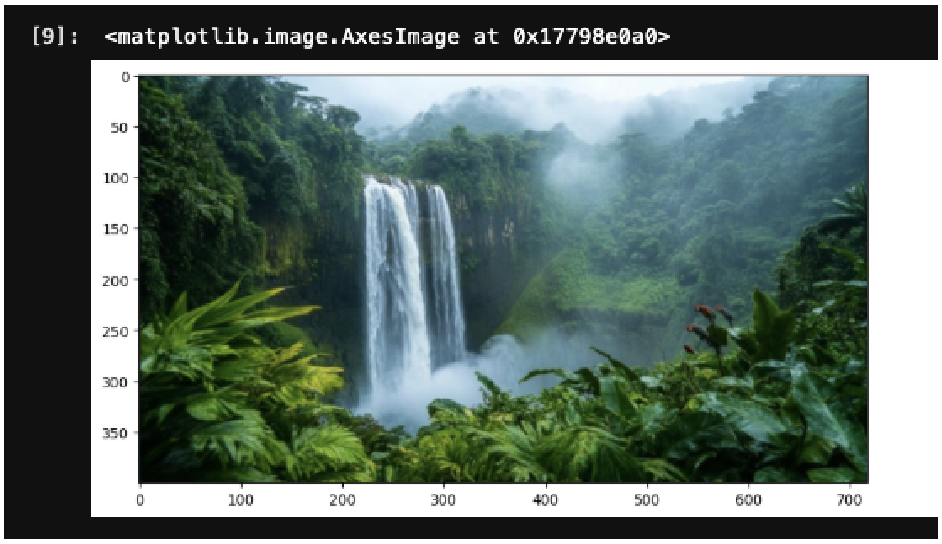
ASSIGNMENT 4: Style Transfer with Deep Neural Networks Report

Author: Shambhavi Danayak
Professor: Bo Shen
Student ID: 012654513
Submission date: 03/26/2025

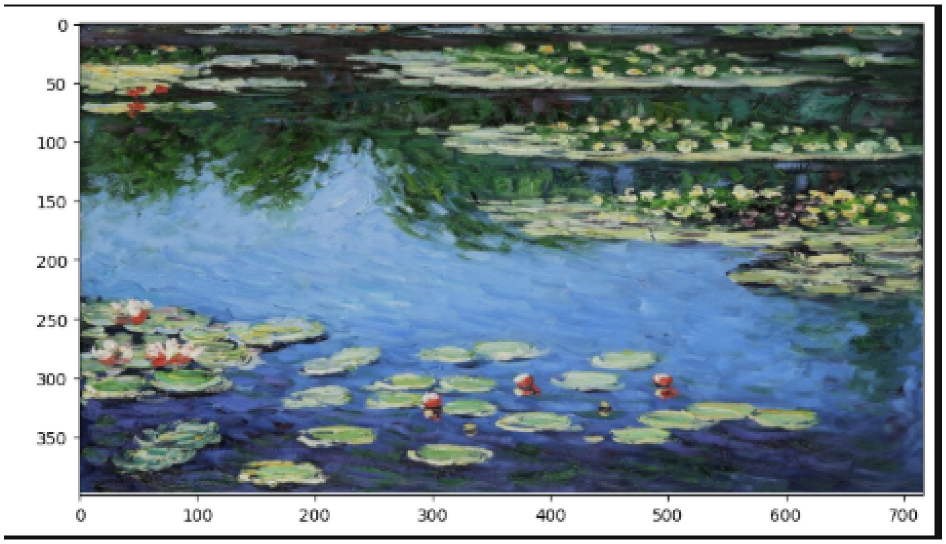
This assignment is based on using a pre-trained VGG19 Net model to extract content or style features from a passed in image.
For the assignment these are the images i will be using ,

CONTENT IMAGE:



```
[9]:  <matplotlib.image.AxesImage at 0x17798e0a0>
```

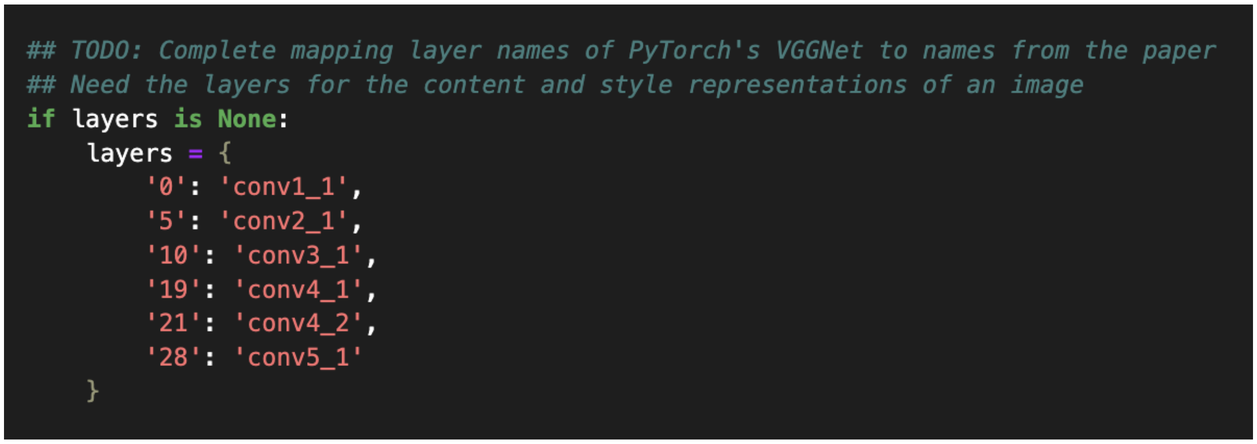
STYLE IMAGE:



TODO: complete the mapping of layer names to the names found in the paper for the content representation and the style representation.

