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 P. Rioflorido, MSEE

Submitted on:
October 19, 2022

Submitted by:
Bataan, Franch Lee D.

Preparation:

In this activity, the student must use the Neural Style Transfer technique to create an image where the style of the other picture is applied to the chosen photo between photo A and photo B.

Photo A is used as a base photo for this activity as shown below:

Figure 1. Technological Institute of the Philippines in Bird's-eye view

The artworks shown below are the styles used by the student for the Neural Style Transfer activity:

Figure 2. Amorsolo, Fernando Cueto. (1939). *View of Bataan from Manila Bay*. [Oil in Canvas]. London, United Kingdom

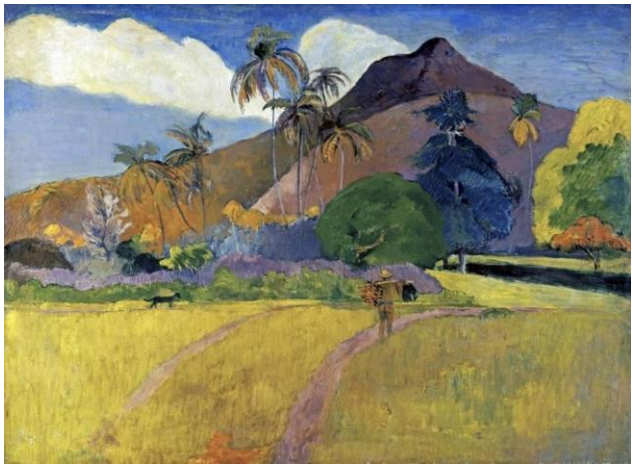

Figure 3. Gauguin, Paul. (1893). *Tahitian Landscape with a Mountain*. [Oil in Canvas]. France.

Background Information:

Neural style transfer is a technique that separates the image's content and style then combine them together to create a new output which has the same style as the style image (Baheti, 2022). With the application of convolutional neural network, it can be able to differentiate images from one another by being able to pick out and detect patterns and make sense of them. Convolutional neural network or CNN is a class of Deep Neural network that has a filter that performs pattern recognition between objects. It extracts the feature of the image and convert it into lower dimension without losing the characteristics of the image

An image is composed of pixels where each pixels represents color values of RGB. A convolution operation is applied where every 3x3 grid of pixels of the input image will be multiplied to the filter, a distinctive pattern part of an image, and each product values will take its average to be part of the feature map. As the 3x3 grid slid over every 3x3 block of input image, it creates a feature map that represents a certain feature of the input image.

$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau$$

Convolution formula (Eq. 1)

The activity uses two images with different styles to create a new image that uses the qualities of the two style images. The first style image used is the work of Fernando Cueto Amorsolo entitled "*View of Bataan from Manila Bay"* an artwork created at year 1939 while the second style image used is from the work of Paul Gaguin entitled, "*Tahitian Landscape with a Mountain",* an artwork develops at year 1893. The texture and qualities of these two style images is captured by using the feature space that is designed for capturing the information of the texture which was built above filter responses for the network layers. Incorporating the feature correlations of different layers, the input image's multiscale representation can be obtained that can get the texture information of the image.

The activity uses a pre-trained model called the VGG19. It is a CNN architecture that consist of nineteen layers with 16 of the layers are convolution layers, the remaining three are fully connected layer, five MaxPool Layers and has 1 SoftMax Layer (Dr. Sec. Info, 2021). Using the pretrained model, the activity allows the student to reconstruct the input images from its layers then combine the style of artworks with content image.

