TECHNOLOGICAL INSTITUTE OF THE PHILIPPINES
938 Aurora Blvd., Cubao, Quezon City

COLLEGE OF ENGINEERING AND ARCHITECTURE
ELECTRONICS ENGINEERING DEPARTMENT

1ST SEMESTER SY 2022 - 2023

Prediction and Machine Learning

COE 005
ECE41S11

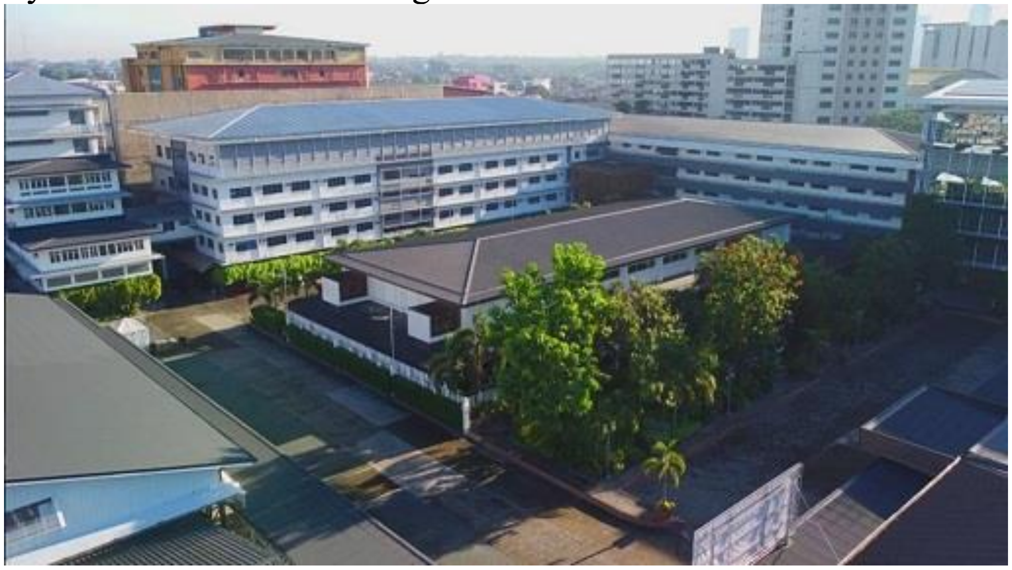
Homework 2
Neural Style Transfer

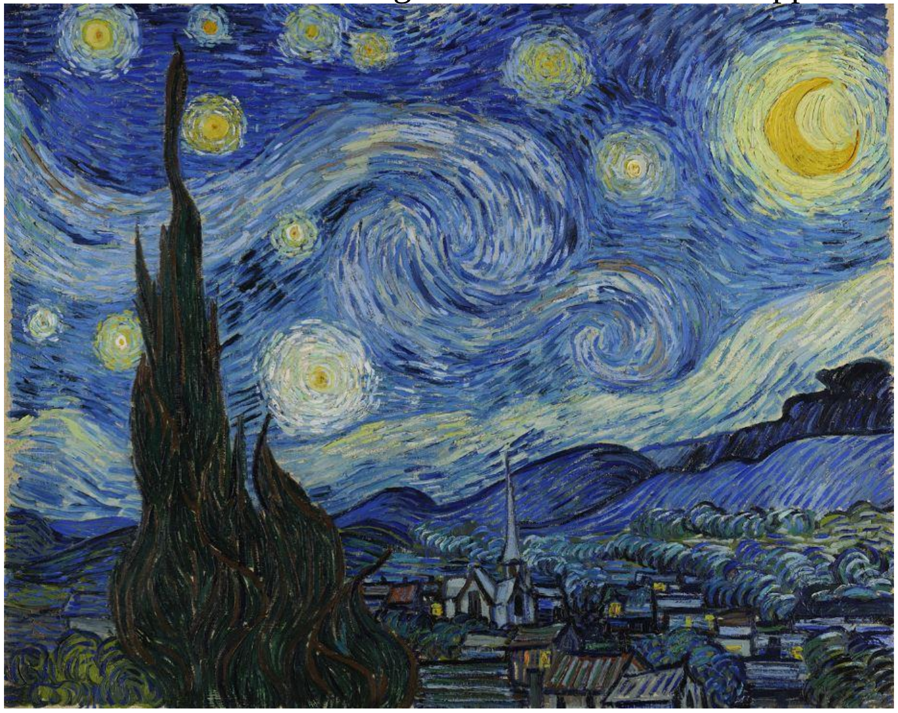
Submitted to:
Engr. Christian Lian Paulo Rioflorido

Submitted on:
10/19/2022

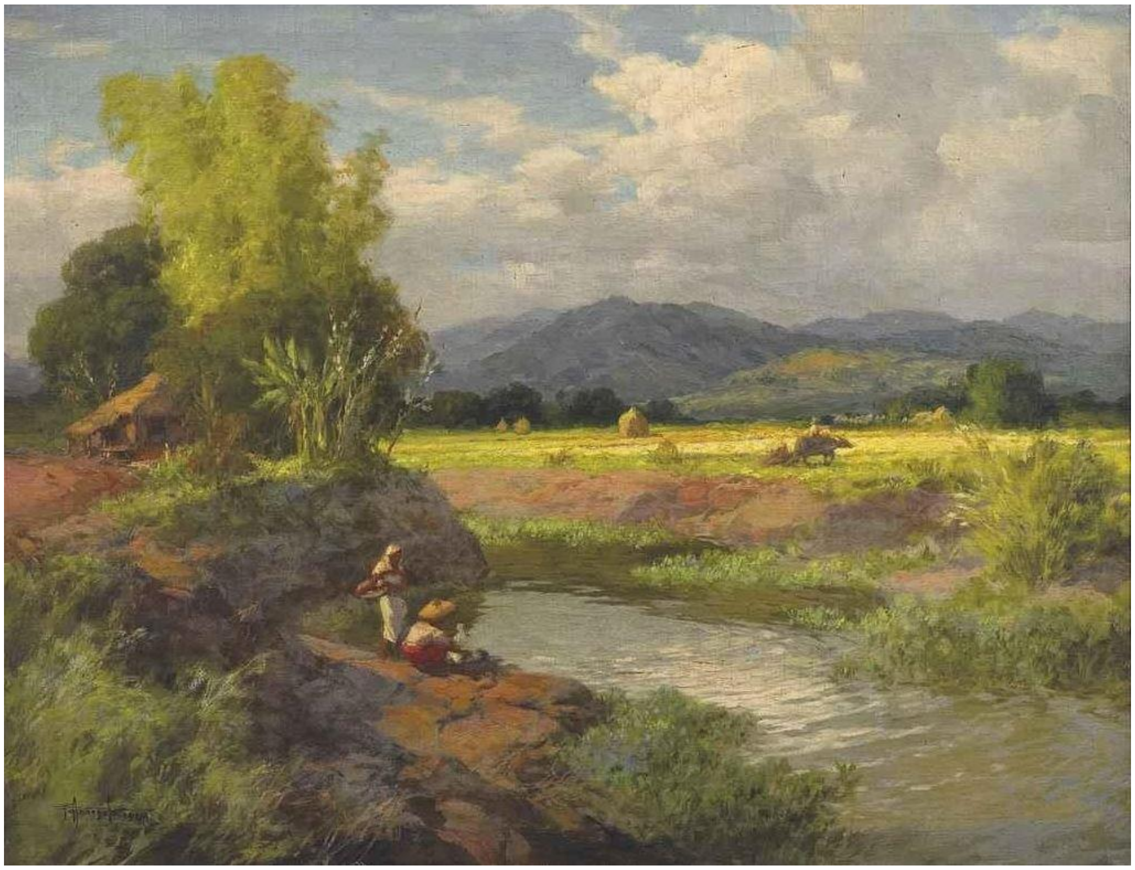
Submitted by:
So, Don Gabriel L.

INPUT AND OUTPUT IMAGES:
Input Images:
1. The Base Photo or the Content image is the Base Photo A of Technological Institute of the Philippines provided to us. The first art to be used is “The Starry Night” by Vincent Willem van Gogh in 1889.