Paper: [Image Style Transfer Using Convolutional Neural Networks]

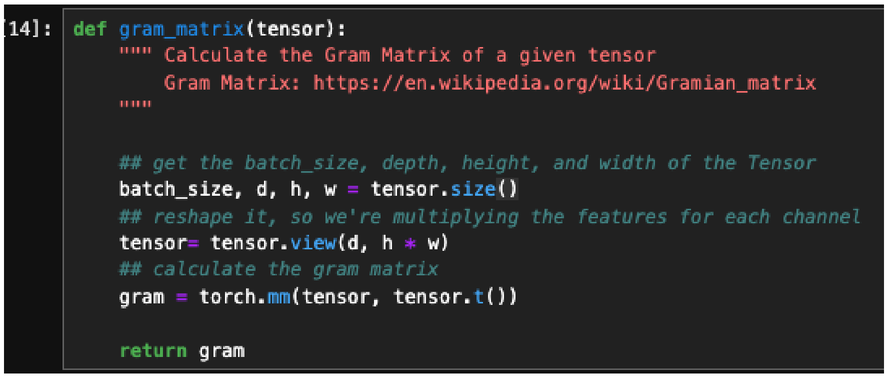
After reading the paper, in the section "3. Results" it mentions, "The images shown in Fig 3 were synthesised by matching the content representation on layer 'conv4 2' and the style representation on layers 'conv1 1', 'conv2 1', 'conv3 1', 'conv4 1' and 'conv5 1' (wl = 1/5 in those layers, wl = 0 in all other layers) . "



```
## TODO: Complete mapping layer names of PyTorch's VGGNet to names from the paper
## Need the layers for the content and style representations of an image
if layers is None:
    layers = {
        '0': 'conv1_1',
        '5': 'conv2_1',
        '10': 'conv3_1',
        '19': 'conv4_1',
        '21': 'conv4_2',
        '28': 'conv5_1'
    }
```

EXPLANATION OF MAPPING: Pytorch uses numeric strings (eg. '0', '1', '2' …) in its sequential model therefore mapping from these indices to the layer names used in the paper. These specific layers correspond to the desired convolutional blocks.
Style layers: conv1_1, conv2_1, conv3_1, conv4_1, conv5_1
Content layer: Conv4_2

TODO: Complete the gram_matrix function.



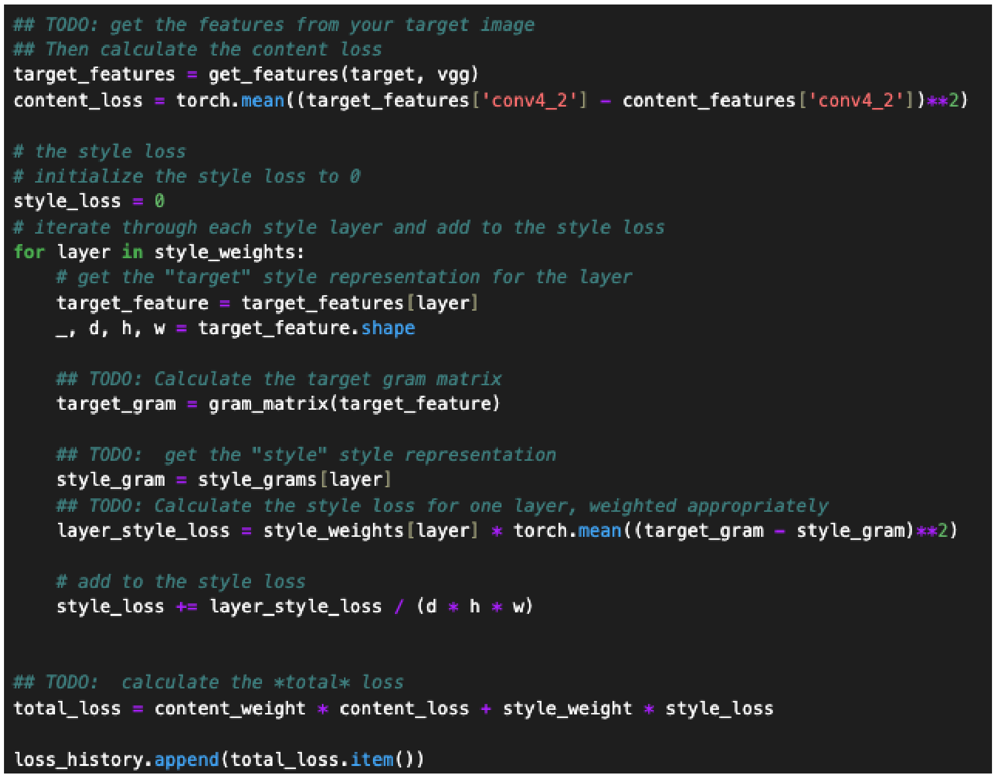
```
[14]: def gram_matrix(tensor):
          """ Calculate the Gram Matrix of a given tensor
              Gram Matrix: https://en.wikipedia.org/wiki/Gramian_matrix
          """

          ## get the batch_size, depth, height, and width of the Tensor
          batch_size, d, h, w = tensor.size()
          ## reshape it, so we're multiplying the features for each channel
          tensor= tensor.view(d, h * w)
          ## calculate the gram matrix
          gram = torch.mm(tensor, tensor.t())

          return gram
```

EXPLANATION: Gram Matrix captures the correlations between feature maps and is essential for computing the style loss in neural style transfer. First we extract the dimensions of the input tensor, then we reshape the tensor to prepare for gram matrix calculation and finally calculate the gram matrix.

