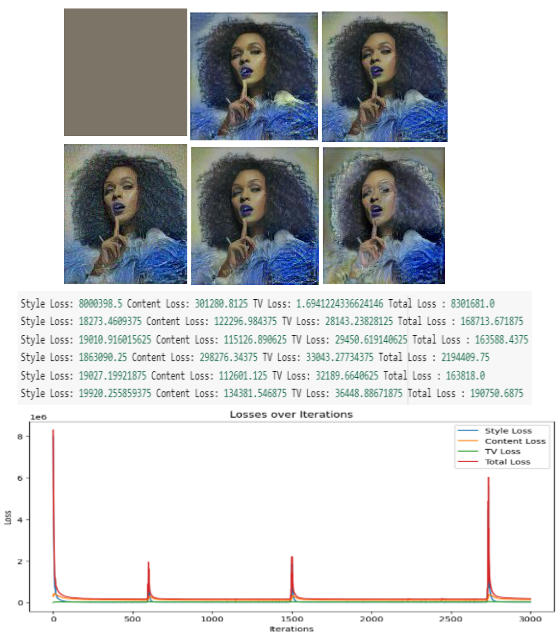
RESULT ANALYSIS

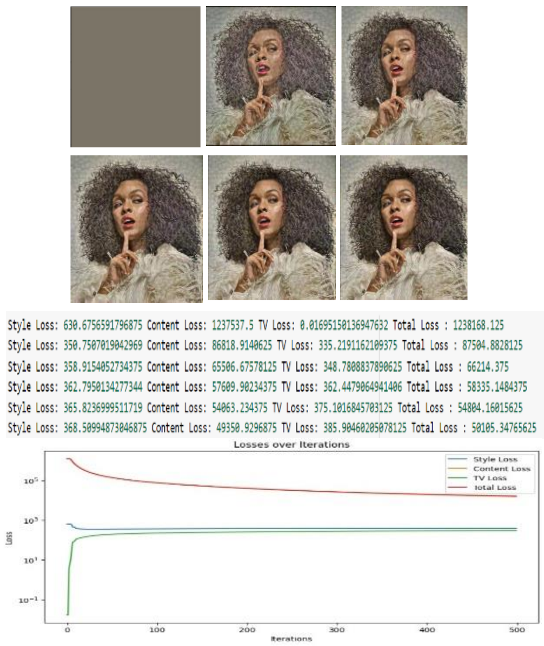
<table>
<tr><td></td><td colspan="2">ReLU LAYERS</td></tr>
<tr><td>Hyperparameter</td><td>ADAM Optimizer</td><td>LBFGS Optimizer</td></tr>
<tr><td>500 iteration
Pooling: max
weights:
lr=10
alpha=5e0
beta=1e2
tv=1e-3</td><td>
</td><td>
</td></tr>
<tr><td>500 iteration
Pooling: avg
weights:
lr=10
alpha=5e0
beta=1e2
tv=1e-3</td><td>
</td><td>
</td></tr>
<tr><td>3000 iteration
Pooling: max
weights:
lr=10
alpha=5e0
beta=1e2
tv=1e-3</td><td>
</td><td>
</td></tr>
</table>

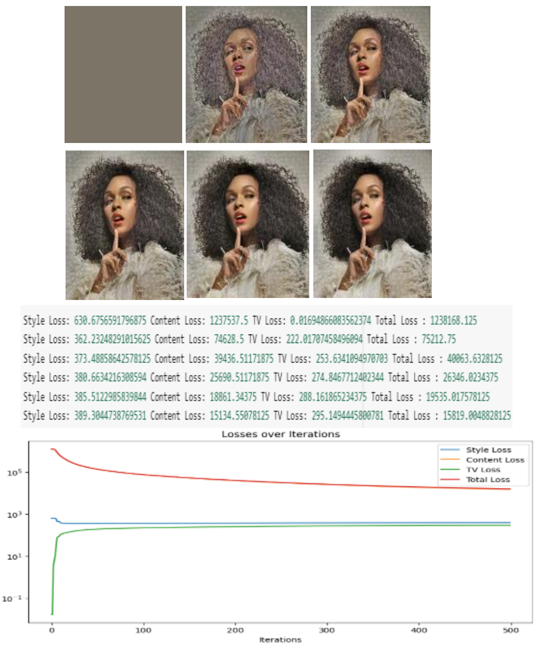
3000 iteration
Pooling: avg
weights:
lr=10
alpha=5e0
beta=1e2
tv=1e-3



