23CSAI08H

Deep Learning

Deep learning CW2: Neural Style Transfer

By:

ID:	Name:	Contribution:
211320	Mohammed Amein	Data preprocessing + VGG19
205001	Soha Ashraf	Data preprocessing + VGG16

Introduction:

Neural Style Transfer (NST) is a technique which transfers one of two images into the other. Combing two images which are content image and style image. Extracting the style from style image and applying it on the content image as it used as object to change the content image appearance. In this project, Neural Style Transfer (NST) is a model that can modify the appearance of the image style. The NST consists of two parts which are content image and style image.[1]. The used architectures for this model are mostly CNN and GANs with their different types. The two used architectures are Convolutional Neural Network (CNN) and Generative adversarial networks (GAN). GAN is a type of neural network that consists of a generator and discriminator. The discriminator is used for distinguishing between the real image and the generated image. The generator is used for generating a new image that is similar to the training images. [2]. There are several advancements in NST and GAN one of them is CycleGANs that has the capabilities, challenges, and limitations of NST.[3]. Neural style transfer employs a pre-trained Convolutional Neural Network for feature extraction and separation of content and style representations from an image. Neural style transfer network has **two inputs:** Content image and Style image. The content image is recreated as a newly generated image which is the only trainable variable in the neural network. The architecture of the model performs the training using two **loss terms**: Content Loss and Style Loss. **Content Loss:** by applying the mean square MSE difference between matrices generated by the content layer, when passing the generated image and the original image. **Style Loss:** calculating the gram matrix. The gram matrices calculation involves calculating the inner product between the vectorized feature maps of a particular layer. The Gram matrix represents the inner product of each vector and its corresponding vectors within the same matrix. Its significance in contemporary machine learning stems from applications in deep learning loss, particularly in the computation of loss functions during style transfer, which relies on the gram matrix.

Dataset:

- **Datasets:**

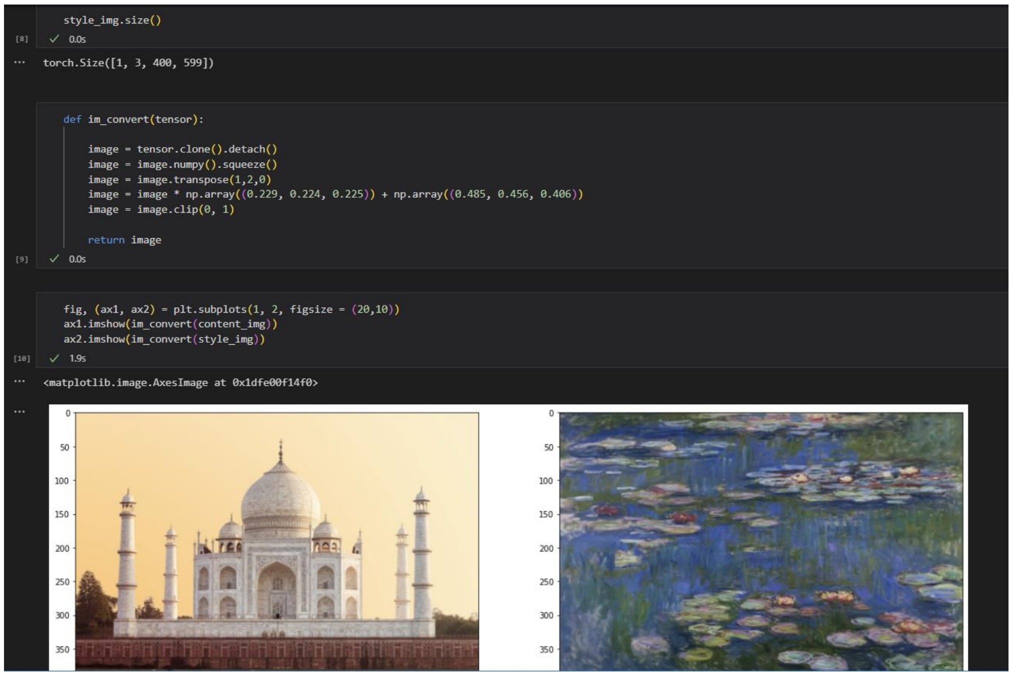
Utilizing the COCO/WikiArt NST dataset from kaggle, which encompasses paintings from 195 diverse artists. This comprehensive dataset comprises 42,129 training images and 10,628 images for testing purposes. Dataset for Neural Style Transfer consisting of COCO2017 images and Kaggle competition "Painter by Numbers" dataset.