TODO: Define content, style, and total losses.



```
## TODO: get the features from your target image
## Then calculate the content loss
target_features = get_features(target, vgg)
content_loss = torch.mean((target_features['conv4_2'] - content_features['conv4_2'])**2)

# the style loss
# initialize the style loss to 0
style_loss = 0
# iterate through each style layer and add to the style loss
for layer in style_weights:
    # get the "target" style representation for the layer
    target_feature = target_features[layer]
    _, d, h, w = target_feature.shape

    ## TODO: Calculate the target gram matrix
    target_gram = gram_matrix(target_feature)

    ## TODO:  get the "style" style representation
    style_gram = style_grams[layer]
    ## TODO: Calculate the style loss for one layer, weighted appropriately
    layer_style_loss = style_weights[layer] * torch.mean((target_gram - style_gram)**2)

    # add to the style loss
    style_loss += layer_style_loss / (d * h * w)


## TODO:  calculate the *total* loss
total_loss = content_weight * content_loss + style_weight * style_loss

loss_history.append(total_loss.item())
```

Content loss: computed as the mean squared error between the target and content features at layer conv4_2. Mathematically,

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

where,

- $\mathbf{F_{ij}}^{\mathbf{target}}$ = Activation of the target image, feature map i, position j
- $\mathbf{F_{ij}}^{\mathbf{content}}$ = Activation of the content image, feature map i, position j

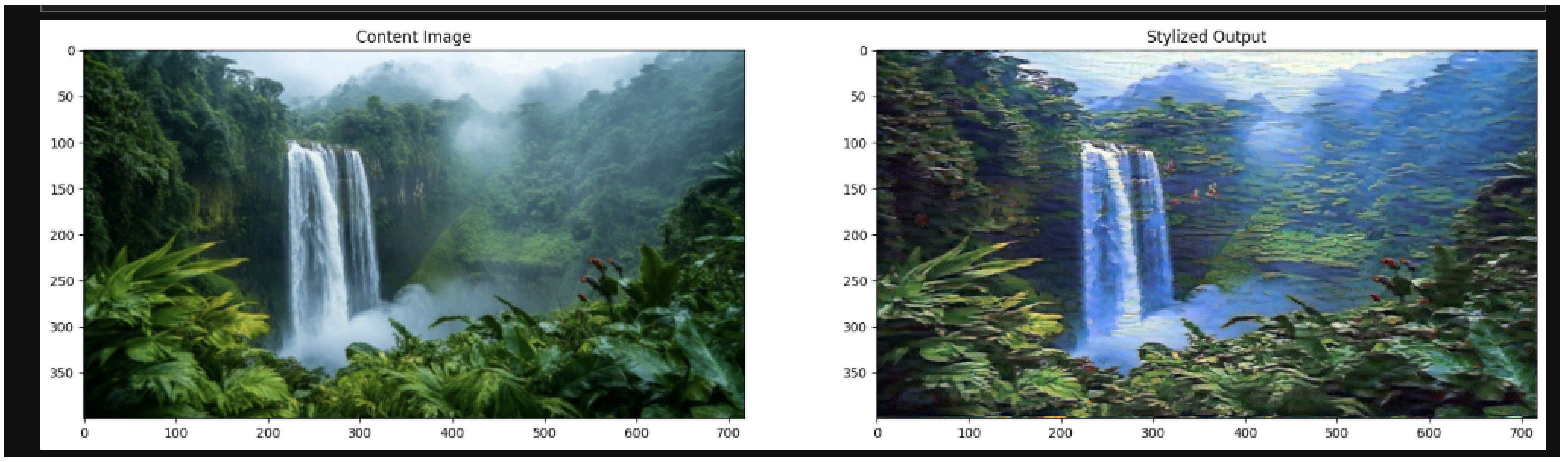
Style Loss: is calculated layer by layer using specified style_weights. Mathematically total style loss,

$$\mathcal{L}_{\text{style}}(\vec{a},\vec{x}) = \sum_{l=0}^{L} w_l E_l,$$

- a= The style image
- x= target image
- L= Total loss
- w $_l$= weight for layer l
- E_L= style loss for layer l given by