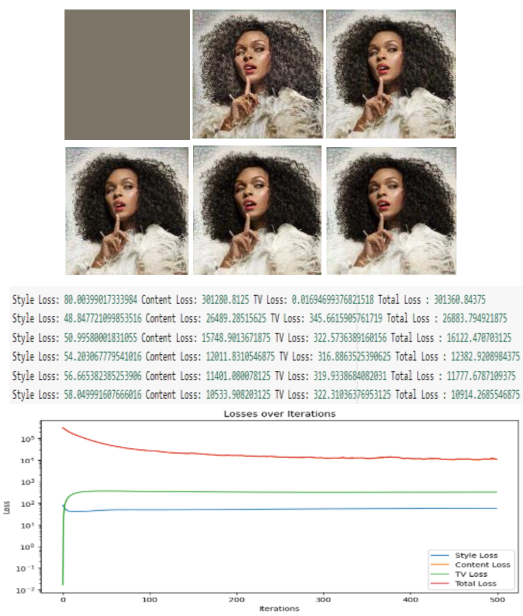


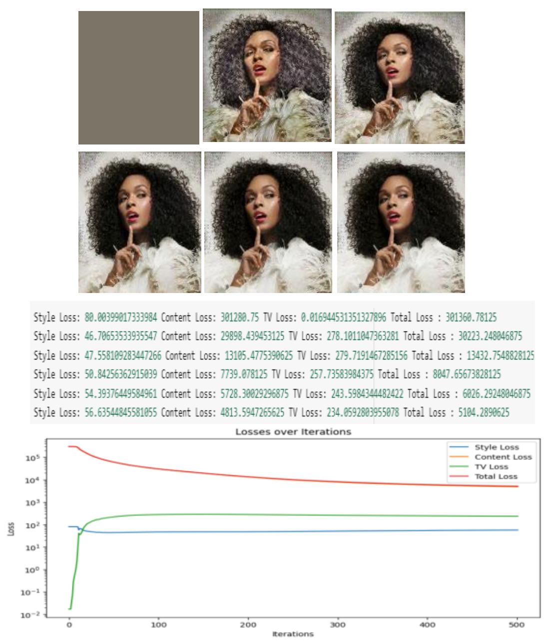
500 iteration
Pooling: max
weights:
lr=10
alpha=5
beta=1e-3
tv=1e-5