Dataset: https://www.kaggle.com/datasets/shaorrran/coco-wikiart-nst-dataset-512-100000/data

Due to the huge size of the dataset, we could not train it on our models. So, we chose some images to train the models on them.

Data Preprocessing:

Resize all the used images to make them the same size and reduce the big size of image. Transforming images into tensors.



```
style_img.size()
```

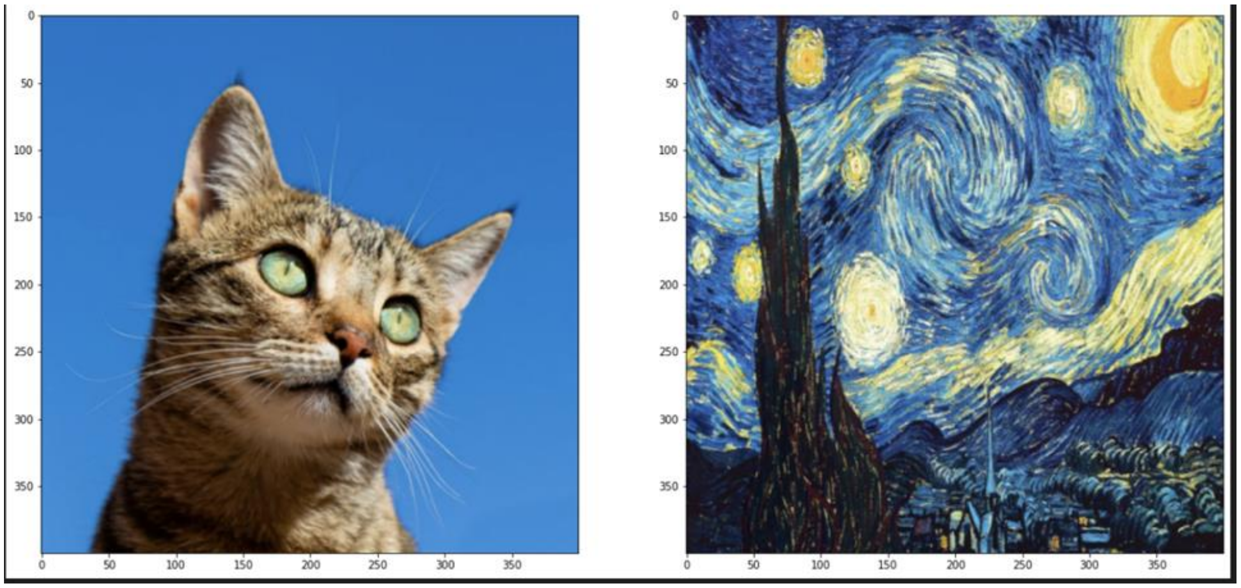
```
torch.Size([1, 3, 400, 599])
```

```
def im_convert(tensor):

    image = tensor.clone().detach()
    image = image.numpy().squeeze()
    image = image.transpose(1,2,0)
    image = image * np.array((0.229, 0.224, 0.225)) + np.array((0.485, 0.456, 0.406))
    image = image.clip(0, 1)

    return image
```

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (20,10))
ax1.imshow(im_convert(content_img))
ax2.imshow(im_convert(style_img))
```