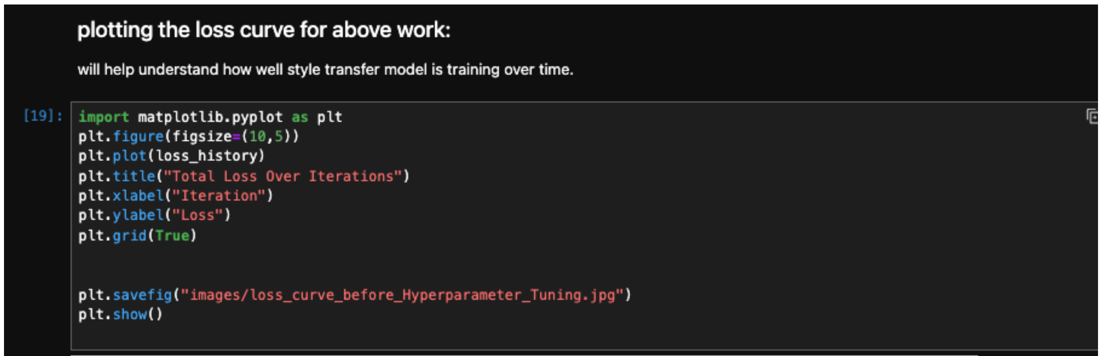
$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^l - A_{ij}^l\right)^2$$

CONTENT AND FINAL TARGET IMAGE,



LOSS CURVE,

Please turn over

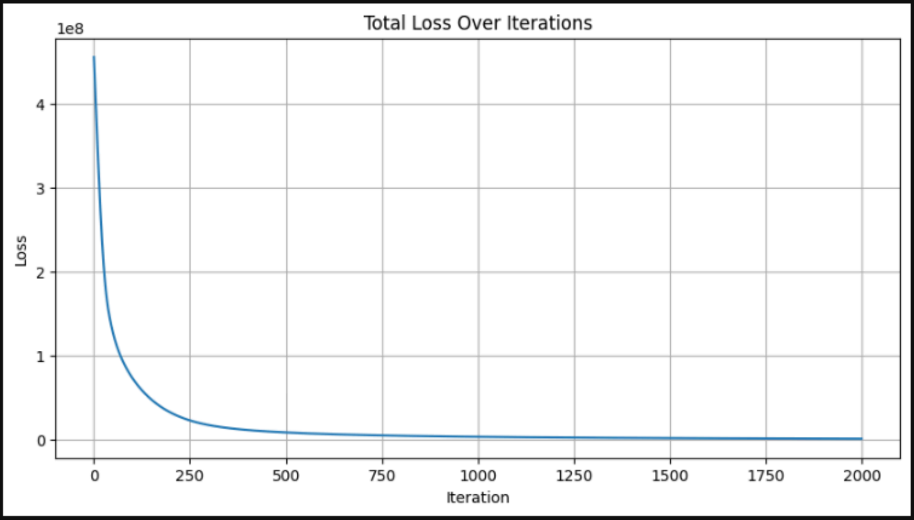


plotting the loss curve for above work:

will help understand how well style transfer model is training over time.

```
[19]: import matplotlib.pyplot as plt
plt.figure(figsize=(10,5))
plt.plot(loss_history)
plt.title("Total Loss Over Iterations")
plt.xlabel("Iteration")
plt.ylabel("Loss")
plt.grid(True)


plt.savefig("images/loss_curve_before_Hyperparameter_Tuning.jpg")
plt.show()
```



INFERENCE:

- Total loss starts with almost $5x10^6$ hinting that initially there is a big mismatch between the generated and target features.
- It gradually decreases over the steps. Most of the reduction in the loss value is happening before 750 steps and then flattens until 2000 steps. Therefore we can conclude the model is learning to match style and content.

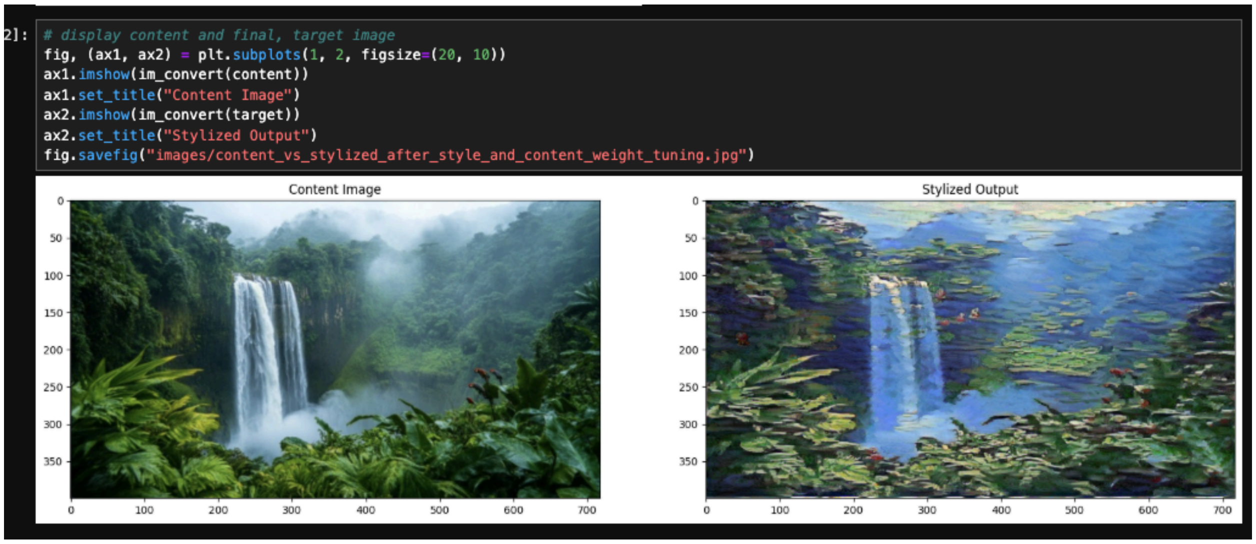
Part 2: play with as many hyper parameter settings as you can to experiment and compare the outcome

a) <u>Tuning content_weight and style weight</u>

Theoretically changing content_weight to a higher value will emphasize preserving the structure and detail of the original content image more thereby making the styled output retain more of the original layout and shapes.

