Technological Institute of the Philippines
938 Aurora Blvd., Cubao, Quezon City

College of Engineering and Architecture
Electronics Engineering Department

Homewrok II
NEURAL STYLE TRANSFER

Submitted by:
Gabotero, Clive Jake A.
ECE41S1

Submitted to:
Engr. Christian Lian Paulo P. Riofloridо, MSEE

October 2022

In this Neural Style Transfer activity, I used pretrained models from the TensorFlow Hub and thus it is broadly elaborated in this paper how optimization is applied on output images while blending two input images. On the first image blending or style transfer, I used the image of the TechnoCore building in the Technological Institute of the Philippines Quezon City as the content image and Juan Luna's Spolarium as the styling image. And on the second style transfer, I used the same picture of the content image but with Leonardo Da Vinci's The Battle of Anghiari as the styling image.

Style Transfer 1 (TechnoCore building + Juan Luna's Spolarium):

To start the Neural Style Transfer, it is important to load the necessary libraries.

```
import tensorflow_hub as hub
import tensorflow as tf
from matplotlib import pyplot as plt
import numpy as np
import cv2

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
```

In these next lines of code, the tensor is being set up while the content image and the style image is loaded up from the google drive.

```
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)
```

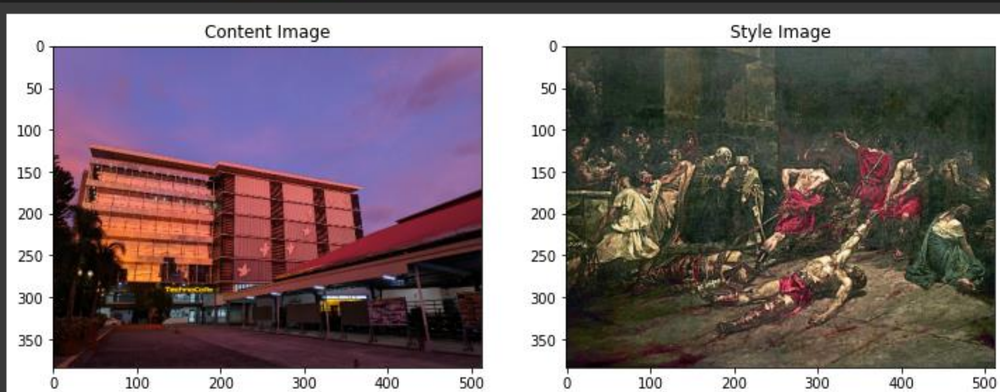
```
content_path = tf.keras.utils.get_file('/content/drive/MyDrive/COE 005 Homework/techno_core.jpg',
                                        'https://drive.google.com/drive/folders/1_GNRtQ9TZPYPCaAu5TAQM_C8Fksi
style_path = tf.keras.utils.get_file('/content/drive/MyDrive/COE 005 Homework/Juan.jpg',
                                      'https://drive.google.com/drive/folders/1_GNRtQ9TZPYPCaAu5TAQM_C8FksiC
```

Then, we make sure that the imported images are what we wanted to import by plotting them. We first define the dimensions and convert the type of image into float for the program to be able to plot the image.

```python
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
```

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

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

```python
content_image = load_img(content_path)
style_image = load_img(style_path)

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

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



From the lines of codes above, we have successfully plotted and checked the images that we have loaded. The content image being the TechnoCore building and the style image as the spolarium. Now we are going to define the content and style representations of the image. Load the VGG19 and test run it on our image to ensure its use.

```
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])
```

```
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]
```

```
[('cinema', 0.29461548),
 ('library', 0.040501863),
 ('fire_engine', 0.0394187),
 ('planetarium', 0.037683602),
 ('freight_car', 0.03280033)]
```

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

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

```
input_6
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
```

```
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)
```

To determine the representation of content and style from the images, these intermediary layers are required. At these intermediate levels, we match the corresponding style and content target representations for an input image.

Now it is time to build the model. We first specify the inputs and outputs in defining a model using the functional API and builds a VGG19 model which returns a list of layers.