```
<matplotlib.image.AxesImage at 0x1dfe00f14f0>
```

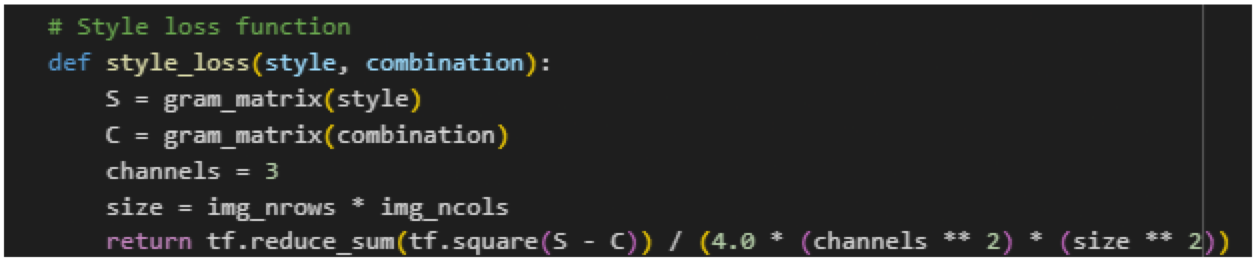
Another example:



Loss function:

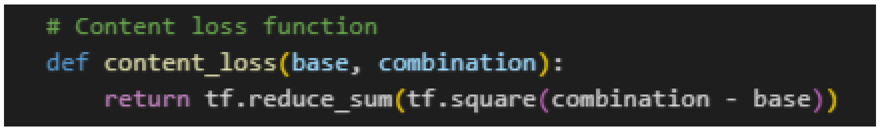
VGG the loss function was done by combining both style loss and content loss

Style loss to ensure the model follows the given style



```python
# Style loss function
def style_loss(style, combination):
    S = gram_matrix(style)
    C = gram_matrix(combination)
    channels = 3
    size = img_nrows * img_ncols
    return tf.reduce_sum(tf.square(S - C)) / (4.0 * (channels ** 2) * (size ** 2))
```

Content loss to ensure the models saves the original image structure