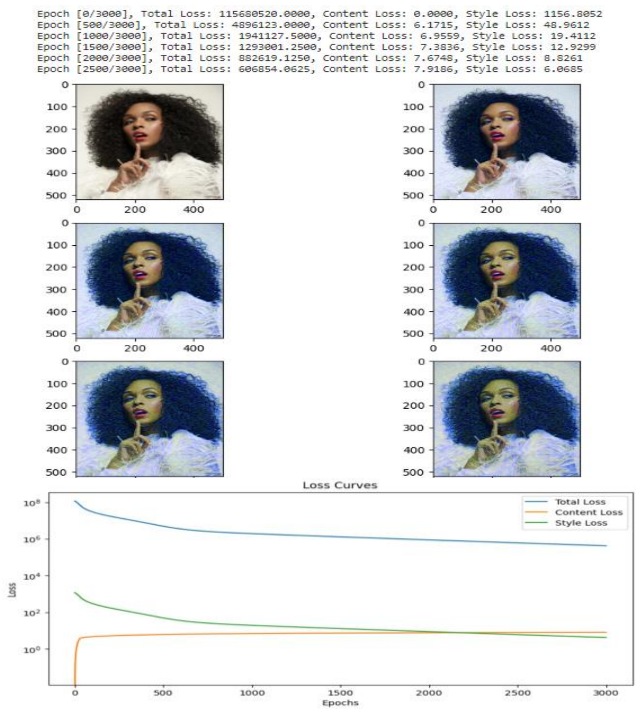
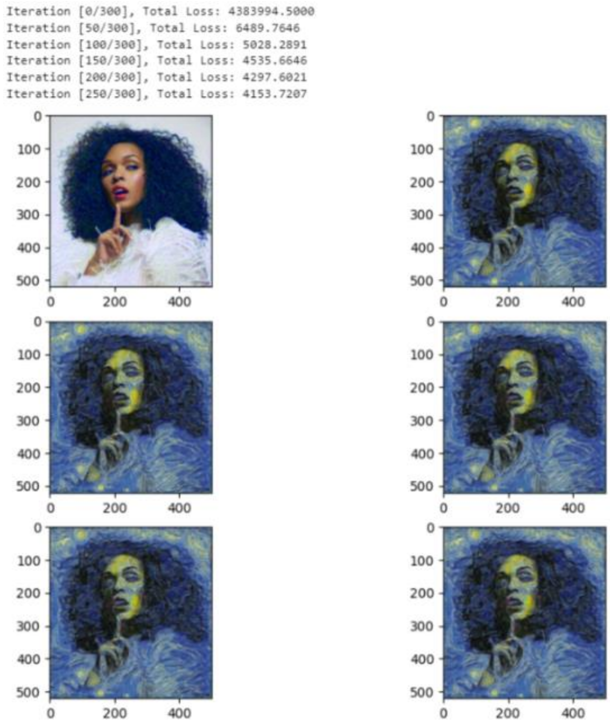
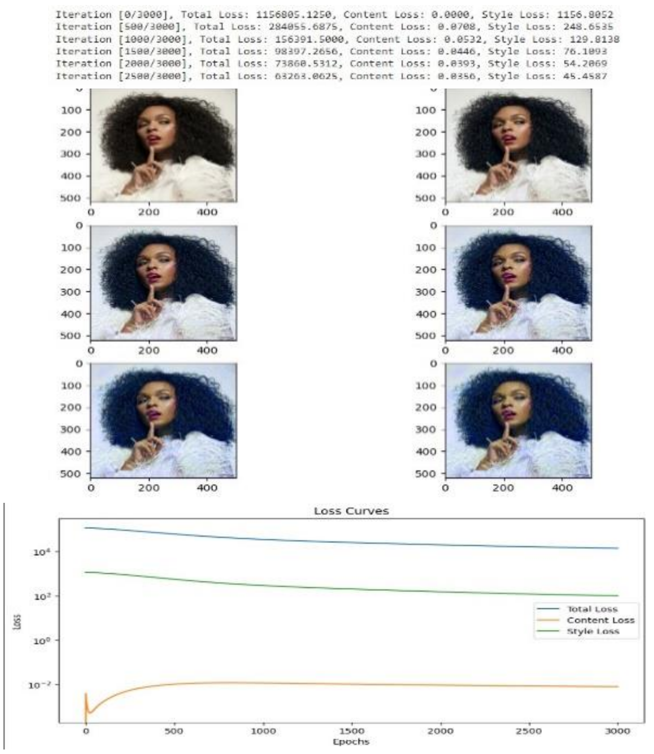
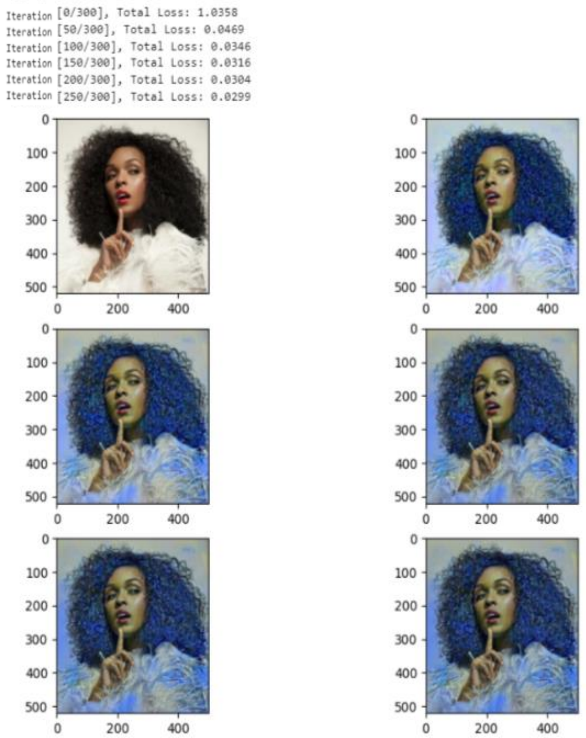