Base Photo A. Technological Institute of the Philippines


Vincent Willen van Gogh (1889). *The Starry Night*.
Source: (the van gogh gallery, n.d)

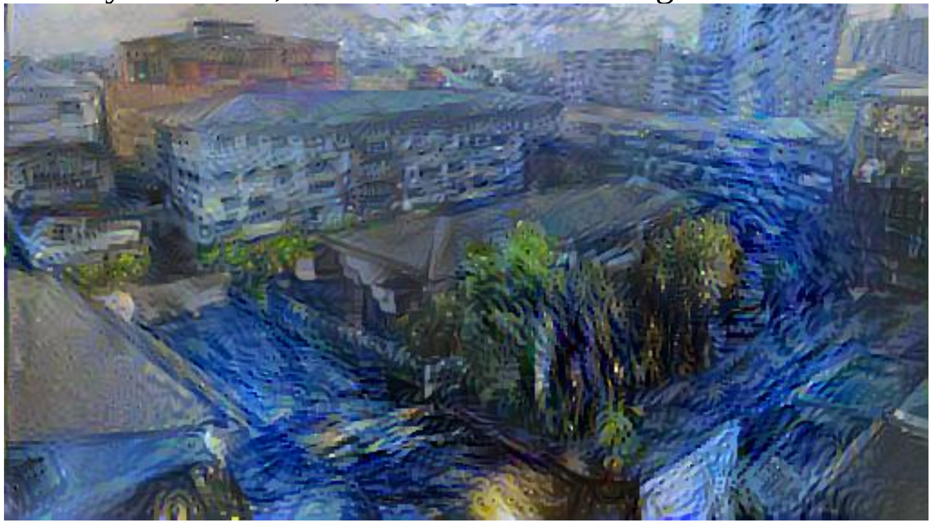
2. The same base photo provided to us will be used for the second style transfer. The second art to be used is the “Marikina Valley” by Fernando Amorsolo in 1948.



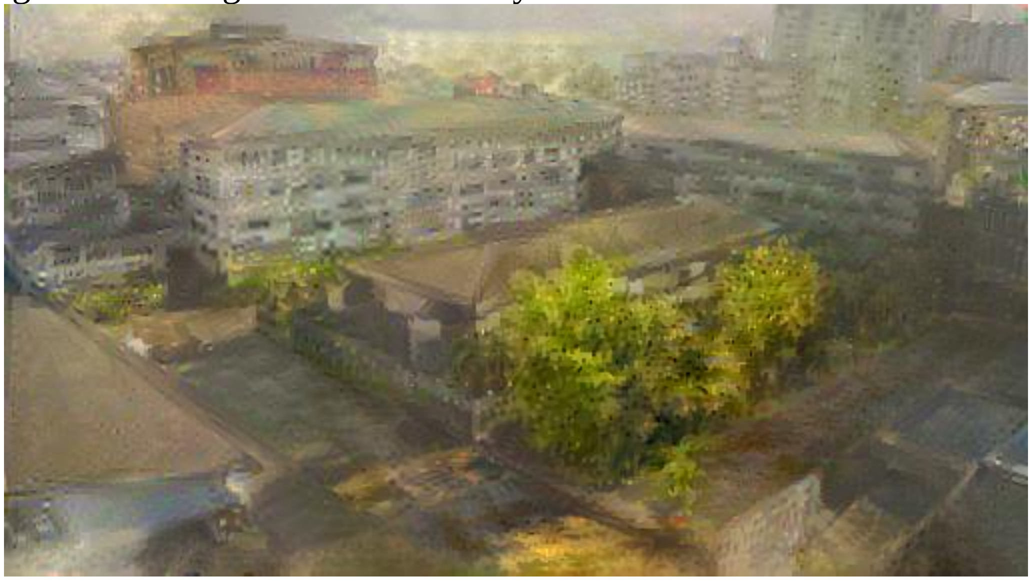
Fernando Amorsolo (1948). *Marikina Valley*.
Source: (mutual art, n.d)

Output Image:

1. For the first style transfer, below is the created image from the two images:



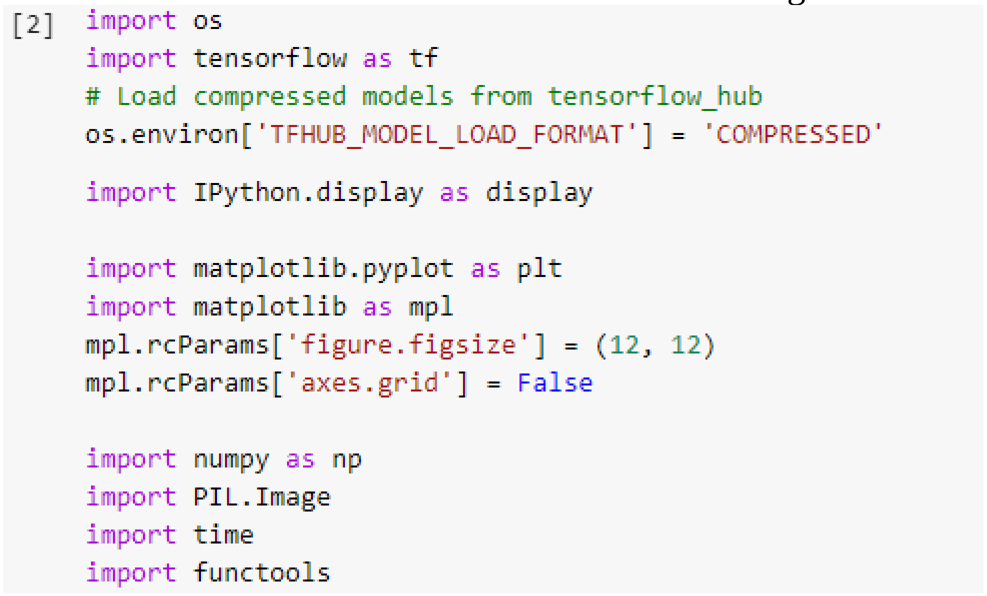
2. The generated image for the second style transfer is shown below:



PROCEDURE AND SCREENSHOTS OF THE PROGRAM:

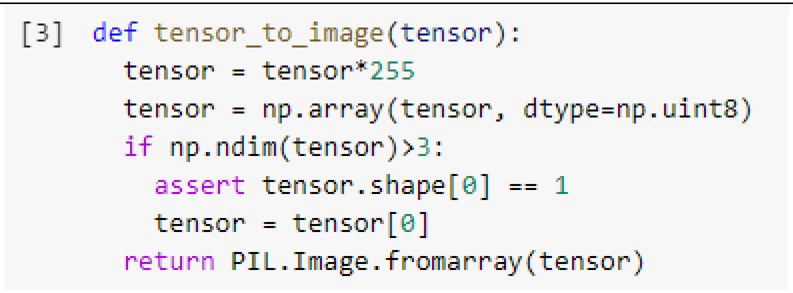
The editor used for this exercise is the Google Collab.

1. First, import the libraries needed for the program. Here, we also set the size of the figure which will be used for the visualization of the images.

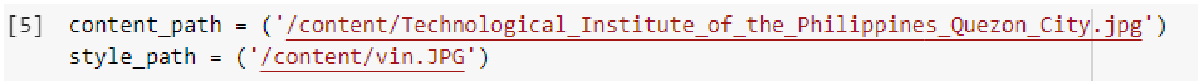


```
[2] import os
    import tensorflow as tf
    # Load compressed models from tensorflow_hub
    os.environ['TFHUB_MODEL_LOAD_FORMAT'] = 'COMPRESSED'

    import IPython.display as display

    import matplotlib.pyplot as plt
    import matplotlib as mpl
    mpl.rcParams['figure.figsize'] = (12, 12)
    mpl.rcParams['axes.grid'] = False

    import numpy as np
    import PIL.Image
    import time
    import functools
```

2. We made a function for converting a tensor to image. We also store the image saved in our local into a variable for us to easily call it later.

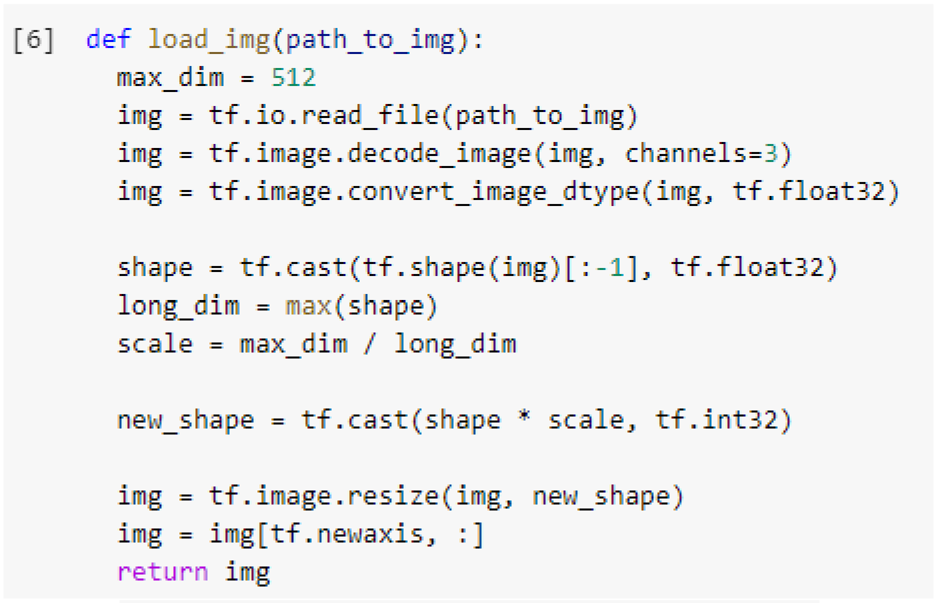


```
[3] def tensor_to_image(tensor):
      tensor = tensor*255
      tensor = np.array(tensor, dtype=np.uint8)
      if np.ndim(tensor)>3:
        assert tensor.shape[0] == 1
        tensor = tensor[0]
      return PIL.Image.fromarray(tensor)
```