```python
# Content loss function
def content_loss(base, combination):
    return tf.reduce_sum(tf.square(combination - base))
```

Optimization function

stochastic gradient descent (SGD) with an exponential decay learning rate schedule learning rate starts at 100.0 and decays exponentially with a decay rate of 0.96 every 100 steps.

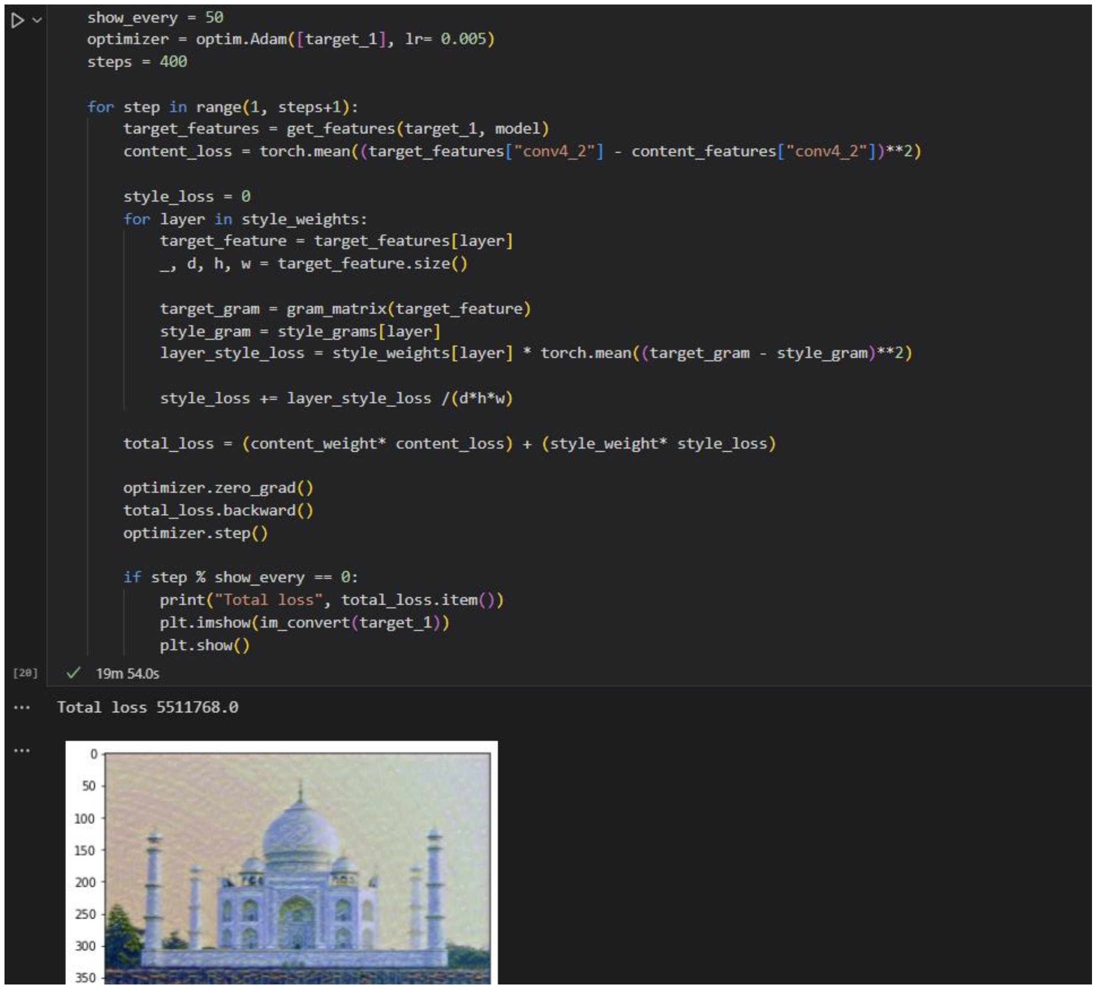
The used model:

Convolution Neural Networks - CNNs:

- VGG 16.
- VGG 19.

- VGG 16:

The VGG 16 model has 16 layers, 3 fully connected layers and 13 convolutional layers. It contains max pooling and convolution layers. For extracting style images features to apply it on content images' object, the images should go through the model and extract the features from the style images from each block conv1_1, conv2_1 in the first layer, then extracting the content images features from the second layer. Results after calculating the total loss of content loss and style loss:

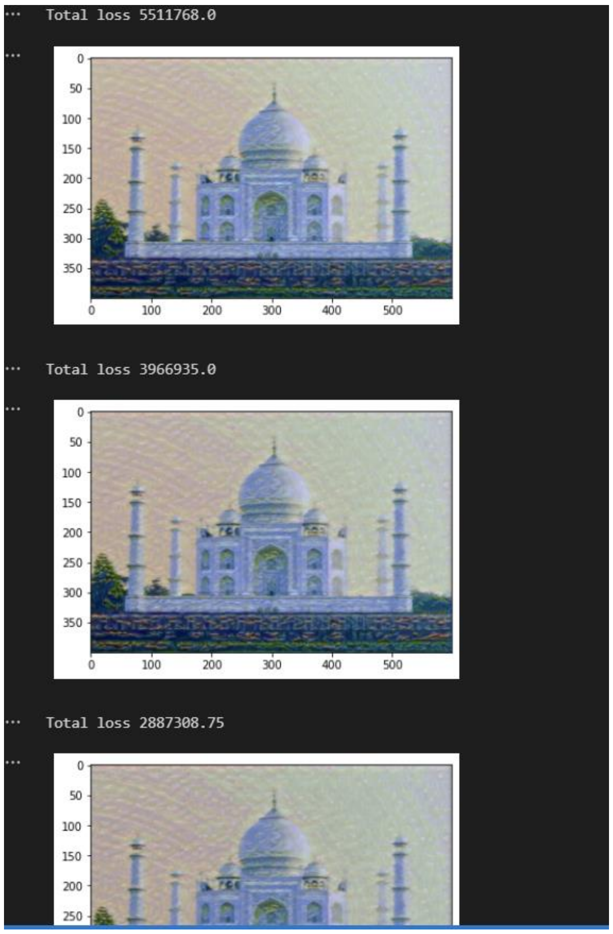


```
show_every = 50
optimizer = optim.Adam([target_1], lr= 0.005)
steps = 400

for step in range(1, steps+1):
    target_features = get_features(target_1, model)
    content_loss = torch.mean((target_features["conv4_2"] - content_features["conv4_2"])**2)

    style_loss = 0
    for layer in style_weights:
        target_feature = target_features[layer]
        _, d, h, w = target_feature.size()