500 iteration
Pooling: avg
weights:
lr=10
alpha=5
beta=1e-3
tv=1e-5

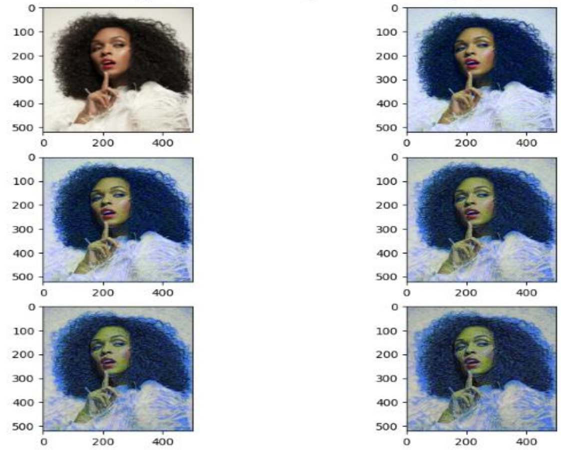


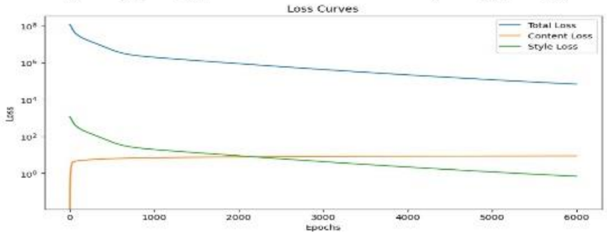


<table>
<tr><th colspan="2">CONVOLUTION LAYERS</th></tr>
<tr><th>ADAM Optimizer</th><th>LBFGS Optimizer</th></tr>
<tr><td>3000 iteration
weights: lr=0.003,alpha=1,beta=1e5

</td>
<td>300 iteration
weights: alpha=1,beta=1e5

</td></tr>
<tr><td>3000 iteration
weights: lr=0.001,alpha=5e5,beta=1e2

</td>
<td>300 iteration
weights: alpha=1,beta=0.001

</td></tr>
</table>

6000 iteration
weights: lr=0.003,alpha=1,beta=1e5

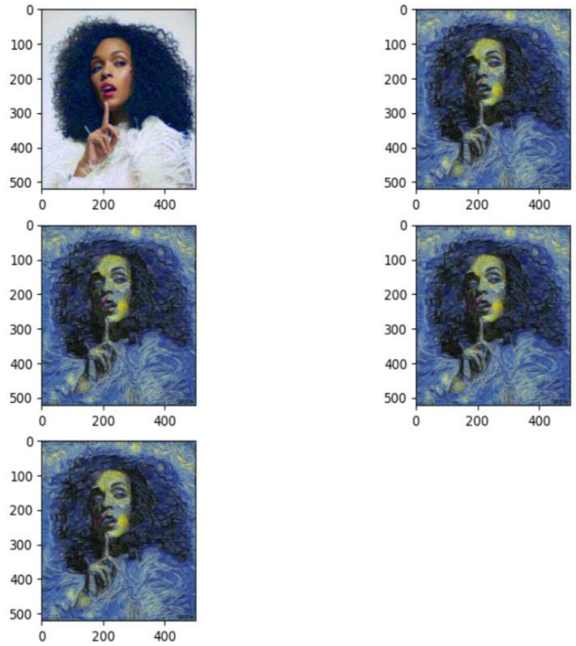
```
Iteration [0/6000], Total Loss: 115680520.0000, Content Loss: 0.0000, Style Loss: 1156.8052
Iteration [1000/6000], Total Loss: 1940983.3750, Content Loss: 6.9595, Style Loss: 19.4098
Iteration [2000/6000], Total Loss: 882441.5625, Content Loss: 7.6882, Style Loss: 8.8243
Iteration [3000/6000], Total Loss: 423097.5312, Content Loss: 8.1125, Style Loss: 4.2309
Iteration [4000/6000], Total Loss: 216157.2969, Content Loss: 8.3863, Style Loss: 2.1615
Iteration [5000/6000], Total Loss: 117174.7188, Content Loss: 8.5713, Style Loss: 1.1717
```