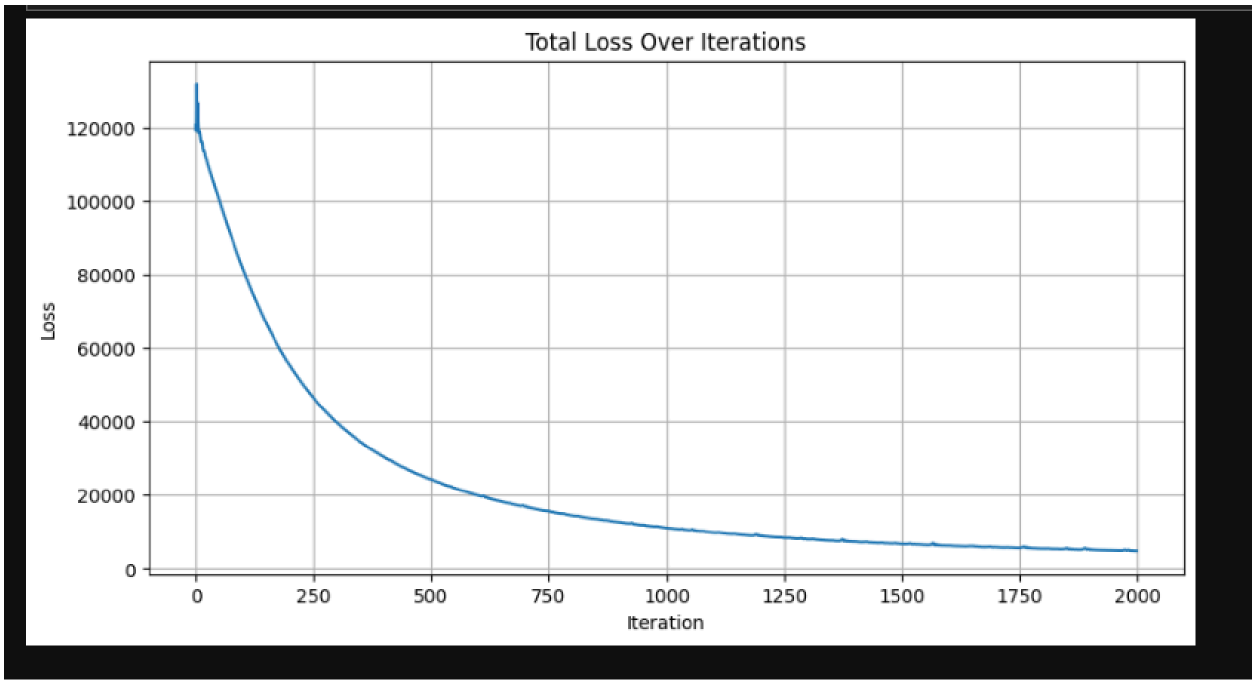
Tuning style_weight to a lower weight value will decrease the influence of style features, resulting in more subtle texture and color adaptations from the style image rather than strong artistic transformations.
Here ***style_weight was lowered*** to a value of 1x10^5 and ***content_weight was increased*** to a value of 10.

CONTENT AND FINAL TARGET IMAGE,



```
2]: # display content and final, target image
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 10))
    ax1.imshow(im_convert(content))
    ax1.set_title("Content Image")
    ax2.imshow(im_convert(target))
    ax2.set_title("Stylized Output")
    fig.savefig("images/content_vs_stylized_after_style_and_content_weight_tuning.jpg")
```

LOSS CURVE,



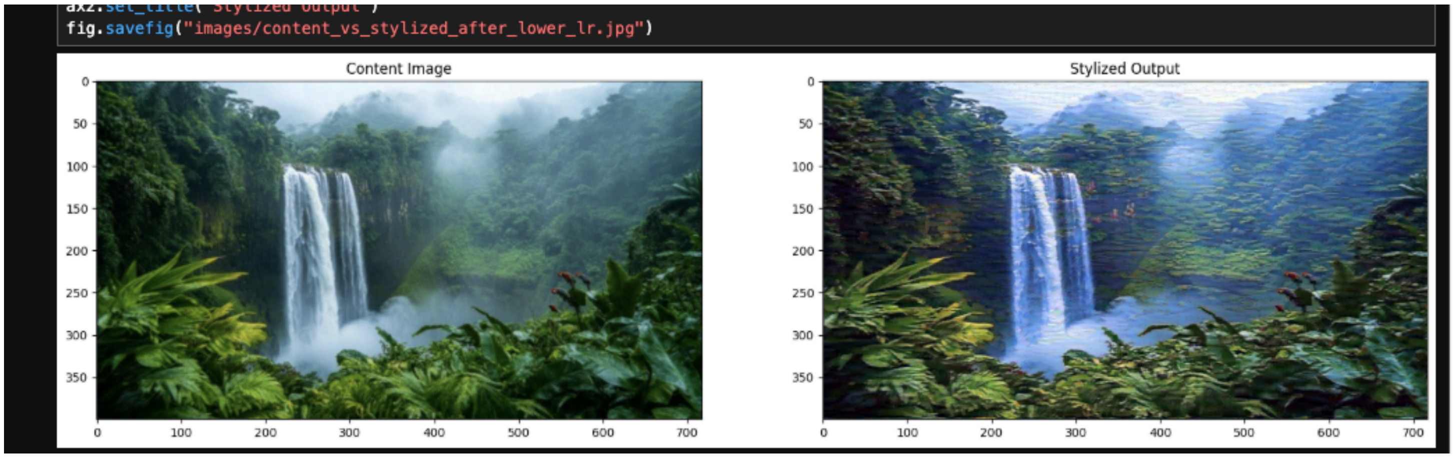
INFERENCE:

- We can clearly see from the generated target image that the original structure of the content image was retained more.
- The loss curve shows that the loss starts with a little more than 120000 and gradually decreases until 2000 steps are completed hinting at the fact that not a lot of mismatch between the generated and target features.

b) <u>Tuning learning Rate</u>

Lowering the learning rate from 0.003 to 0.001 slows down the optimization process, causing the model to make smaller updates to the target image at each iteration. Slower learning can lead to less distorted and smoother, as the style blending happens more carefully over time.

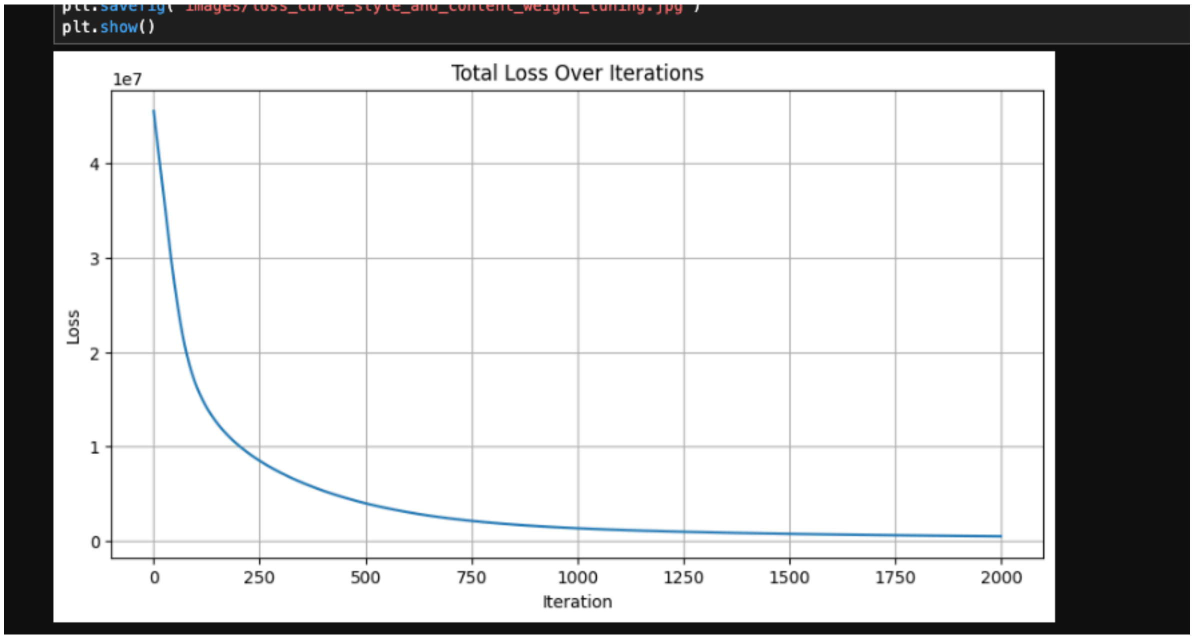
CONTENT AND FINAL TARGET IMAGE,



```
ax2.set_title("Stylized Output")
fig.savefig("images/content_vs_stylized_after_lower_lr.jpg")
```

LOSS CURVE,

Please turn over



```
plt.savefig('images/loss_curve_style_and_content_weight_tuning.jpg')
plt.show()
```

INFERENCE:

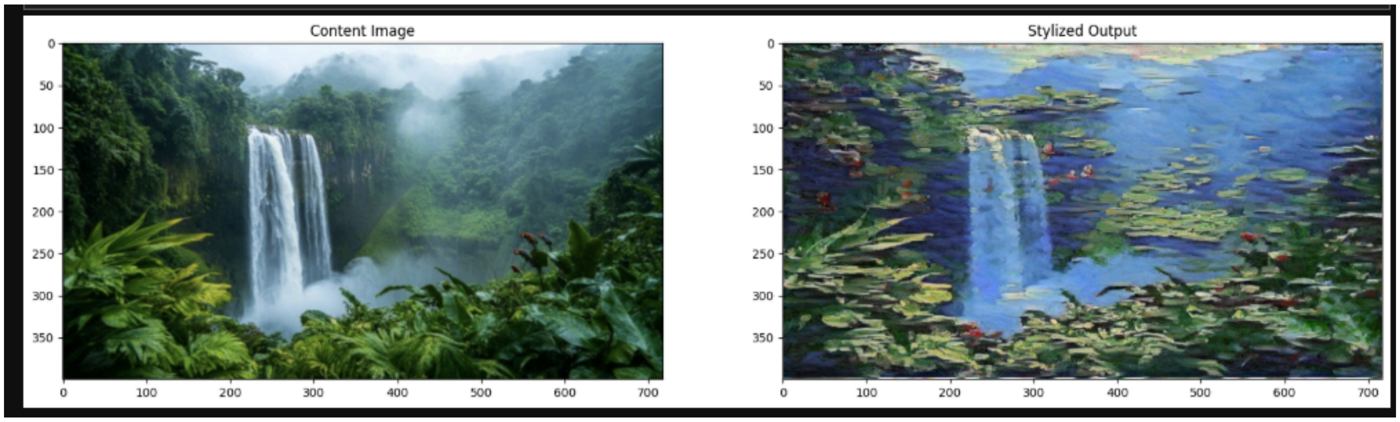
- The total loss starts at a higher value and decreases gradually. The curve is a smoother and more stable decline indicating a slower and controlled learning.
- From the generated target image we can see that lower learning rate led to finer adjustments i.e it preserved more of the original content image details while steadily integrating style features.
- Final target image looks cleaner and visually coherent, style did not overpower the image.