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

        style_loss += layer_style_loss /(d*h*w)

    total_loss = (content_weight* content_loss) + (style_weight* style_loss)

    optimizer.zero_grad()
    total_loss.backward()
    optimizer.step()

    if step % show_every == 0:
        print("Total loss", total_loss.item())
        plt.imshow(im_convert(target_1))
        plt.show()
```

[20] ✓ 19m 54.0s

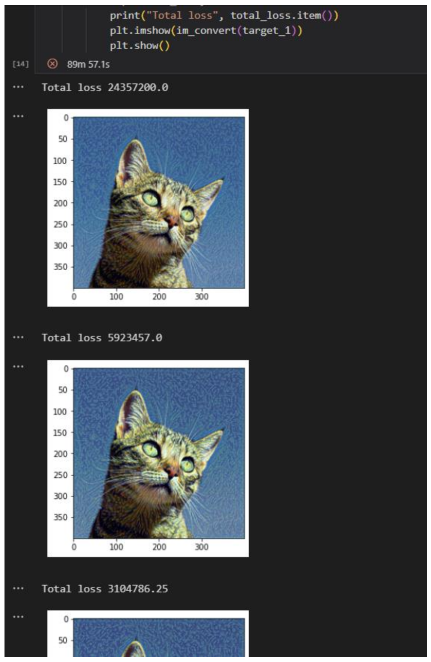
```
Total loss 5511768.0
```

Examples of the loss output (show_every = 50, steps = 400 = 7 iteration):



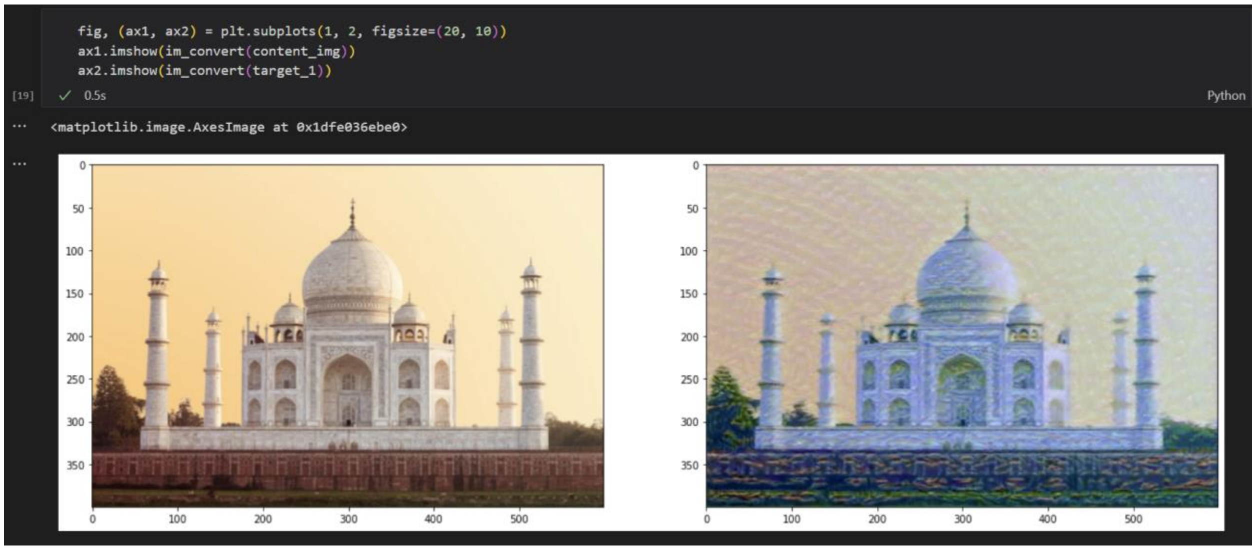
```
Total loss 5511768.0