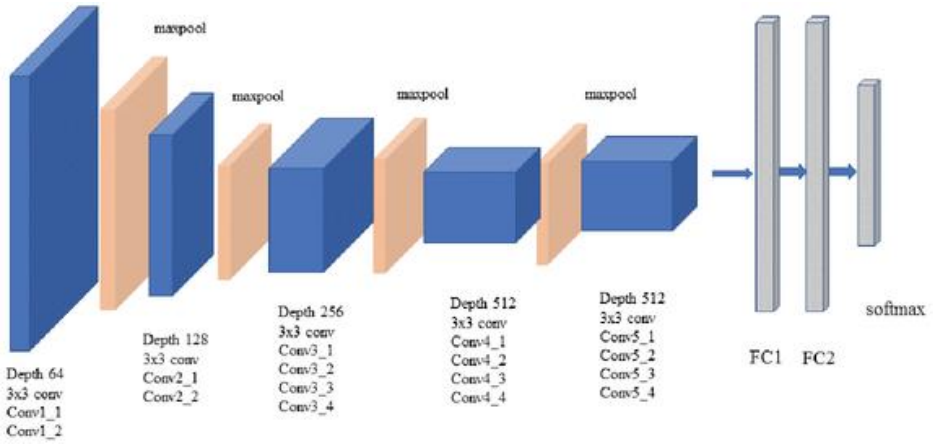

Figure 4. VGG19 Architecture (Source: *Blog.Techcraft,* 2021)

Materials Needed:

- Python IDE
- Google Collab or Jupyter notebook
- 1 Base photo and 2 artworks from different artist

Libraries:

- os
- Tensorflow
- IPython.display
- Matplotlib
- Numpy
- PIL.Image
- time
- functools

Simulation:

As all of the images that will be used for the activity is downloaded, the libraries needed can now be imported which will help the student to simplifies the algorithm and lessen the workload for the activity. All of the libraries imported have functionality related to the activity.

```
#Importing libraries needed
import os
import tensorflow as tf
import IPython.display as display
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np
import PIL.Image
import time
import functools
```

Figure 5. Libraries used for the activity.

Python library is a collection of modules of codes that helps to simplifies the algorithm using predefined set of codes that provides different system functionalities. The repetitiveness of using a code can be reduced by importing a library and calling a particular function in a module. In calling a specific library, the import function is used. In this simulation, different libraries were being used (Parthmanchanda81, 2021).

```
#Loading compressed models from tensorflow_hub
os.environ['TFHUB_MODEL_LOAD_FORMAT'] = 'COMPRESSED'
```

Figure 6. Loading Compressed Models from TensorFlow_hub

This code allows the student to read the compressed models from remote storage. In this case, models are loaded from TensorFlow_Hub.

```
content_path='/content/Technological_Institute_of_the_Philippines_Quezon_City.jpg'
style_path='/content/FA.png'
```

Figure 7.1. Path of the Base Image and Style Image (Artwork 1)

```
content_path='/content/Technological_Institute_of_the_Philippines_Quezon_City.jpg'
style_path='/content/Sold Price_ PAUL GAUGUIN - TAHITIAN LANDSCAPE WITH A MOUNTAIN - June 2, 0119 12_00 AM PDT.jpg'
```

Figure 7.2. Path of the Base Image and Style Image (Artwork 2)

The student used google collab due to its faster E-GPU performance that is fitted for this activity. Photos are uploaded from google drive and declared the path of the base photo and artworks to be read by the algorithm. The student created a two-python file and declared the different artworks to each algorithm.

```
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)
```

Figure 8. Converting the Input Tensor to Image Format

Figure 8 defines a function that allows the student to convert the input tensor to image format by changing the values of the pixels to [0,255]. Pixels are being converted from float to integer type. The PIL.Image.from array(tensor) is used for converting the tensor to image.

```
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
```

Figure 9. Loading Images to 512-pixel dimension.

Figure 9 defines a function that allows the student to load and limit images to 512 pixels. The process involves reading the declared path, convert the data type of the images that it read to float32, get the maximum dimension of the input image then resize the image with the data type is now change into int32.

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

Figure 10. Displaying Images

The figure 10 shows a function that allows the student to display the input images using the matplotlib.pyplot python library once the function is called.

```
base_image=load_img(content_path)
style_image=load_img(style_path)
```

Figure 11. Declaring the Images with 512-pixel Dimension

The images are now converted to 512-pixel dimension using the load_img function defined on figure 9. The reason why it has to undergo with the load_img function is to allow the student to display the images with the same pixel dimension.

```
mpl.rcParams['figure.figsize'] = (12, 12)
mpl.rcParams['axes.grid'] = False
plt.subplot(1, 2, 1)
imshow(base_image, 'Base Image')
plt.subplot(1, 2, 2)
imshow(style_image, 'View of Bataan from Manila Bay')
```