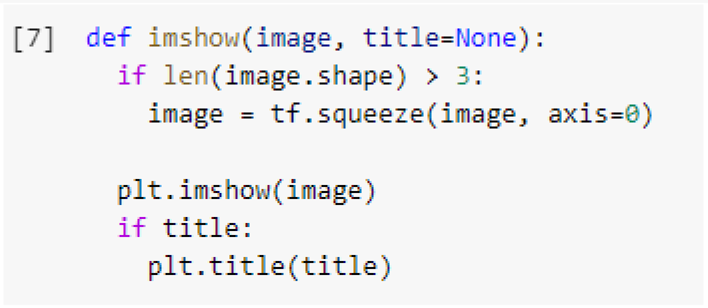


```
[5] content_path = ('/content/Technological_Institute_of_the_Philippines_Quezon_City.jpg')
    style_path = ('/content/vin.JPG')
```

3. We made a function to display the image. Also, while loading of the image, we set a function for limiting the pixels for its maximum dimension, which is set to 512 pixels.

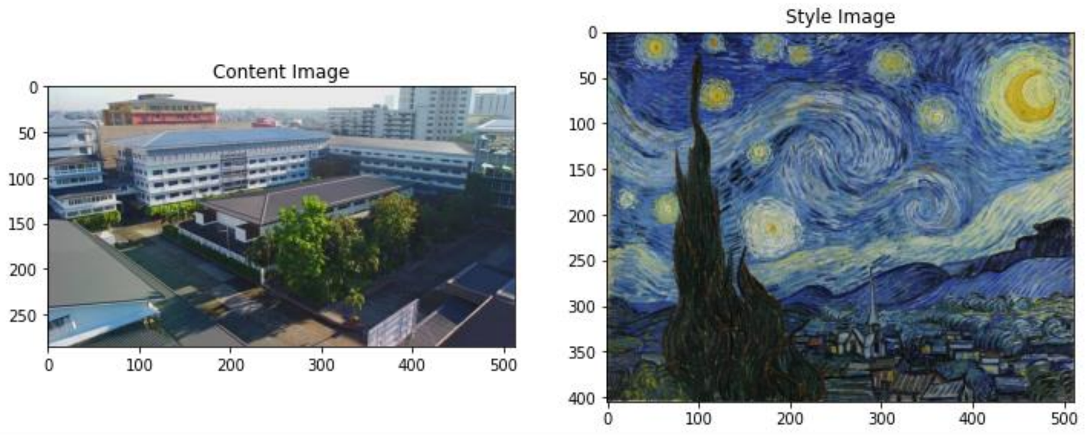


```
[6] def load_img(path_to_img):
      max_dim = 512
      img = tf.io.read_file(path_to_img)
      img = tf.image.decode_image(img, channels=3)
      img = tf.image.convert_image_dtype(img, tf.float32)

      shape = tf.cast(tf.shape(img)[:-1], tf.float32)
      long_dim = max(shape)
      scale = max_dim / long_dim

      new_shape = tf.cast(shape * scale, tf.int32)

      img = tf.image.resize(img, new_shape)
      img = img[tf.newaxis, :]
      return img
```



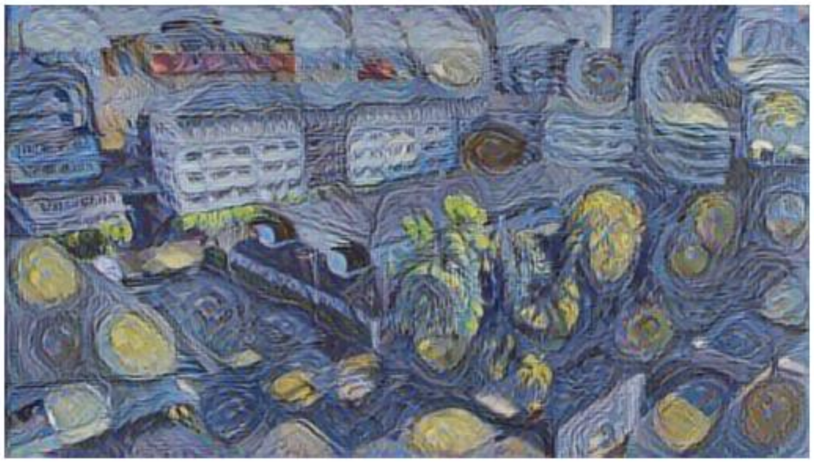
```
[7] def imshow(image, title=None):
      if len(image.shape) > 3:
        image = tf.squeeze(image, axis=0)

      plt.imshow(image)
      if title:
        plt.title(title)
```

```
[8]  content_image = load_img(content_path)
     style_image = load_img(style_path1)

     plt.subplot(1, 2, 1)
     imshow(content_image, 'Content Image')

     plt.subplot(1, 2, 2)
     imshow(style_image, 'Style Image')
```



4. We then demonstrate how the procedure is done when using a fast style transfer through an already trained model, which is the TensorFlow hub. This can also what we call the transfer learning wherein we take a trained model and feed new datas into it as long as the task to be done is the same as what the model is intended to do. We can see that the two images are quickly combined with the pretrained model.

```
[9] import tensorflow_hub as hub
    hub_model = hub.load('https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/2')
    stylized_image = hub_model(tf.constant(content_image), tf.constant(style_image))[0]
    tensor_to_image(stylized_image)
```

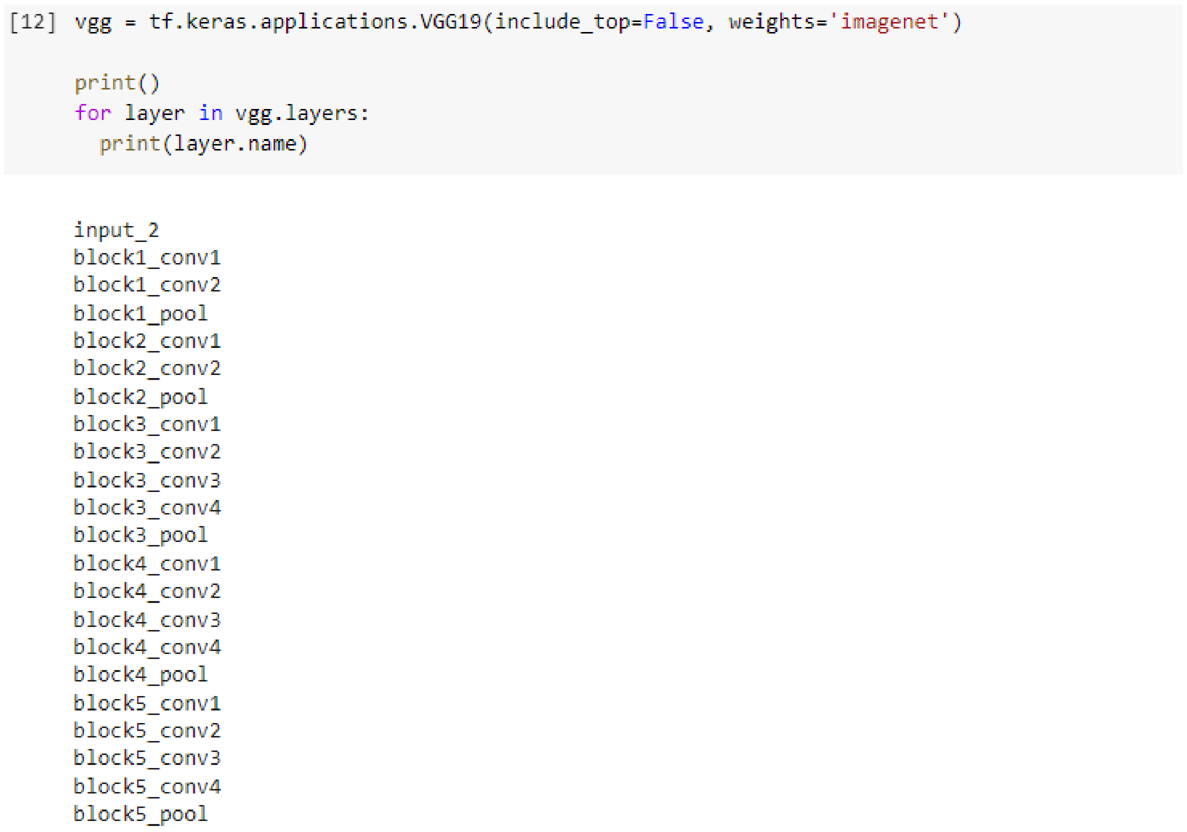


5. In here, we preprocess the input using the pretrained image classification network called VGG19. We also used VGG19 to try predicting the top 5 object or class in the image.

```
[10] x = tf.keras.applications.vgg19.preprocess_input(content_image*255)
     x = tf.image.resize(x, (224, 224))
     vgg = tf.keras.applications.VGG19(include_top=True, weights='imagenet')
     prediction_probabilities = vgg(x)
     prediction_probabilities.shape

     TensorShape([1, 1000])
```

```
[11] predicted_top_5 = tf.keras.applications.vgg19.decode_predictions(prediction_probabilities.numpy())[0]
     [(class_name, prob) for (number, class_name, prob) in predicted_top_5]

     [('dam', 0.13795137),
      ('dock', 0.11999227),
      ('pier', 0.0731488),
      ('garbage_truck', 0.043840103),
      ('crane', 0.0381756)]
```