Total loss 3966935.0

Total loss 2887308.75
```

Another one on different image (show_every = 400, steps = 6000 = 15 iterations, but it was interrupted at 5 iteration):



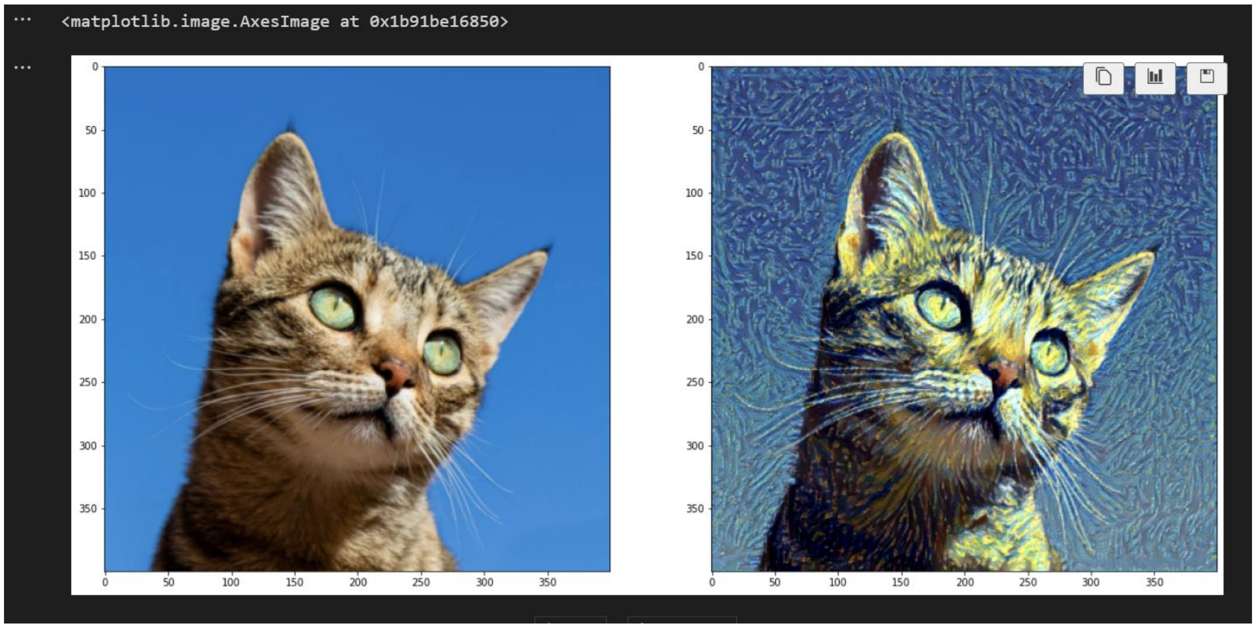
```
            print("Total loss", total_loss.item())
            plt.imshow(im_convert(target_1))
            plt.show()
[14]   89m 57.1s

Total loss 24357200.0

Total loss 5923457.0

Total loss 3104786.25
```

Result:



```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 10))
ax1.imshow(im_convert(content_img))
ax2.imshow(im_convert(target_1))
```