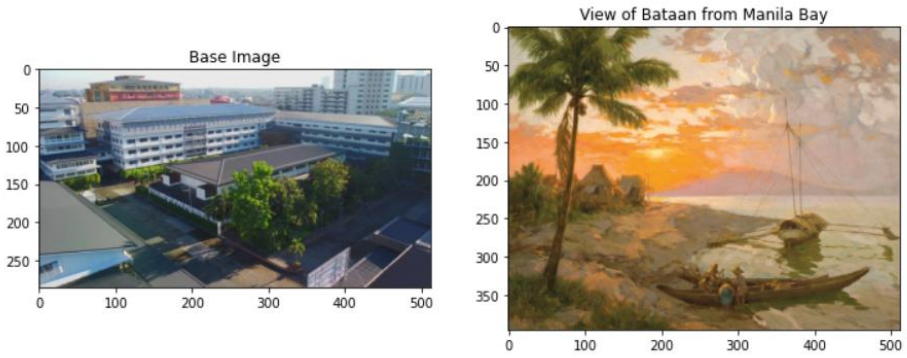

Figure 12.1 Displaying the Content and Style Image (Artwork 1)

```
mpl.rcParams['figure.figsize'] = (12, 12)
mpl.rcParams['axes.grid'] = False
plt.subplot(1, 2, 1)
imshow(base_image, 'Base Image')
plt.subplot(1, 2, 2)
imshow(style_image, 'TAHITIAN LANDSCAPE WITH A MOUNTAIN')
```

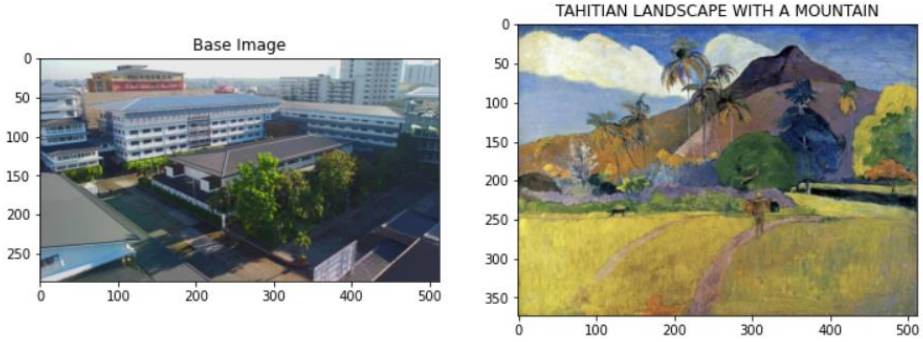

Figure 12.2 Displaying the Content and Style Image (Artwork 2)

Figure 12.1 and figure 12.2 shows the images that will be used for Neural style transfer. The images are displayed using the imshow function in figure 10. The style image from artwork 1 and artwork 2 have different qualities and art styles. The objective of the student is to transfer these qualities to the base image to create an image that posses the qualities of the artworks.

```
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(base_image), tf.constant(style_image))[0]
tensor_to_image(stylized_image)
```

Figure 13. Style Transfer using the Original Algorithm

Figure 13.1. Style Transfer using Artwork 1

Figure 13.2. Style Transfer using Artwork 2

```
x=tf.keras.applications.vgg19.preprocess_input(base_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
vgg = tf.keras.applications.VGG19(include_top=False, weights='imagenet')

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

Figure 14. Loading the VGG19 model

```
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
```

Figure 14 is used test the model if it can identify the content image. Content image must be resized as the VGG19 model used 224, 224 dimensions. As the student test the pre-trained model if it can predict the content image, the layers of VGG19 are being printed out to see the layer names without the classification head. The layers are needed to successfully define the style and content representations from the declared images that will be used for style transfer.

The image at the left side shows the intermediate layers of the pretrained model and the student has to choose for representing both the style and content of the image.

```
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)
```

Figure 15. Choosing the Intermediate Layers

Here, the student choose block5_conv2 as content layer while block1_conv1, block2_conv1, block3_conv1, block4_conv1, and block5_conv1 as for the style_layers. This allows the student to describe the content and tyle of the images used for this activity.

```
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
```

Figure 16. Create and load the Model

Figure 16 defines a new function that allows to identify which layers does the content and style must be extracted. Since the network will be used to extract the style and content of the images, the inckude_top is set to false instead of True. The network is already trained, and it is not necessary to train the network again. Using the vgg,get_layer(name), the weights of the layers are obtained.