```
def vgg_layers(layer_names):

  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
```

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

    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()
```

And the output list of layers:

```
block1_conv1
  shape:  (1, 384, 512, 64)
  min:  0.0
  max:  822.94116
  mean:  19.770576

block2_conv1
  shape:  (1, 192, 256, 128)
  min:  0.0
  max:  4083.1968
  mean:  119.76165

block3_conv1
  shape:  (1, 96, 128, 256)
  min:  0.0
  max:  6125.7256
  mean:  114.87366

block4_conv1
  shape:  (1, 48, 64, 512)
  min:  0.0
  max:  12711.32
  mean:  459.0458

block5_conv1
  shape:  (1, 24, 32, 512)
  min:  0.0
  max:  2022.1099
  mean:  35.536526
```

Using the tf.linalg.einsum function, calculating the Gram matrix in describing the style of the image can be achieved.

```python
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)
```

Now we extract the content and the style tensors using these next lines of codes.

```python
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}
```

```python
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())
```

Therefore, the tensor representations are as follows.

```
Styles:
   block1_conv1
    shape:  (1, 64, 64)
    min:  0.0043622446
    max:  18652.807
    mean:  720.94574

   block2_conv1
    shape:  (1, 128, 128)
    min:  0.0
    max:  142128.66
    mean:  19088.027

   block3_conv1
    shape:  (1, 256, 256)
    min:  0.06989401
    max:  477494.78
    mean:  19772.676

   block4_conv1
    shape:  (1, 512, 512)
    min:  0.0
    max:  6317188.0
    mean:  304248.03

   block5_conv1
    shape:  (1, 512, 512)
    min:  0.0
    max:  109087.49
    mean:  2228.7

Contents:
   block5_conv2
    shape:  (1, 24, 32, 512)
    min:  0.0
    max:  1108.4441
    mean:  16.152256
```

With the style and content extracted, we can implement the style transfer algorithm by calculating the mean square error for the image's output then we create an optimizer using the weighted losses to get the total loss.

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

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

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

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

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

```
[ ] 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
```

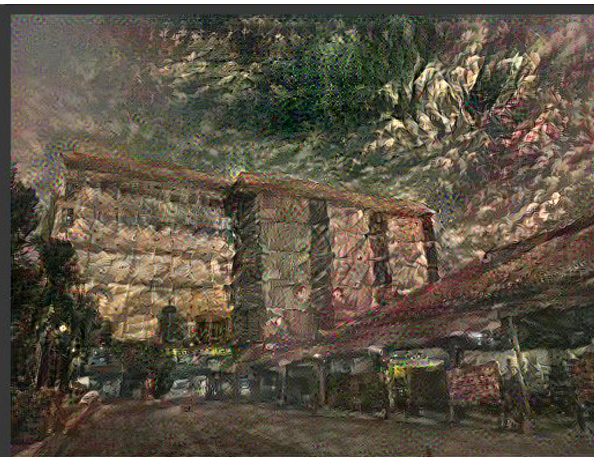
```
[ ] @tf.function()
    def train_step(image):
      with tf.GradientTape() as tape:
        outputs = extractor(image)
        loss = style_content_loss(outputs)

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

Then we test and see an output image.

```
train_step(image)
train_step(image)
train_step(image)
tensor_to_image(image)
```

We can therefore perform a longer optimization since the algorithm is working.

```python
import time
start = time.time()