```
<matplotlib.image.AxesImage at 0x1dfe036ebe0>
```



```
<matplotlib.image.AxesImage at 0x1b91be16850>
```

VGG 19:

The VGG19 model is an extension of VGG16, containing 19 layers in total, with 16 convolutional layers and 3 fully connected layers. it employs max pooling and convolution layers throughout. When using VGG19 for style transfer tasks, the process involves passing both style and content images through the model to extract their respective features. The style image features are typically extracted from earlier convolutional layers, while the content image features are extracted from deeper layers. The results are achieved by calculating the total loss, which encompasses both the content loss and the style loss, to produce the stylized output.

Result:

For vgg19 we tried one style image on 3 different images

Style image :

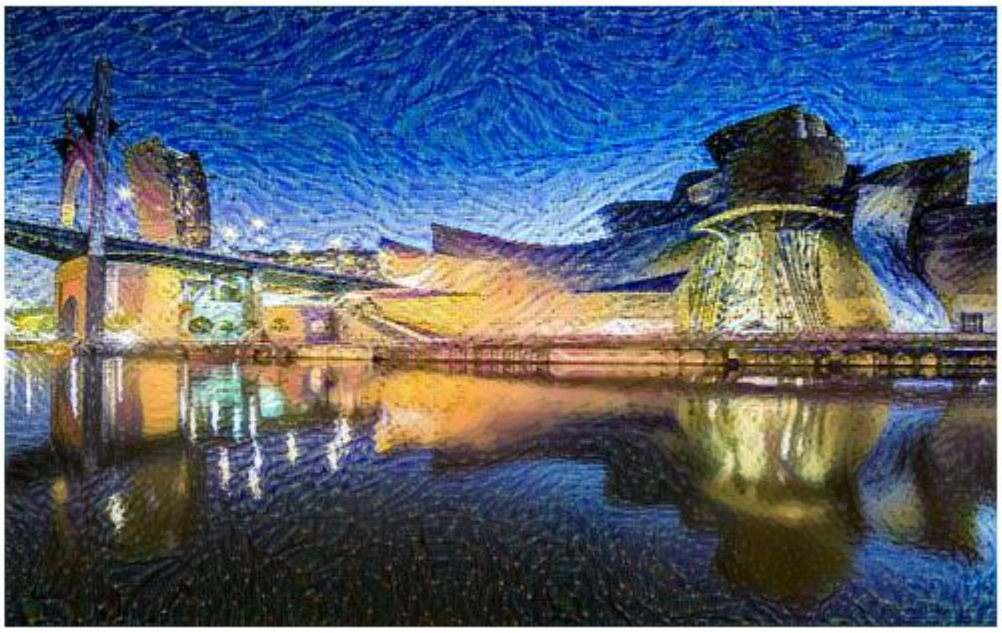
Content image 1:

Content image 2:

Content image 3:

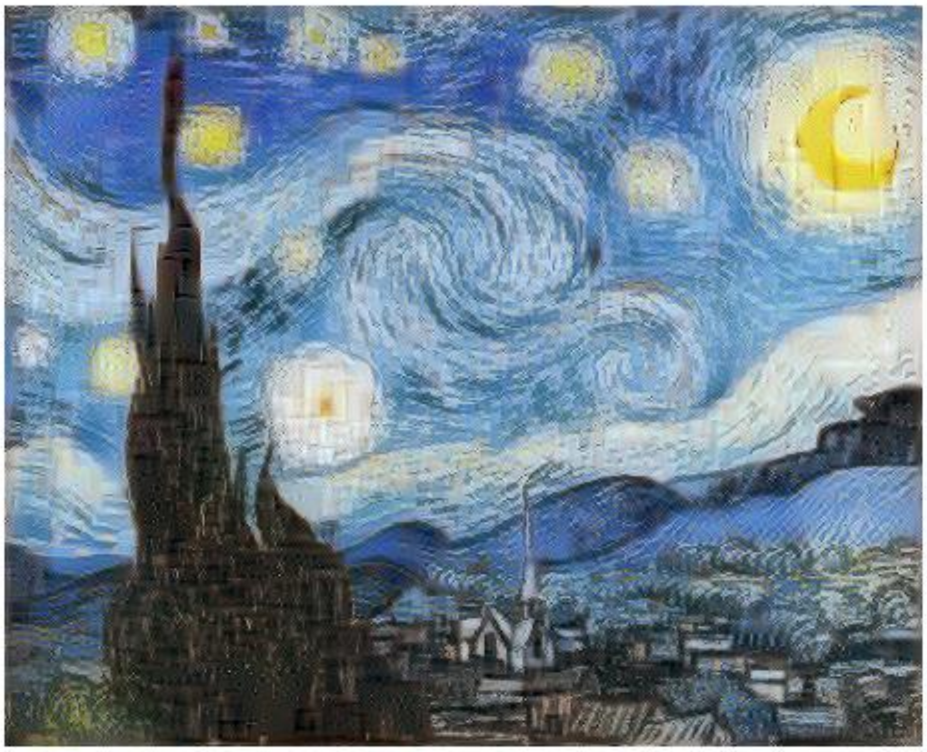
Content 1 result:

After 4000 epoch the loss was 284.37



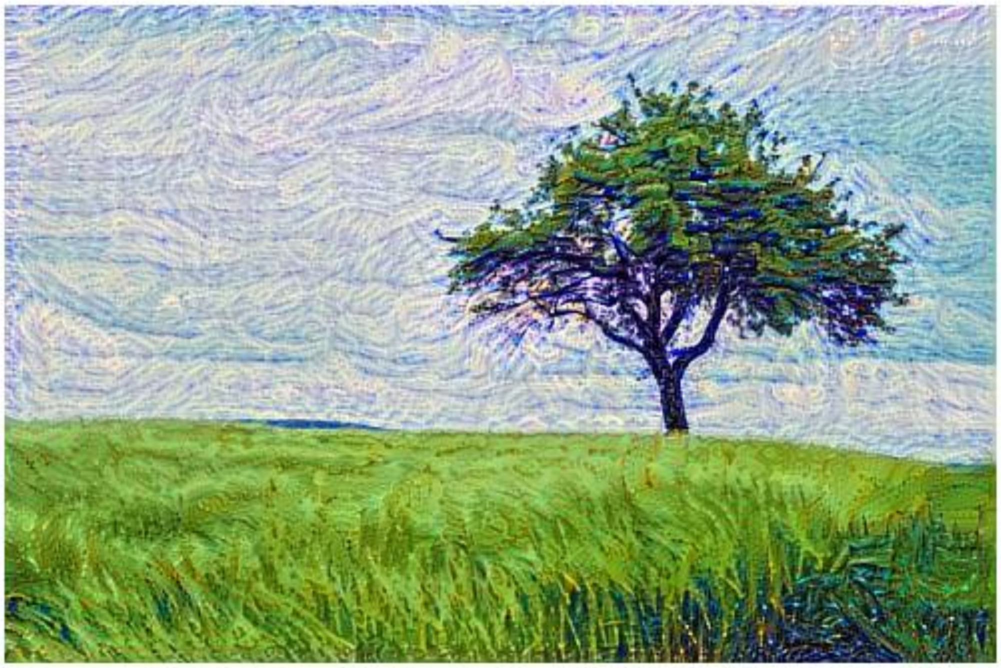
Content 2 result

After 90 epoch the loss was 3863.55



Content 3 result

After 440 epoch the loss was 1313.13



Metrics for evaluation:

Structural Similarity Index (SSIM): it is a perceptual metric that evaluates the two images structural similarity and refers it to an index by integrating the image quality.

Peak Signal-to-Noise Ratio (PSNR): it is the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation.

References:

[1] P. Baheti, “Neural Style Transfer: Everything You Need to know [Guide],” V7, Apr. 10, 2024. https://www.v7labs.com/blog/neural-style transfer#:~:text=Neural%20Style%20Transfer%20is% 20a,are%20its%20real-world%20applications.

[2] A. Helwan, “Recent advancements in GANs and style transfer - Abdulkader Helwan - medium,” Medium, Dec. 06, 2023. [Online]. Available: https://abdulkaderhelwan.medium.com/recent advancements-in-gans-and-style-transfer b7ba2b54cc68#:~:text=History%20of%20GANs%20a nd%20Style%20Transfer, Generative%20adversarial%20networks&text=Style% 20transfer%20is%20the%20process,to%20create%20a %20new%20image.

[3] Y. Liao and Y. Huang, “Deep Learning-Based application of Image Style Transfer,” Mathematical Problems in Engineering, vol. 2022, pp. 1–10, Aug. 2022, doi: 10.1155/2022/1693892.

[4] “COCO/WikiArt NST Dataset,” Kaggle, Feb. 05, 2022. https://www.kaggle.com/datasets/shaorrran/coco-wikiart-nst-dataset-512-100000/data

[5] GfG, “Neural Style Transfer with TensorFlow,” GeeksforGeeks, Dec. 06, 2023. https://www.geeksforgeeks.org/neural-style-transfer-with-tensorflow/

[6] P. Baheti, “Neural Style Transfer: Everything You Need to know [Guide],” V7, Apr. 24, 2023. https://www.v7labs.com/blog/neural-style-transfer

[7] Jauregui, A. F. (2022, August 19). *How to code Neural Style Transfer in Python*. Ander Fernández. https://anderfernandez.com/en/blog/how-to-code-neural-style-transfer-in-python/

[8] *نقل النمط العصبي*. (n.d.). TensorFlow. https://www.tensorflow.org/tutorials/generative/style_transfer?hl=ar