```
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()
```

Figure 17. Statistics of the Layers

The style_extractor is declared by calling the vgg_layers and pass style_layers as an input of the argument for the aforementioned declared variable then is was called and pass the style_image*255 to return the names and outputs of the layer-wise. Then the shape, min, max, and mean is displayed the statistics of each output in layers.

```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)
```

Figure 18. Calculating the Style

$$G^l_{cd} = \frac{\sum_{ij} F^l_{ijc}(x) F^l_{ijd}(x)}{IJ}$$

Gram Matrix (Eq. 2)

The equation above is used to calculate the gram matrix for a specific layer of the model.

```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}
```

Figure 19. Function for Extracting Style and Content

Figure 19 shows a class that ables the student to build a model, having both the content and style tensors of the input images can be returned.

```python
extractor = StyleContentModel(style_layers, content_layers)
results = extractor(tf.constant(base_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())
```

Figure 20. Displaying the Shape, Min, Max and Mean of the Style and Content of the Image.

The images below are the output from figure 20.

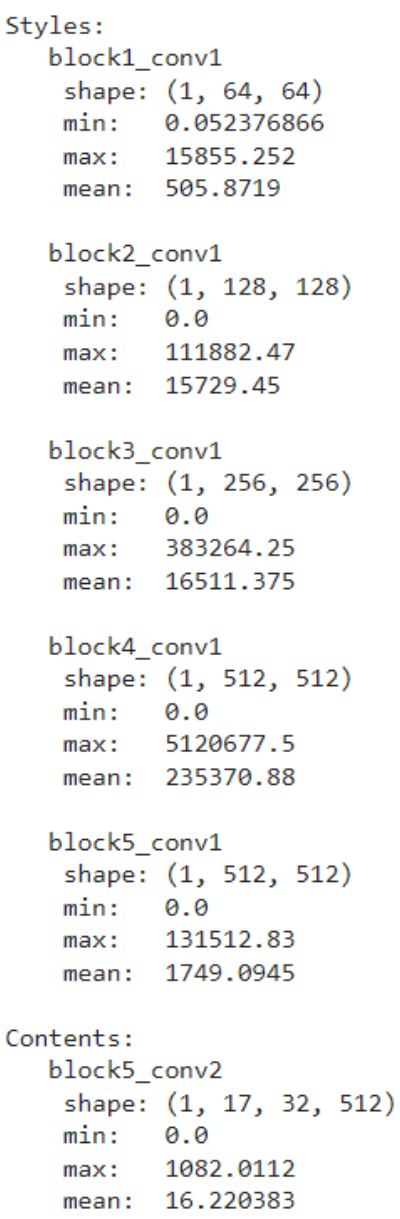

Figure 20.1 Output Information of the Content and Style for Artwork 1

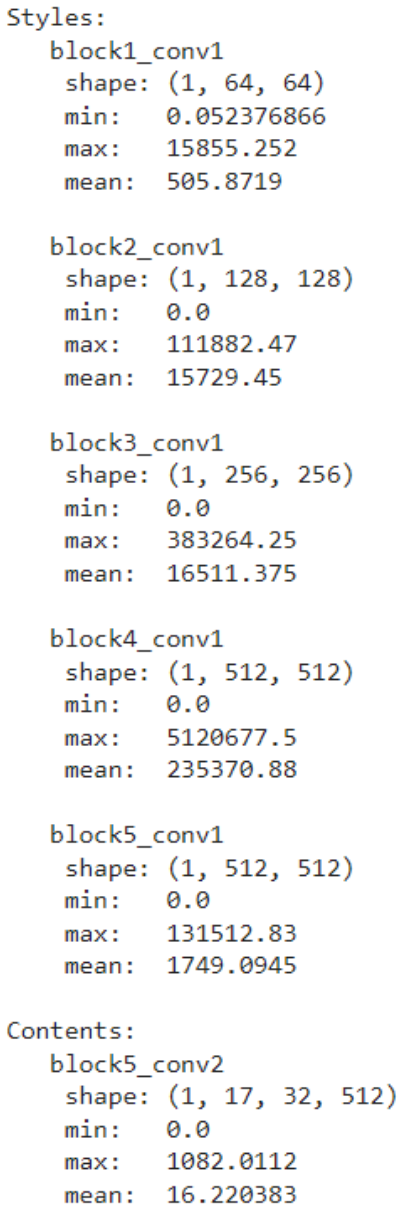

Figure 20.1 Output Information of the Content and Style for Artwork 2

Here, the information about the content and style image were displayed for both Artworks 1 and 2. Since the model and layers used for both of the artworks are the same, the values for both of the content and style image at each specified layer does not differ from one another.

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


image= tf.Variable(base_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
```