epochs = 40
steps_per_epoch = 120

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))
```

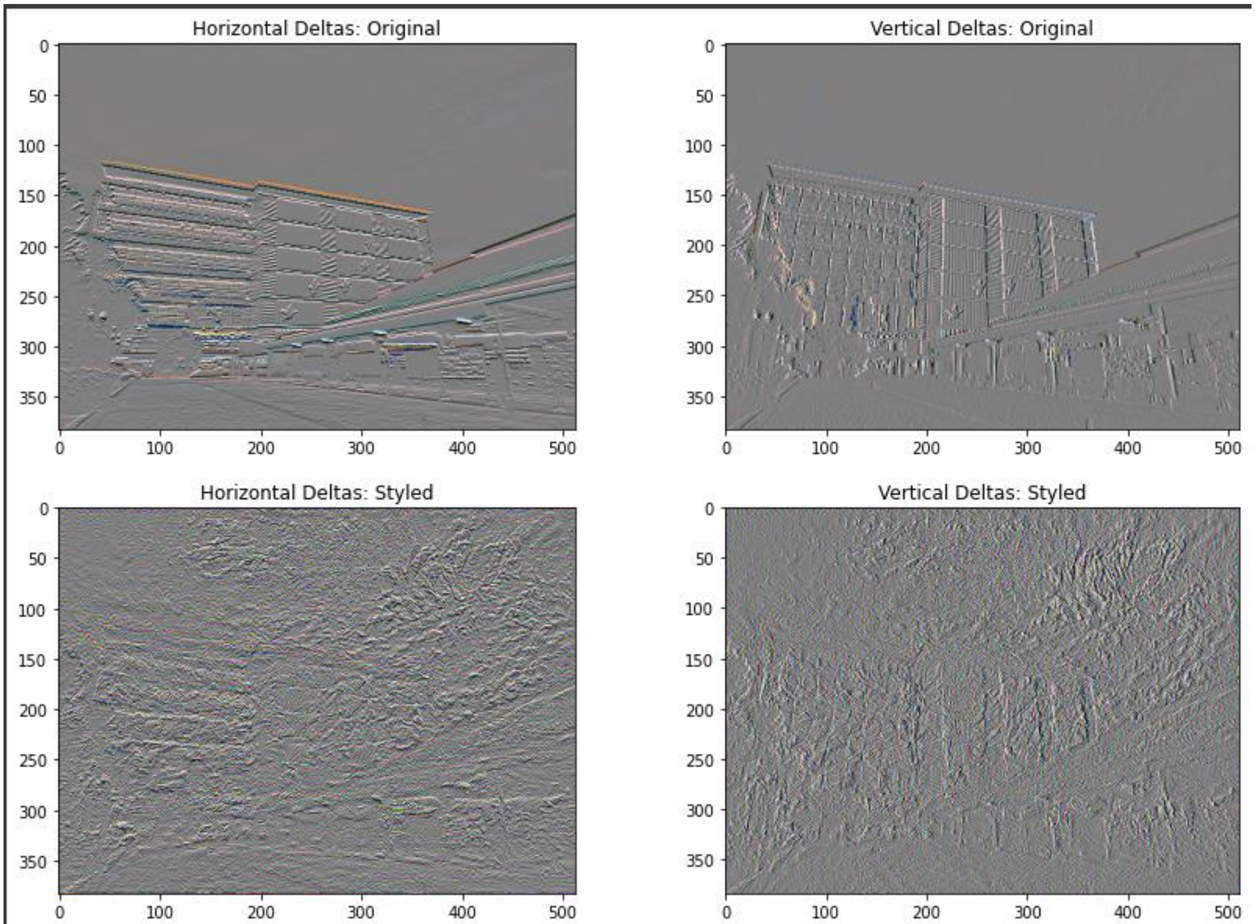