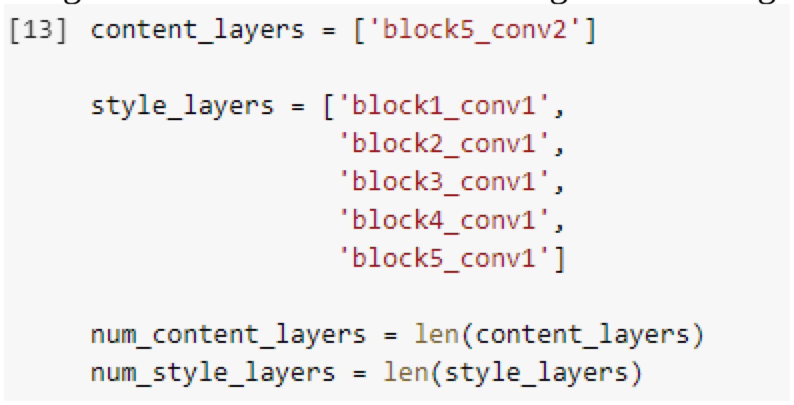
6. We now load the VGG19 model along with its layers. We can see that it has 5 blocks of convolutional layers with max pooling.



```
[12] vgg = tf.keras.applications.VGG19(include_top=False, weights='imagenet')

     print()
     for layer in vgg.layers:
       print(layer.name)
```

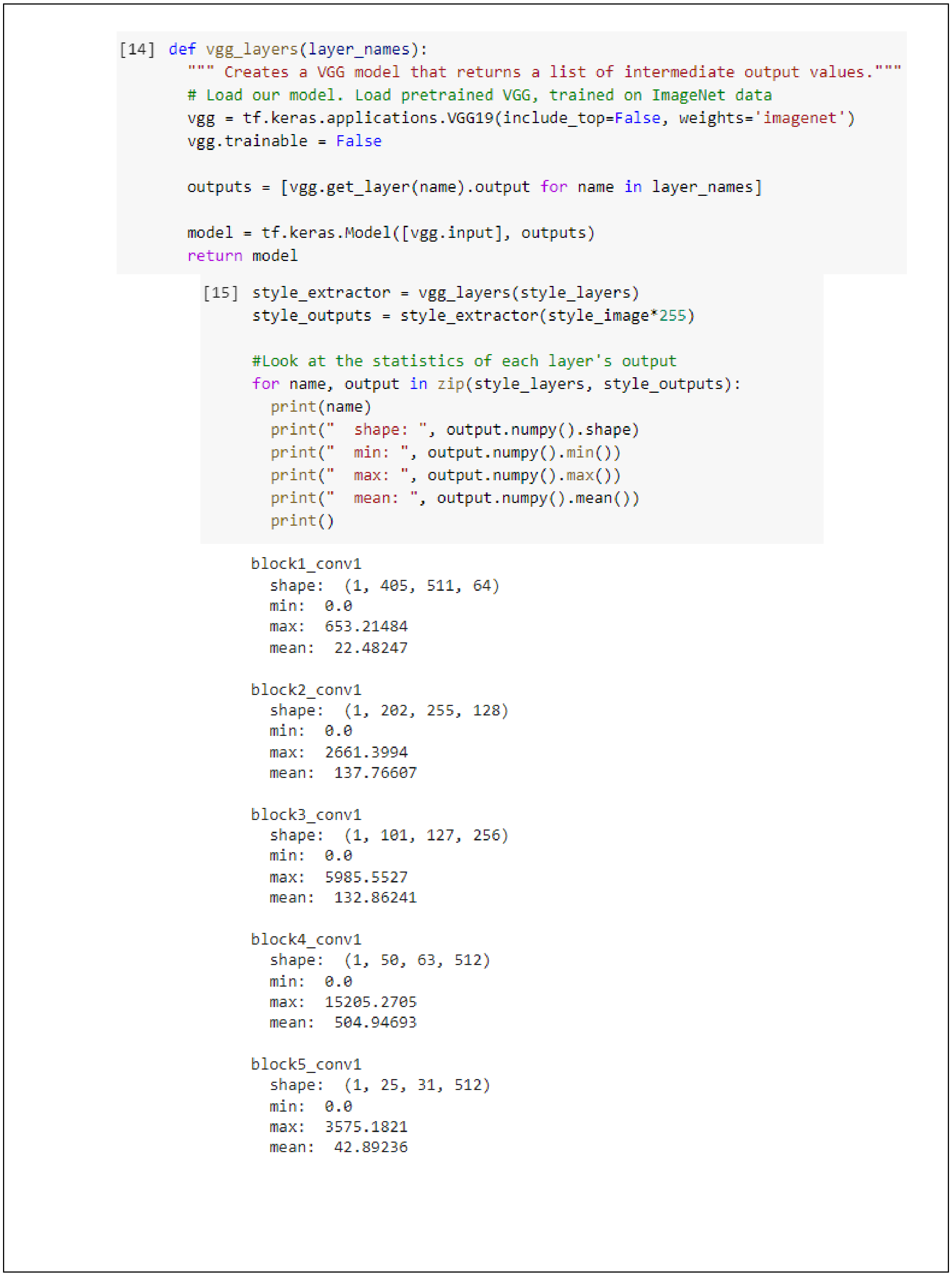
```
     input_2
     block1_conv1
     block1_conv2
     block1_pool
     block2_conv1
     block2_conv2
     block2_pool
     block3_conv1
     block3_conv2
     block3_conv3
     block3_conv4
     block3_pool
     block4_conv1
     block4_conv2
     block4_conv3
     block4_conv4
     block4_pool
     block5_conv1
     block5_conv2
     block5_conv3
     block5_conv4
     block5_pool
```

7. We then define which blocks of layer will be used for the styling. This will help in determining the features like texture and edges in the image.



```
[13] content_layers = ['block5_conv2']

     style_layers = ['block1_conv1',
                     'block2_conv1',
                     'block3_conv1',
                     'block4_conv1',
                     'block5_conv1']

     num_content_layers = len(content_layers)
     num_style_layers = len(style_layers)
```

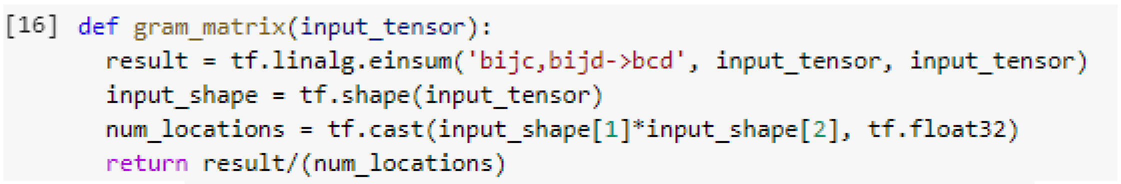
8. We define and create a VGG model. We also calculate the style by the function made for the gram matrix. We also build a class style model that extract and returns the style and content tensors. We also printed the information of each layer of the image like the shape.