Figure 21. Obtaining the Target Content and Style

Using the extractor function, the target content from style_image is acquired and pass to the style_targets variable name. The same process is applied for the base_image but were passed to the content_targets variable name. Figure 21 also defines a function where the pixel values is being kept to 0 and 1.

```
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
```

Figure 22. Calculation of Content and Style Loses

Figure 22 allows the student to calculate how much loss is in the Content and Style. It basically calculates the how different the generated image to its style features.

```
@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))
```

Figure 23. train_step Function

The function for this figure performs gradient calculation thus, updates the pixel values of the image for every train step epoch.

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

Figure 24. Implementing the train_step Function to Generate an Image

Figure 25. Output of the Image after using train_step function.

```
import time
start = time.time()
epochs = 100
steps_per_epoch = 100
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))
```

Figure 26. Generating an Output image without Variation Loss

The student tried multiple values of epoch and through observation, the run time increases as the number epoch values increases. For both the artworks, epochs are set to 100. Lower values of epochs only results into much nosier images.

Figure 26.1. Output using Artwork 1 of Neural Style Transfer without including the total variation weight and image.total_variation

Figure 26.2. Output using Artwork 2 of Neural Style Transfer without including the total variation weight and image.total_variation

Both the figures 26.1 and 26.2 produce a lot of high frequency artifacts. This result to unclear image where small dots is noticeable to the output images generated by the model. To further optimize the output images, steps below are necessary to lessen produce a clear image.

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

    return x_var, y_var
```

Figure 27. High Frequency Explicit Regularization

Here, a high frequency explicit regularization term is applied to the high frequency components of the image This function also helps to find the difference between the neighboring pixels of the image.

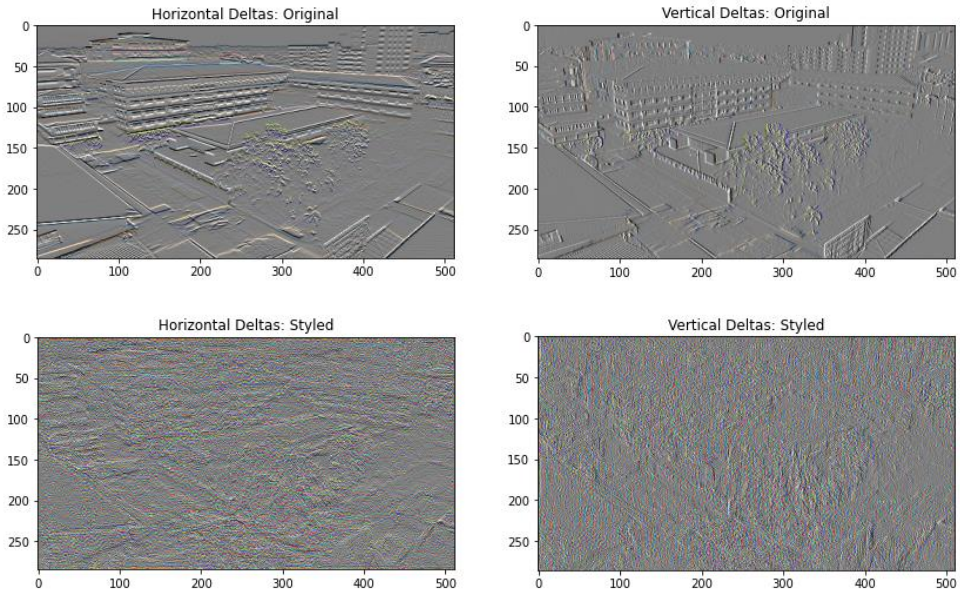

Figure 27.1. Horizontal and Vertical High Frequency Components for Original image and Styled Image