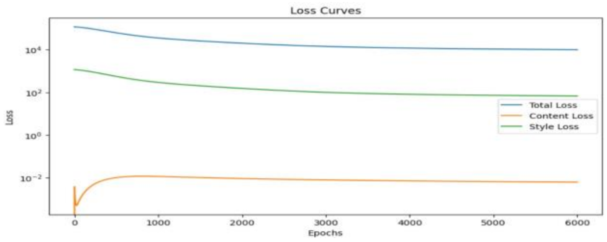
1000 iteration
weights: alpha=1,beta=1e5

```
Iteration [0/1000], Total Loss: 5308006.5000
Iteration [200/1000], Total Loss: 4262.4429
Iteration [400/1000], Total Loss: 3935.6465
Iteration [600/1000], Total Loss: 3877.1897
Iteration [800/1000], Total Loss: 3855.0149
```



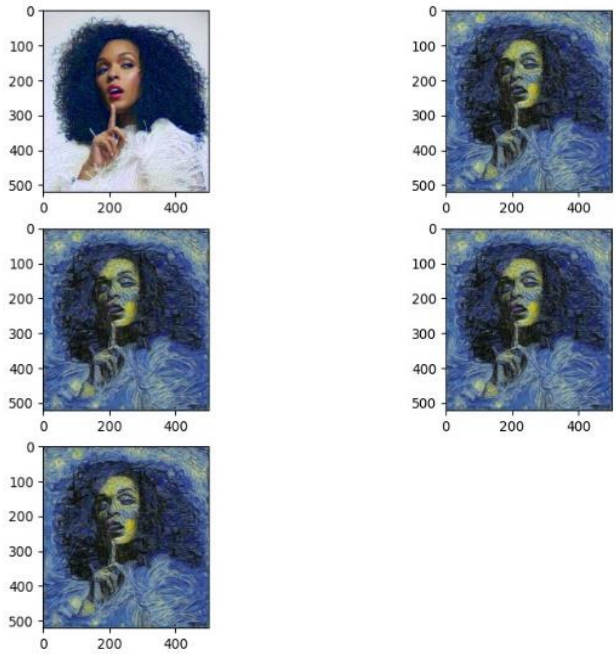
6000 iteration
weights: lr=0.001,alpha=5e5,beta=1e2

```
Iteration [0/6000], Total Loss: 115680.5156, Content Loss: 0.0000, Style Loss: 1156.8052
Iteration [1000/6000], Total Loss: 34821.6797, Content Loss: 0.0116, Style Loss: 290.1237
Iteration [2000/6000], Total Loss: 19859.5859, Content Loss: 0.0092, Style Loss: 152.3782
Iteration [3000/6000], Total Loss: 14029.5400, Content Loss: 0.0080, Style Loss: 100.3280
Iteration [4000/6000], Total Loss: 11734.6221, Content Loss: 0.0072, Style Loss: 81.2722
Iteration [5000/6000], Total Loss: 10613.2617, Content Loss: 0.0067, Style Loss: 72.6217
```



1000 iteration
weights: alpha=1,beta=5e5

```
Iteration [0/1000], Total Loss: 4368317.0000
Iteration [200/1000], Total Loss: 4292.1768
Iteration [400/1000], Total Loss: 3961.7681
Iteration [600/1000], Total Loss: 3904.2039
Iteration [800/1000], Total Loss: 3879.7854
```