```
Train step: 4800
Total time: 336.5
```

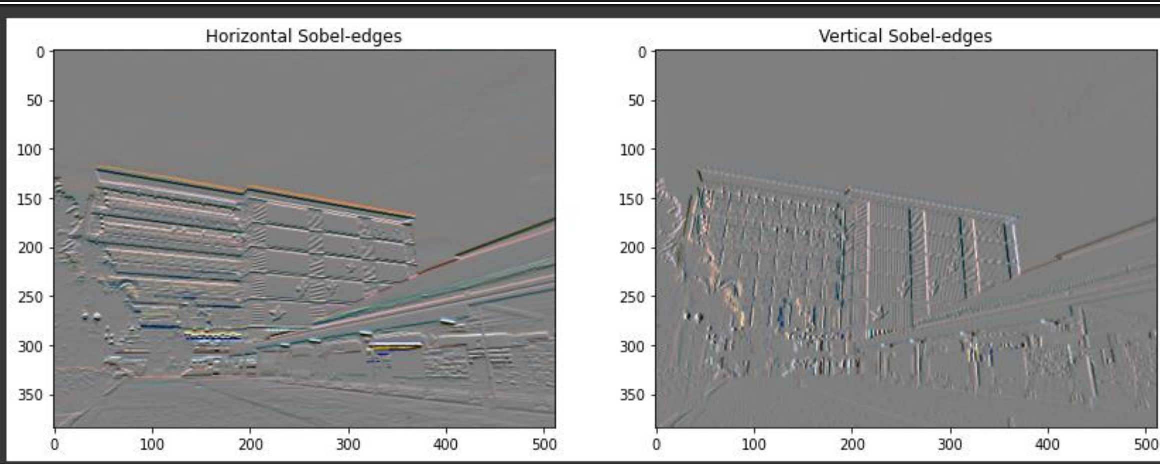
With this output, we can obtain the variation loss from the following lines of codes.

```python
def high_pass_x_y(image):
  x_var = image[:, :, 1:, :] - image[:, :, :-1, :]
  y_var = image[:, 1:, :, :] - image[:, :-1, :, :]

  return x_var, y_var
```

```python
[ ] 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")
```




```
[ ]  plt.figure(figsize=(14, 10))

     sobel = tf.image.sobel_edges(content_image)
     plt.subplot(1, 2, 1)
     imshow(clip_0_1(sobel[..., 0]/4+0.5), "Horizontal Sobel-edges")
     plt.subplot(1, 2, 2)
     imshow(clip_0_1(sobel[..., 1]/4+0.5), "Vertical Sobel-edges")
```



```
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))
```

```
total_variation_loss(image).numpy()
```

```
140401.11
```

```
tf.image.total_variation(image).numpy()
```

```
array([140401.11], dtype=float32)
```

Now we must re-run the optimization using a weigh for the total variation loss to obtain a better output.

```
##Re-run Optimization
total_variation_weight=30
```

```
@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))
```

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

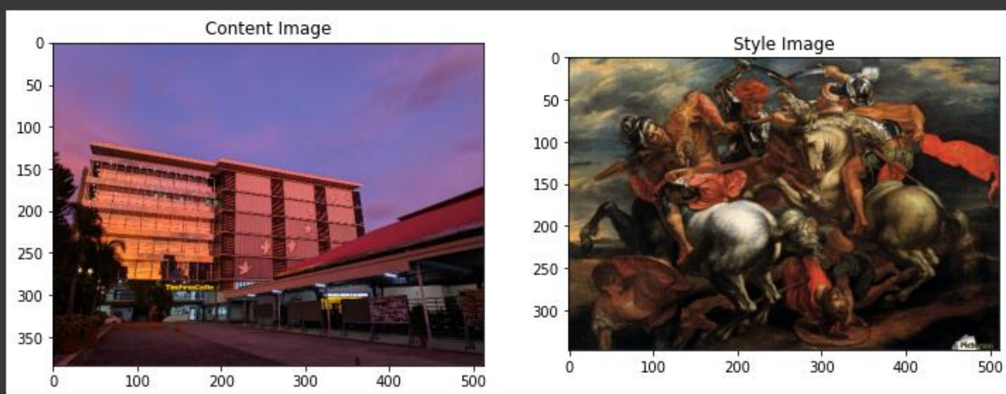
```
import time
start = time.time()

epochs = 40
steps_per_epoch = 120

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: 4800
Total time: 343.6
```

Style Transfer 2 (TechnoCore building + Leonardo Da Vinci's The Battle of Anghiari):

For the second style transfer, the same algorithm or codes with the first style transfer have been used. The only difference is that the style image used is Leonardo Da Vinci's The Battle of Anghiari. Shown below are the lines of codes showing the content, style, and finally the output images.