```
plt.figure(figsize=(14, 10))
sobel = tf.image.sobel_edges(base_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")
```

Figure 28. Using Sobel Edge Detector

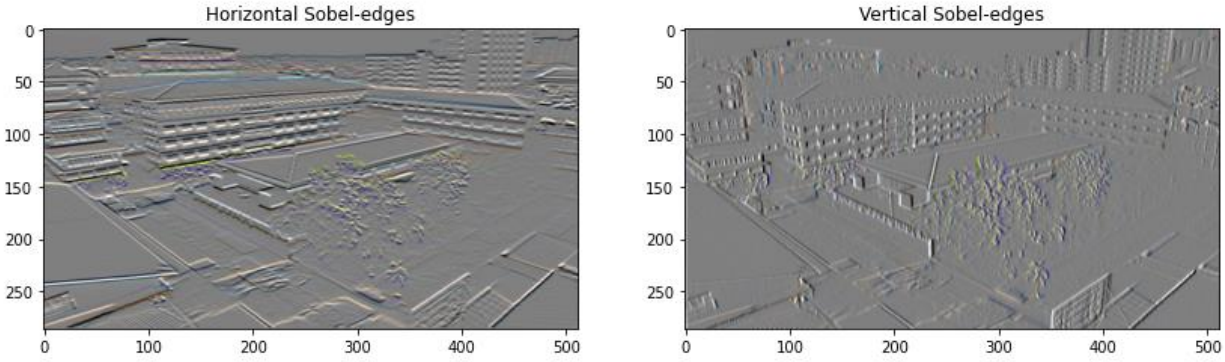

Figure 28.1. Output using Sobel Edge Detector

The output of both the Horizontal and Vertical image also capable in detecting the edges of the image.

```
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()

153653.66


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

array([153653.66], dtype=float32)


total_variation_weight=30
```

Figure 29. The Variation Loss Function.

Figure 29 shows a function for getting the total variation loss of the image. This helps to further optimize the image through identification of the rate of change using the function defined in figure 27. Additionally, total variation weight is adjusted to a value of 30.

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

Figure 30. Reinitializing the Optimizer and Image Variable

```
import time
start= time.time()
epochs= 100
steps_per_epoch= 100
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))
```

Figure 31. Generating an Output image with applied Variation Loss

```
Train step: 10000
Total time: 551.0
```

Figure 31.1. Neural Style Transfer Image using Artwork 1

```
Train step: 10000
Total time: 545.4
```

Figure 31.2. Neural Style Transfer Image using Artwork 2

Conclusion:

In this activity, two artworks with different art styles are used to apply their styles to the base image as shown in figure 1. With the application of variation loss to the image, it lessens the noise, giving a much clearer image to the generated neural style transfer output. Additionally, increasing the number of epochs provides a better quality of image but this will take longer time as a consequence of increasing the number of epoch values.

This activity allows the student to learn how the neural style transfer works and how VGG19 is implemented to use in the said activity. Using E-GPU of google collab really helps to complete the activity and generate a result with less workload to the student's device and faster time. Although the neural style transfer image can get better by configuring the values in the algorithm, it is still shown in the result the characteristic of the artwork which is applied to the base phot used for the activity.

References:

- Amorsolo, Fernando Cueto. (1939). *View of Bataan from Manila Bay*. [Oil in Canvas]. London, United Kingdom. https://www.christies.com/en/lot/lot-5691229
- *Convolution.* (2022). Wolfram Mathworld. https://mathworld.wolfram.com/Convolution.html
- Dr. Sec Infor. (2021). VGG-19 Convolutional Neural Network. Blog.techcraft. https://blog.techcraft.org/vgg-19-convolutional-neural-network/
- Gauguin, Paul. (1893). *Tahitian Landscape with a Mountain*. [Oil in Canvas]. France. https://www.pinterest.ph/pin/450360031487940397/
- Leon Gatys, Alexander Ecker, Matthias Bethge; A Neural Algorithm of Artistic Style . Journal of Vision 2016;16(12):326. doi: https://doi.org/10.1167/16.12.326.
- TensorFlow. (2022). Neural style transfer. TensorFlow. https://www.tensorflow.org/tutorials/generative/style_transfer