<table>
<tr><th colspan="2">Vgg – 19 from Tensorflow</th></tr>
<tr><th colspan="2">SGD Optimizer</th></tr>
<tr>
<td>total_variation_weight = 1e-6
style_weight = 1e-6
content_weight = 2.5e-8
iterations = 3000
initial_learning_rate=100.0, decay_steps=100, decay_rate=0.96</td>
<td>total_variation_weight = 1e-6
style_weight = 1e-6
content_weight = 2.5e-8
iterations = 3000
initial_learning_rate=10.0, decay_steps=50, decay_rate=0.98</td>
</tr>
<tr>
<td>1: loss=8201.82
2: loss=7409.74
3: loss=7106.64
4: loss=6951.28
5: loss=6858.89
6: loss=6798.61

</td>
<td>500: loss=3757.12
1000: loss=3158.35
1500: loss=1684.57
2000: loss=1621.89
2500: loss=1581.88
3000: loss=1554.33

</td>
</tr>
</table>

The two sets of code provided are both implementations of neural style transfer, but they differ in several key aspects:

1. Libraries:
   - The first code uses PyTorch with torchvision for neural style transfer implementation.
   - The second code also uses PyTorch but implements neural style transfer without relying on torchvision. It directly handles image processing and feature extraction.

2. Model Architecture:
   - In the first code, VGG19 is used for feature extraction. It loads the pre-trained VGG19 model from torchvision.
   - The second code loads a VGG19 model but manually adjusts the pooling layers for feature extraction.

3. Loss Functions:
   - Both snippets use different loss functions. The first code snippet defines content loss and style loss functions using mean squared error (MSE) and gram matrix calculations.
   - The second code snippet uses MSE loss for content and style losses and also includes total variation (TV) loss for spatial smoothness.

4. Optimization:
   - Both the code use Adam and lbfgs optimizers, where learning rate can be adjusted.

5. Preprocessing and Postprocessing:
   - The first code snippet preprocesses images using torchvision transforms and PIL, and it includes a function for deprocessing images after style transfer.
   - The second code snippet has its own image preprocessing functions using OpenCV and torchvision transforms for converting images to tensors and vice versa.

6. Color Preservation:
   - The second code snippet includes an option for color preservation during style transfer, which is achieved through color transfer between the content and generated images.

Comparison between our results and Reference Result:

1. Style Loss:
   - For the reference results:
     - Ranges from around 142009.94 to 51000864.0.
   - For your results:
     - Ranges from around 140188.03 to 63067564.0.

Both sets of results show a significant variation in Style Loss across iterations. The Style Loss measures the difference between the style features of the style image and the generated image. A higher Style Loss indicates that the generated image is further away from the style target, which may result in less faithful style transfer.

2. Content Loss:
   - For the reference results:
     - Ranges from around 548498.38 to 1246539.0.
   - For your results:
     - Ranges from around 608578.94 to 1237537.75.

Content Loss measures the difference between the content features of the content image and the generated image. It indicates how well the content of the content image is preserved in the stylized image. Lower Content Loss values generally indicate better preservation of content.

3. TV Loss:
   - For both results:
     - Ranges from around 31419.77 to 32791.23.

TV Loss, or Total Variation Loss, encourages spatial smoothness in the generated image. It penalizes sharp transitions and encourages a more continuous appearance. The similarity in TV Loss values suggests that both sets of results maintain a similar level of spatial smoothness.

4. Total Loss:
   - For the reference results:
     - Ranges from around 740260.0 to 52247404.0.
   - For your results:
     - Ranges from around 785646.44 to 64305100.0.

Total Loss is the sum of Style Loss, Content Loss, and TV Loss. It represents the overall optimization objective, balancing style fidelity, content preservation, and image smoothness. Lower Total Loss values indicate better overall stylization performance.

Comparison:

- Style Loss Variation:
  - The reference results show a wider range of Style Loss values, indicating more variation in style fidelity across iterations compared to your results.
- Content Loss Preservation:
  - Your results generally exhibit slightly higher Content Loss values, suggesting a potential variation in content preservation compared to the reference results.
- TV Loss and Total Loss:
  - Both sets of results maintain similar TV Loss and Total Loss values, indicating comparable levels of spatial smoothness and overall optimization performance.

Overall, the choice of hyperparameters, optimizer, and the inherent randomness in the optimization process can lead to variations in the stylized images and the associated losses. It's essential to experiment with different settings and evaluate the results based on the desired style transfer quality and content preservation.