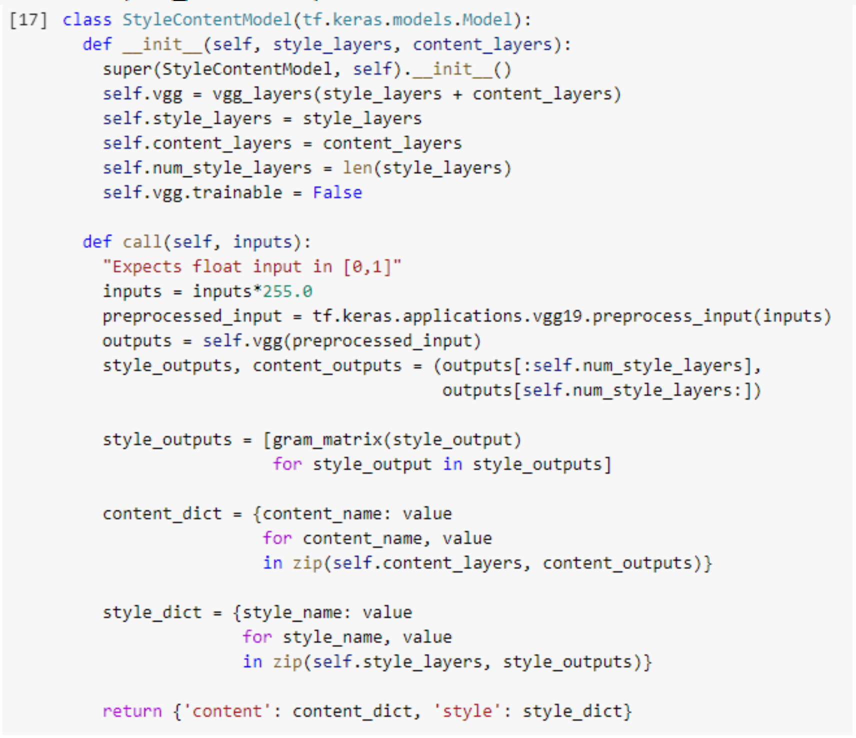
```
[14] def vgg_layers(layer_names):
       """ Creates a VGG model that returns a list of intermediate output values."""
       # Load our model. Load pretrained VGG, trained on ImageNet data
       vgg = tf.keras.applications.VGG19(include_top=False, weights='imagenet')
       vgg.trainable = False

       outputs = [vgg.get_layer(name).output for name in layer_names]

       model = tf.keras.Model([vgg.input], outputs)
       return model
```

```
[15] style_extractor = vgg_layers(style_layers)
     style_outputs = style_extractor(style_image*255)

     #Look at the statistics of each layer's output
     for name, output in zip(style_layers, style_outputs):
       print(name)
       print("  shape: ", output.numpy().shape)
       print("  min: ", output.numpy().min())
       print("  max: ", output.numpy().max())
       print("  mean: ", output.numpy().mean())
       print()
```

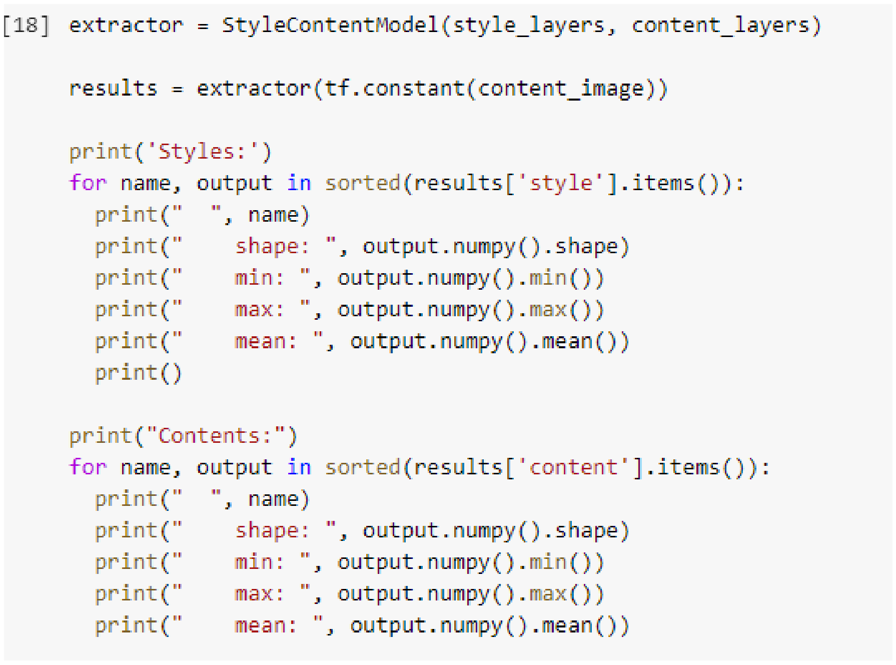
```
block1_conv1
  shape:  (1, 405, 511, 64)
  min:  0.0
  max:  653.21484
  mean:  22.48247

block2_conv1
  shape:  (1, 202, 255, 128)
  min:  0.0
  max:  2661.3994
  mean:  137.76607

block3_conv1
  shape:  (1, 101, 127, 256)
  min:  0.0
  max:  5985.5527
  mean:  132.86241

block4_conv1
  shape:  (1, 50, 63, 512)
  min:  0.0
  max:  15205.2705
  mean:  504.94693

block5_conv1
  shape:  (1, 25, 31, 512)
  min:  0.0
  max:  3575.1821
  mean:  42.89236
```



```
[16] def gram_matrix(input_tensor):
       result = tf.linalg.einsum('bijc,bijd->bcd', input_tensor, input_tensor)
       input_shape = tf.shape(input_tensor)
       num_locations = tf.cast(input_shape[1]*input_shape[2], tf.float32)
       return result/(num_locations)
```



```
[17] class StyleContentModel(tf.keras.models.Model):
       def __init__(self, style_layers, content_layers):
         super(StyleContentModel, self).__init__()
         self.vgg = vgg_layers(style_layers + content_layers)
         self.style_layers = style_layers
         self.content_layers = content_layers
         self.num_style_layers = len(style_layers)
         self.vgg.trainable = False

       def call(self, inputs):
         "Expects float input in [0,1]"
         inputs = inputs*255.0
         preprocessed_input = tf.keras.applications.vgg19.preprocess_input(inputs)
         outputs = self.vgg(preprocessed_input)
         style_outputs, content_outputs = (outputs[:self.num_style_layers],
                                           outputs[self.num_style_layers:])

         style_outputs = [gram_matrix(style_output)
                          for style_output in style_outputs]

         content_dict = {content_name: value
                         for content_name, value
                         in zip(self.content_layers, content_outputs)}

         style_dict = {style_name: value
                       for style_name, value
                       in zip(self.style_layers, style_outputs)}

         return {'content': content_dict, 'style': style_dict}
```

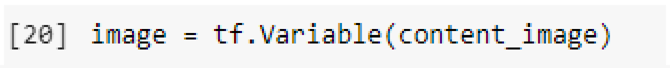


```
[18] extractor = StyleContentModel(style_layers, content_layers)

     results = extractor(tf.constant(content_image))

     print('Styles:')
     for name, output in sorted(results['style'].items()):
       print("  ", name)
       print("    shape: ", output.numpy().shape)
       print("    min: ", output.numpy().min())
       print("    max: ", output.numpy().max())
       print("    mean: ", output.numpy().mean())
       print()

     print("Contents:")
     for name, output in sorted(results['content'].items()):
       print("  ", name)
       print("    shape: ", output.numpy().shape)
       print("    min: ", output.numpy().min())
       print("    max: ", output.numpy().max())
       print("    mean: ", output.numpy().mean())
```

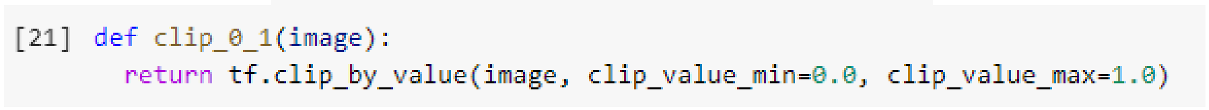
```
Styles:
   block1_conv1
    shape:  (1, 64, 64)
    min:  0.065126754
    max:  16041.414
    mean:  545.6823

   block2_conv1
    shape:  (1, 128, 128)
    min:  0.0
    max:  130720.56
    mean:  17327.887

   block3_conv1
    shape:  (1, 256, 256)
    min:  0.0
    max:  433203.0
    mean:  17715.902

   block4_conv1
    shape:  (1, 512, 512)
    min:  0.0
    max:  5725965.5
    mean:  254118.81

   block5_conv1
    shape:  (1, 512, 512)
    min:  0.0
    max:  148656.88
    mean:  1918.6746

Contents:
   block5_conv2
    shape:  (1, 17, 32, 512)
    min:  0.0
    max:  1146.8445
    mean:  16.948345
```

9. We implement now the style transfer algorithm by using the mean square error loss which is a keras loss function. We set here the style and content target values, storing of the content image into tf.Variable, and creating a function for converting the float image into pixel with values between 0 and 1.

```
[19] style_targets = extractor(style_image)['style']
     content_targets = extractor(content_image)['content']
```

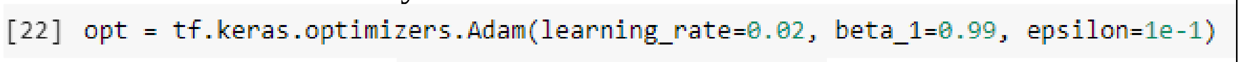


```
[20] image = tf.Variable(content_image)
```