```
[ ]  content_image = load_img(content_path)
     style_image = load_img(style_path)

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

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

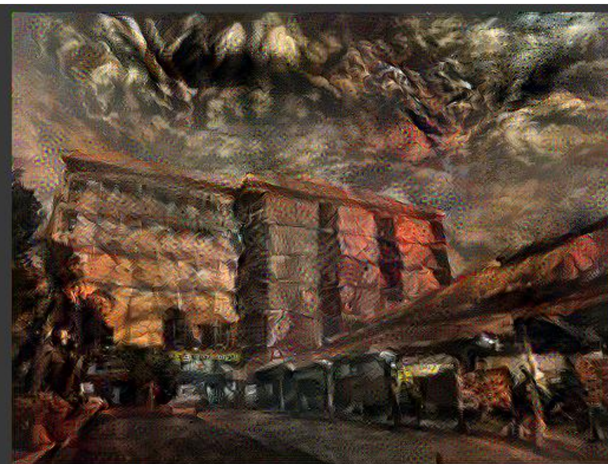


First test run:

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

Prolonged Optimization:

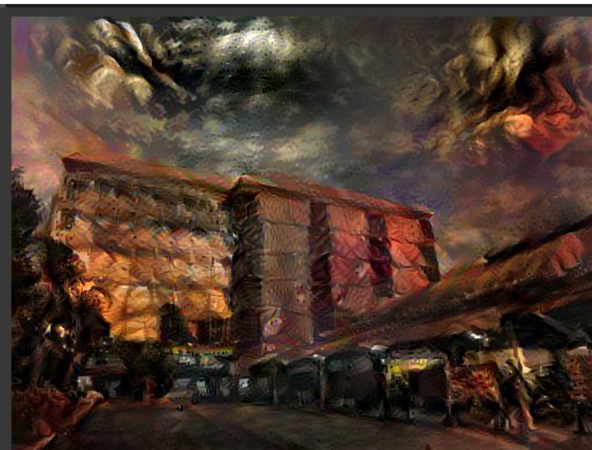
```python
import time
start = time.time()

epochs = 40
steps_per_epoch = 120

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: 4800
Total time: 344.6
```

Final output with a specified total variation loss of 30:

```python
import time
start = time.time()

epochs = 40
steps_per_epoch = 120

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))
```

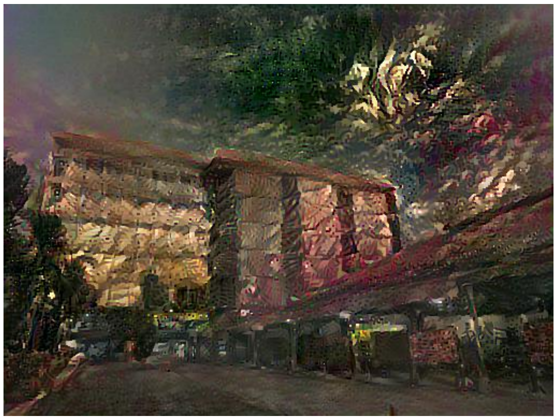


```
Train step: 4800
Total time: 352.3
```

Conclusion:

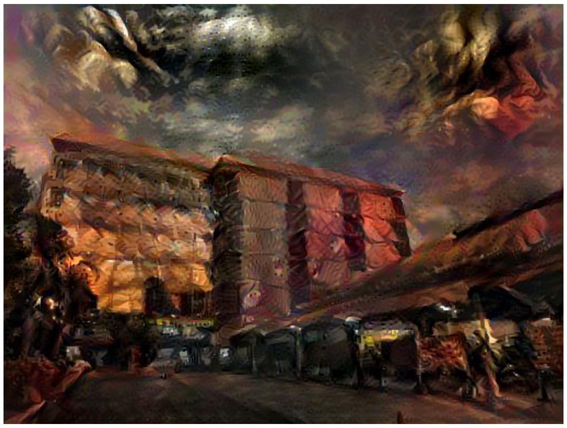
In this activity, I have learned how the Neural Style Transfer works and how painting styles of famous artists can be applied on a digitally captured image. Shown below are the content images, style images, and the resulting images obtained from the two Style Transfers:

Style Transfer 1:



Output Image

Style Transfer 2:



Output Image