c) Tuning number of steps/iterations

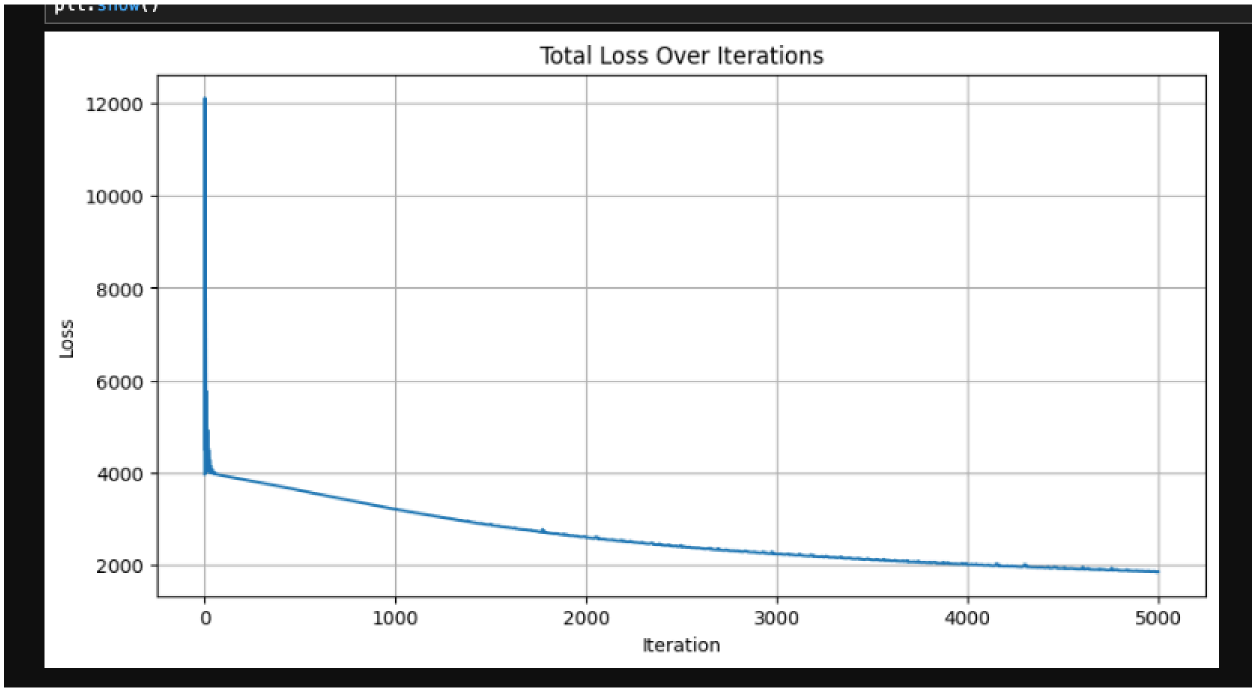
Increasing number of steps/iterations to a value of 5000.
This means the model will take more time to optimize the loss leading to a more emphasised style texture in the target image. Too many steps can also overfit the style, leading to a target image losing content structure.

CONTENT AND FINAL TARGET IMAGE,

Please turn over



LOSS CURVE,



INFERENCE:

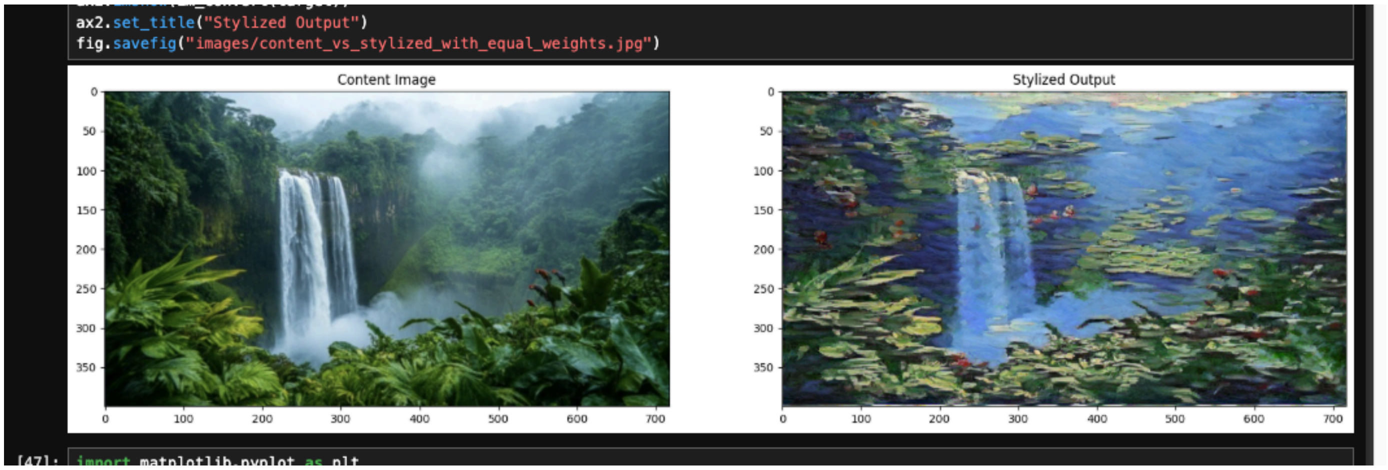
- Increasing number of steps to 5000 increased execution time.
- 5000 steps allowed the model more time to optimize the target image, resulting in stronger and more detailed transfer of style.
- The loss curve shows a sharp dip in loss by ~2000 steps. This indicated rapid learning.
- After 3000 steps the curve flattens, indicating that the model is converging and next improvements are gradual.
- Visually the target image looks richer in style texture, brush stroke etc. but also the details of the content image are slightly overpowered by style i.e. the original structure of the content image seems to not have been retained.

- While more steps helps in deepening style application, the trade-off is the loss of content image structure.

d) Tuning Style_weights dictionary

Each layer like conv1_1, conv2_1, conv3_1 etc. in the VGG19 network and the value of weight affects how the target image is generated. Early layers (conv1_1, conv2_1 etc.) capture fine textures, brush strokes, color patterns whereas Deeper layers such as conv4_1, conv5_1 capture broader patterns, structure, and overall composition.
Here I try and give each layer equal weighting of 1.0 to balance all style equally.
NOTE: I will reduce the number of steps to 1000 steps for faster execution time.

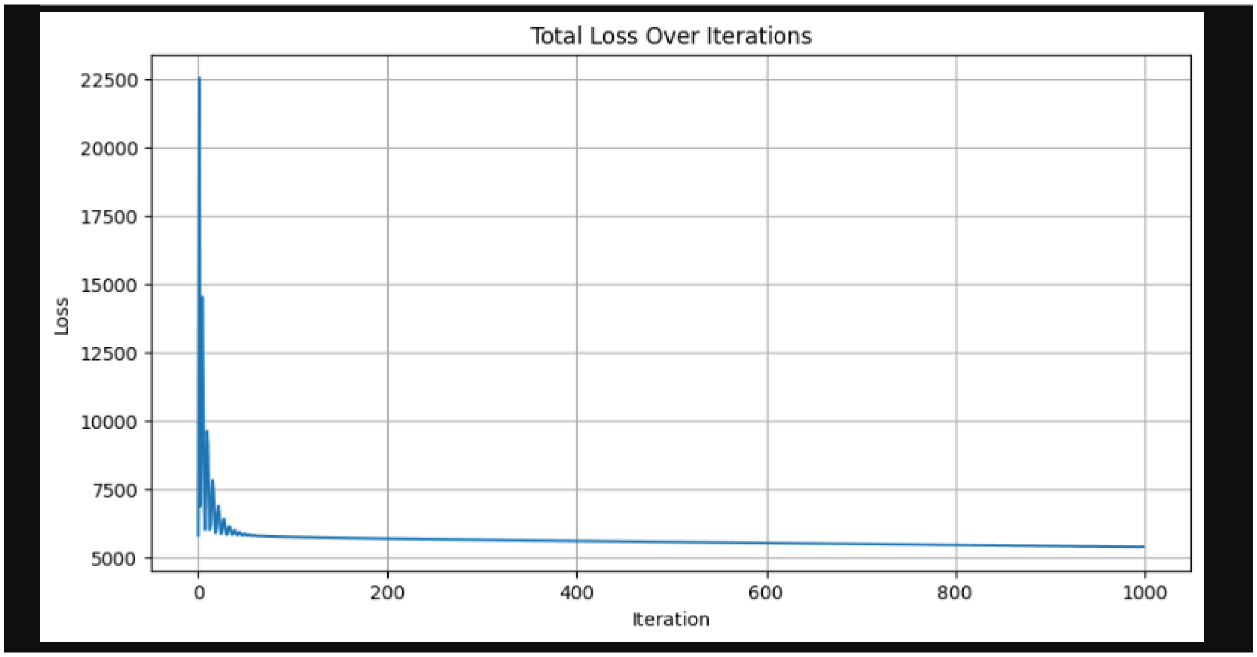
CONTENT AND FINAL TARGET IMAGE,



```
ax2.set_title("Stylized Output")
fig.savefig("images/content_vs_stylized_with_equal_weights.jpg")
```

LOSS CURVE,

Please turn over



INFERENCE:

- Equal weights (1.0) to all style layers ensures balanced contribution from both shallow and deeper layers, resulting in an even application of style features across fine textures and broader patterns.
- The loss curve shows a sharp decline within the first ~150 steps, indicating rapid optimization, followed by a stable plateau, suggesting that the model has effectively converged.
- Visually the target image has strong style texture, strokes etc.
- Overall, this setup produced a visually strong styled image that is well-balanced but is not retaining much of content structure.

Thank you!