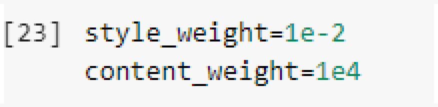


```
[21] def clip_0_1(image):
       return tf.clip_by_value(image, clip_value_min=0.0, clip_value_max=1.0)
```

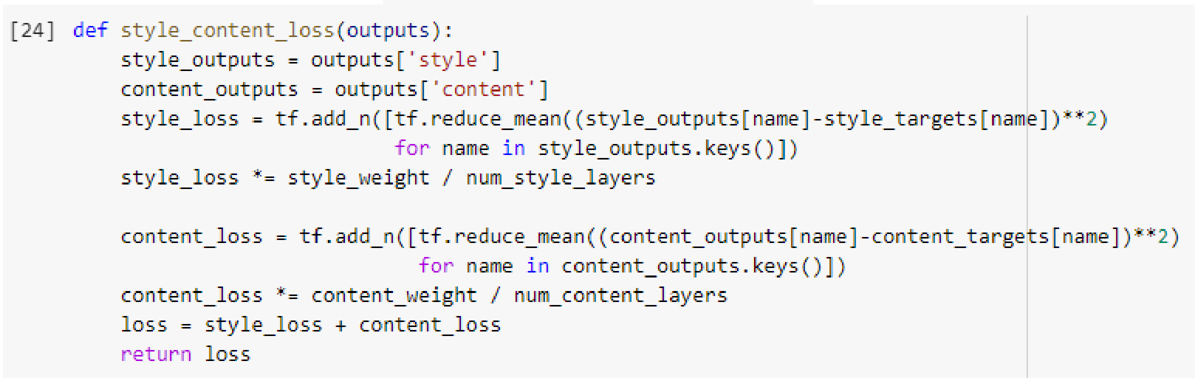
10. Next is by defining an optimizer, which is the adam optimizer for this program. We also set the weight for the style and content image. We also calculate the loss for the style and content and combine it.



```
[22] opt = tf.keras.optimizers.Adam(learning_rate=0.02, beta_1=0.99, epsilon=1e-1)
```

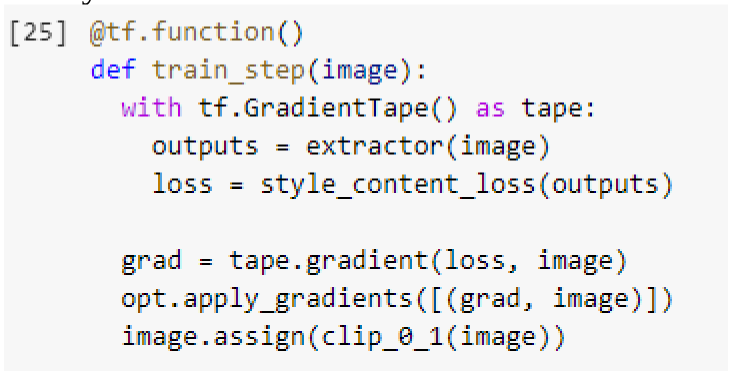


```
[23] style_weight=1e-2
     content_weight=1e4
```

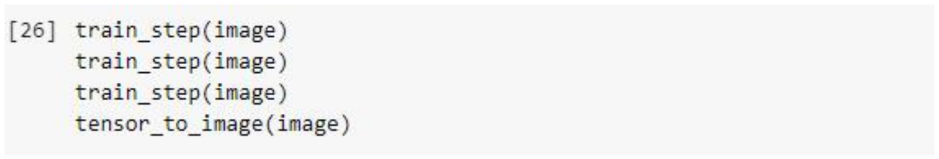


```
[24] def style_content_loss(outputs):
         style_outputs = outputs['style']
         content_outputs = outputs['content']
         style_loss = tf.add_n([tf.reduce_mean((style_outputs[name]-style_targets[name])**2)
                                for name in style_outputs.keys()])
         style_loss *= style_weight / num_style_layers

         content_loss = tf.add_n([tf.reduce_mean((content_outputs[name]-content_targets[name])**2)
                                  for name in content_outputs.keys()])
         content_loss *= content_weight / num_content_layers
         loss = style_loss + content_loss
         return loss
```

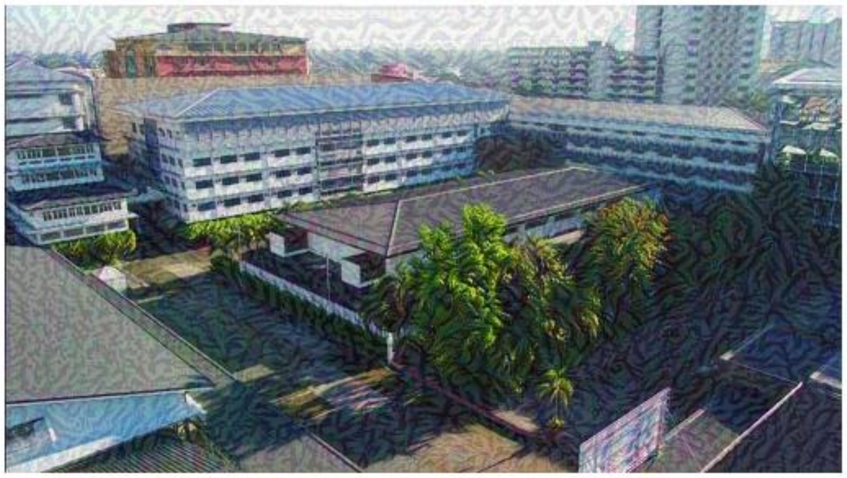
11. We use tf.GradientTape wherein all the extraction and optimization is gathered to update the image. We also display the simple style image of our transfer style model by the test function.



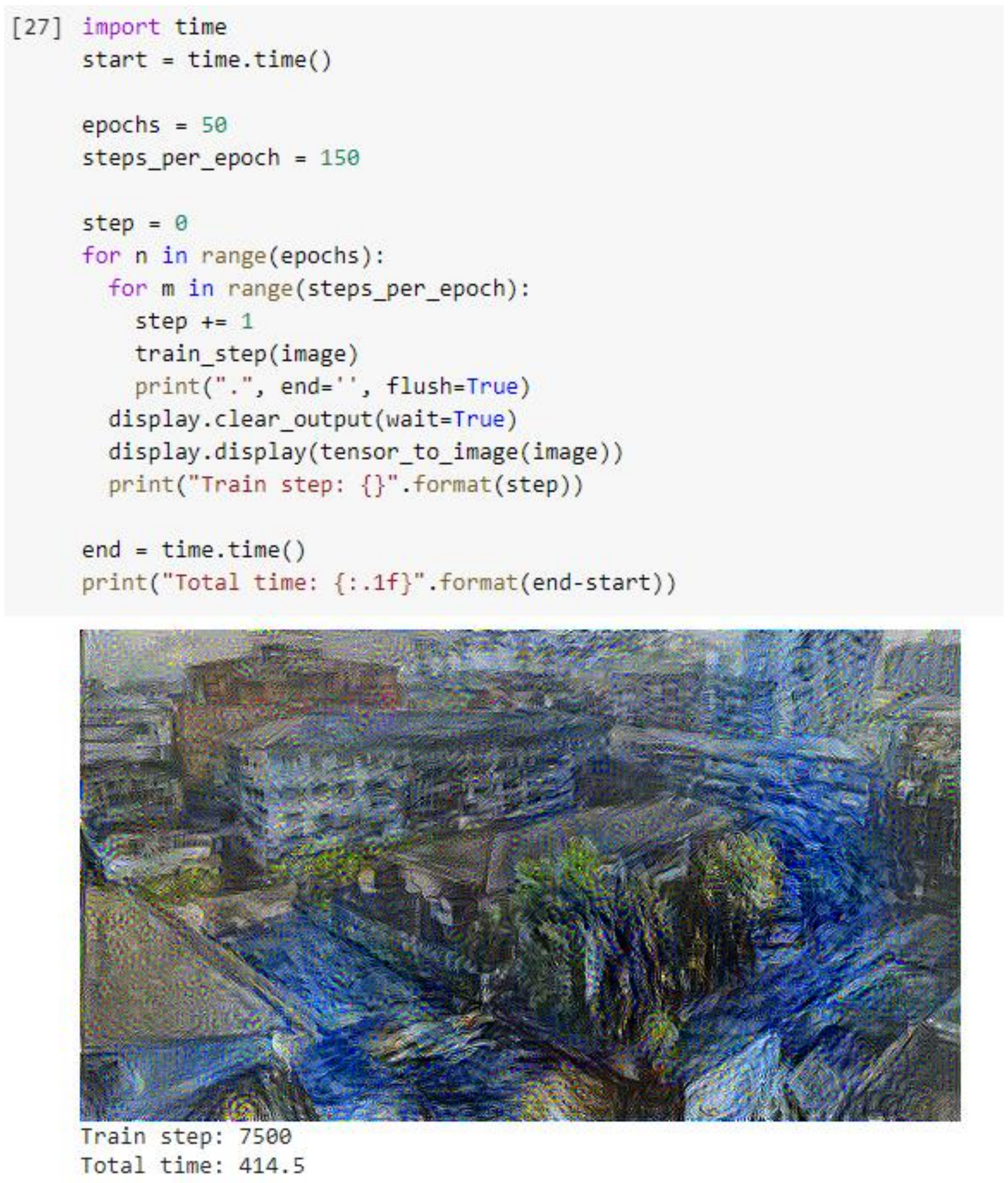
```
[25] @tf.function()
     def train_step(image):
       with tf.GradientTape() as tape:
         outputs = extractor(image)
         loss = style_content_loss(outputs)

       grad = tape.gradient(loss, image)
       opt.apply_gradients([(grad, image)])
       image.assign(clip_0_1(image))
```



```
[26] train_step(image)
     train_step(image)
     train_step(image)
     tensor_to_image(image)
```



12. The training of the model has started by transferring the art style of the style image into the content image. We also define how many epochs as well as steps per epoch for the fitting of the model. The Graphics Processing Unit (GPU) helped in having a faster speed of training. The image displayed shows that it has improved in transferring the art style compared to when we just call a simple style image from the previous step.

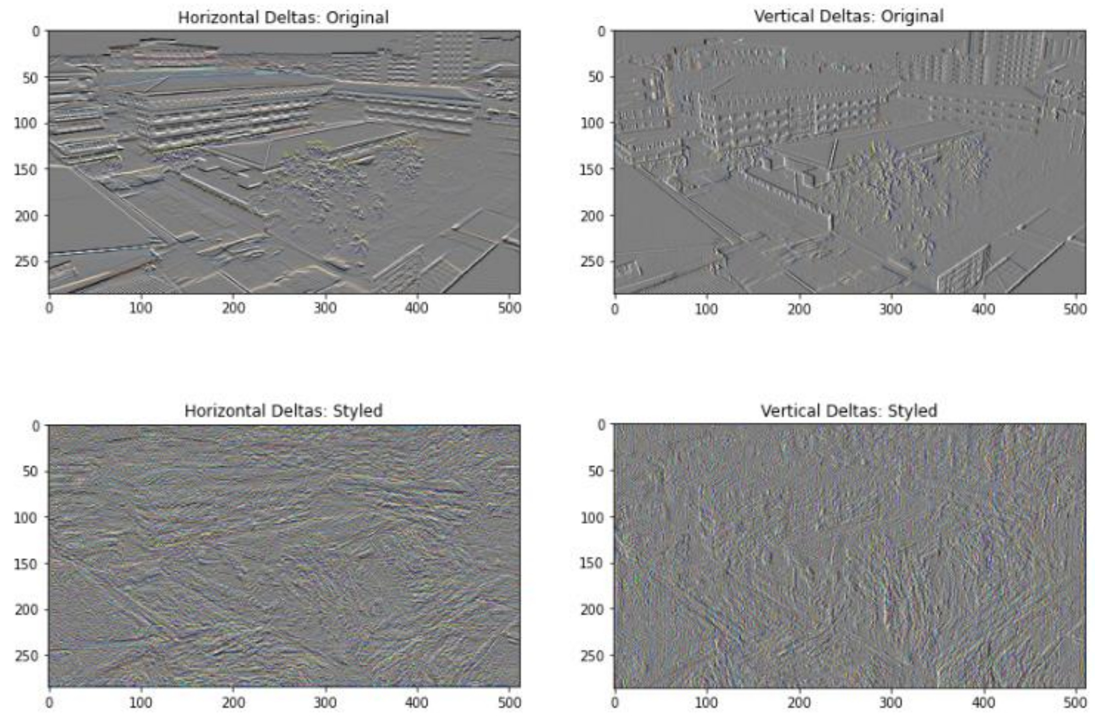


```
[27] import time
     start = time.time()

     epochs = 50
     steps_per_epoch = 150

     step = 0
     for n in range(epochs):
       for m in range(steps_per_epoch):
         step += 1
         train_step(image)
         print(".", end='', flush=True)
       display.clear_output(wait=True)
       display.display(tensor_to_image(image))
       print("Train step: {}".format(step))

     end = time.time()
     print("Total time: {:.1f}".format(end-start))
```

```
Train step: 7500
Total time: 414.5
```

13. We undergo another optimization process where we decrease high frequency artifacts in the image by explication regularization of the image. It emphasizes on the edges and lines in the image. After the optimization process, we reinitialize the image-variable and the optimizer.

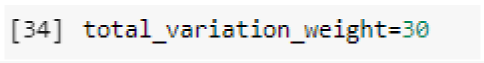
```
[29] x_deltas, y_deltas = high_pass_x_y(content_image)

     plt.figure(figsize=(14, 10))
     plt.subplot(2, 2, 1)
     imshow(clip_0_1(2*y_deltas+0.5), "Horizontal Deltas: Original")

     plt.subplot(2, 2, 2)
     imshow(clip_0_1(2*x_deltas+0.5), "Vertical Deltas: Original")

     x_deltas, y_deltas = high_pass_x_y(image)

     plt.subplot(2, 2, 3)
     imshow(clip_0_1(2*y_deltas+0.5), "Horizontal Deltas: Styled")

     plt.subplot(2, 2, 4)
     imshow(clip_0_1(2*x_deltas+0.5), "Vertical Deltas: Styled")
```


Horizontal Deltas: Original
Vertical Deltas: Original
Horizontal Deltas: Styled
Vertical Deltas: Styled
0
50
100
150
200
250
0
100
200
300
400
500

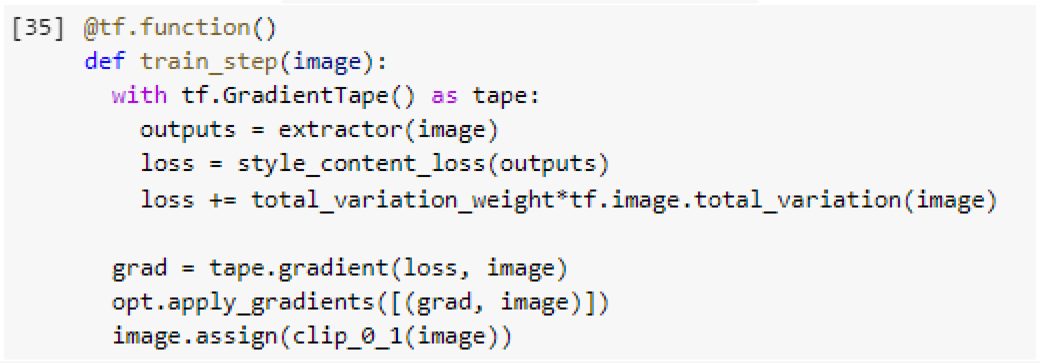
```
[31] def total_variation_loss(image):
       x_deltas, y_deltas = high_pass_x_y(image)
       return tf.reduce_sum(tf.abs(x_deltas)) + tf.reduce_sum(tf.abs(y_deltas))
```

```
[32] total_variation_loss(image).numpy()

     139125.16
```

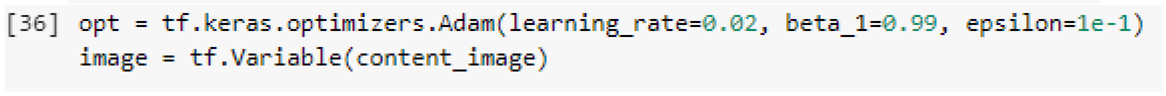
```
[33] tf.image.total_variation(image).numpy()

     array([139125.16], dtype=float32)
```



```
[34] total_variation_weight=30
```



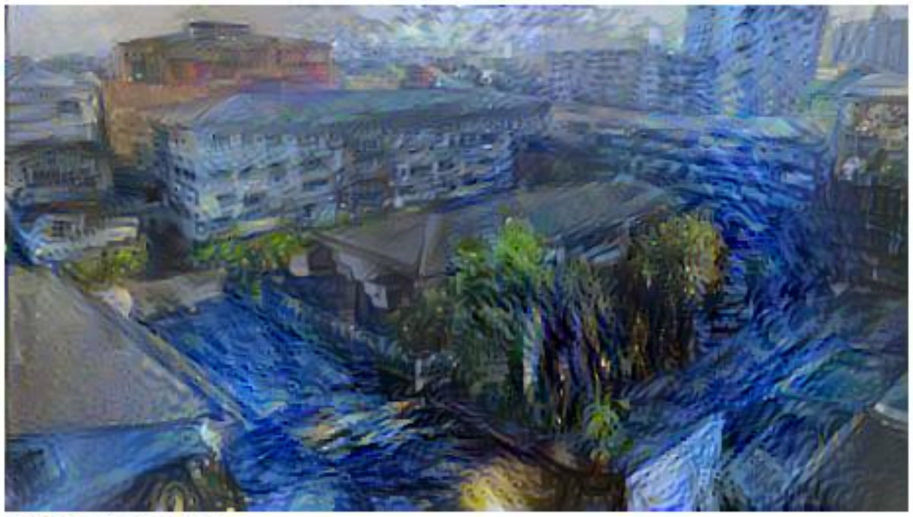
```
[35] @tf.function()
     def train_step(image):
       with tf.GradientTape() as tape:
         outputs = extractor(image)
         loss = style_content_loss(outputs)
         loss += total_variation_weight*tf.image.total_variation(image)

       grad = tape.gradient(loss, image)
       opt.apply_gradients([(grad, image)])
       image.assign(clip_0_1(image))
```



```
[36] opt = tf.keras.optimizers.Adam(learning_rate=0.02, beta_1=0.99, epsilon=1e-1)
     image = tf.Variable(content_image)
```

14. We then retrain it with the same number of epochs and steps per epoch wherein we can observe that the style transfer has a better quality than when it is not optimized again. We can see that the details become more clear, sharp and clean. The famous painting titled "The Starry Night" by Vincent Willem van Gogh was able to be combined with the image of TIP. The newly formed art gives off a cosmic feeling in TIP.

```
[37] import time
     start = time.time()

     epochs = 50
     steps_per_epoch = 150

     step = 0
     for n in range(epochs):
       for m in range(steps_per_epoch):
         step += 1
         train_step(image)
         print(".", end='', flush=True)
       display.clear_output(wait=True)
       display.display(tensor_to_image(image))
       print("Train step: {}".format(step))

     end = time.time()
     print("Total time: {:.1f}".format(end-start))
```

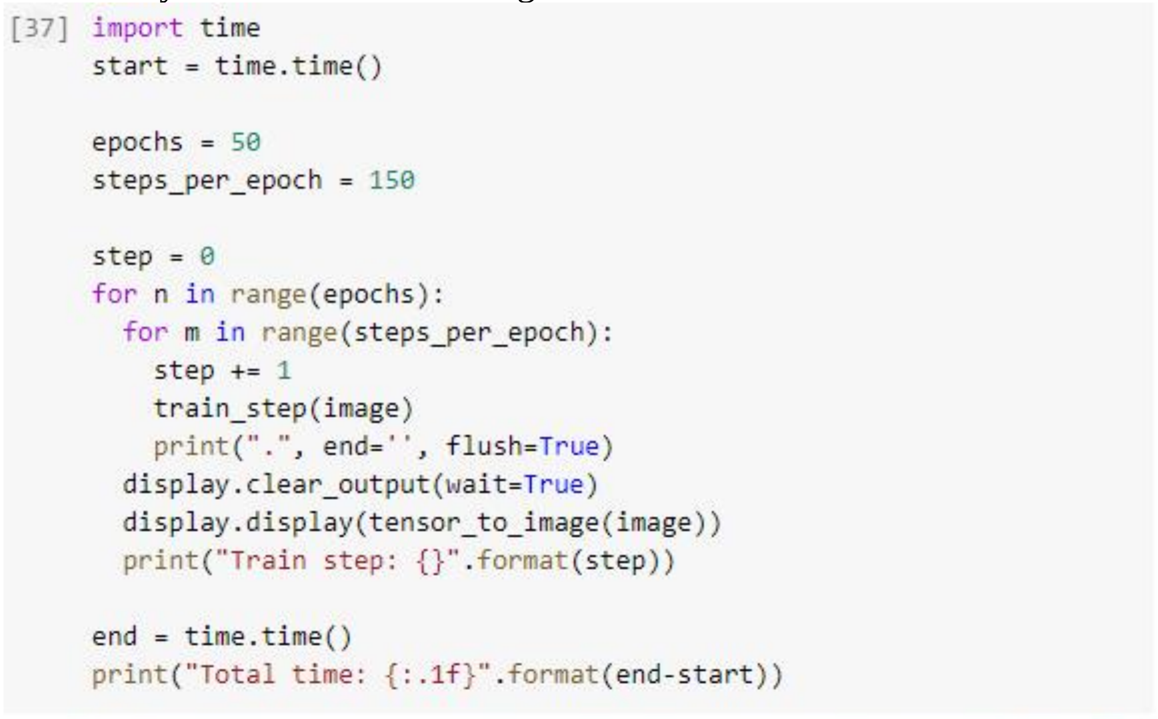


```
Train step: 7500
Total time: 468.8
```

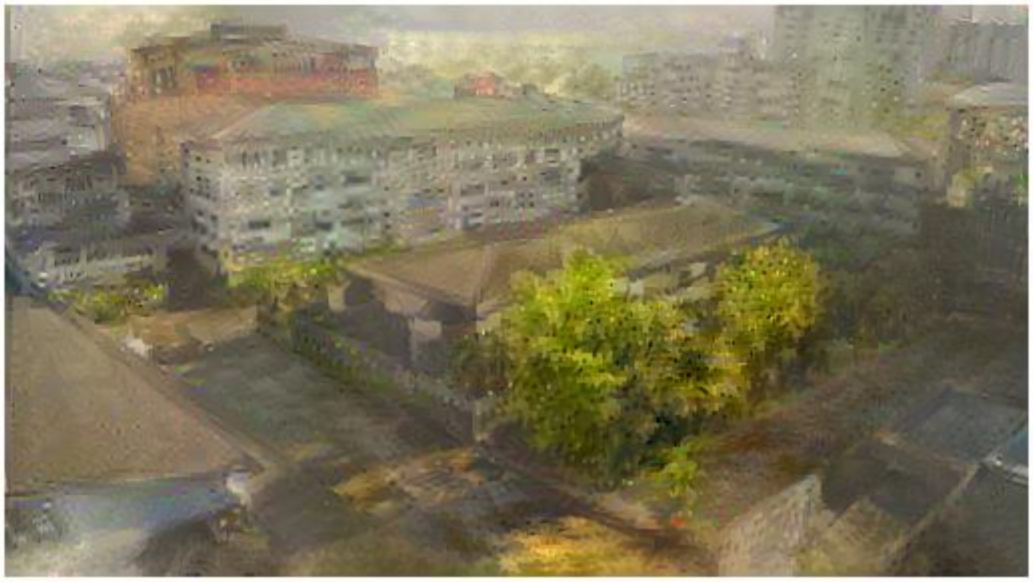
15. The produced art will then be saved in the same directory of the program.

```
file_name = 'stylized-image.png'
tensor_to_image(image).save(file_name)

try:
  from google.colab import files
except ImportError:
   pass
else:
  files.download(file_name)
```

16. Repeat all the procedure for the second style transfer. The second art is a combination of TIP and Fernando Amorsolo's piece, the "Marikina Valley." The newly produced art greatly encapsulated the art style of Fernando Amorsolo, which is illuminated landscape in which the new art captured by highlighting the green scenery as well as the use of light.



```
[37] import time
     start = time.time()

     epochs = 50
     steps_per_epoch = 150

     step = 0
     for n in range(epochs):
       for m in range(steps_per_epoch):
         step += 1
         train_step(image)
         print(".", end='', flush=True)
       display.clear_output(wait=True)
       display.display(tensor_to_image(image))
       print("Train step: {}".format(step))

     end = time.time()
     print("Total time: {:.1f}".format(end-start))
```



```
Train step: 7500
Total time: 497.8
```

DISCUSSION:
This activity helped me to learn and understand neural style transfer while performing the neural style transfer itself. With the help of coding and the programming language Python, I was able to combine two arts from different artists which are Vincent Willem van Gogh and Fernando Amorsolo, to an image of Technological Institute of the Philippines. The generated pictures greatly capture the art style of the two artists. My knowledge is further expanded since it shows me how to do transfer learning as well as an optimization technique and creation of a model through deep learning.

References:

Amorsolo, Fernando. *Marikina Valley*. 1948. *Mutual Art*. https://www.mutualart.com/Artwork/MarikinaValley/0C48E56E431BF782. Accessed October 18, 2022.

Neural style transfer: TensorFlow Core. TensorFlow. (2022, September 27). Retrieved October 18, 2022, from https://www.tensorflow.org/tutorials/generative/style_transfer

Van Gogh, Vincent Willem. *The Starry Night*. 1889. *The Van Gogh Gallery*. https://www.vangoghgallery.com/painting/starry-night.html. Accessed October 18, 2022.