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

College of Engineering and Architecture
Electronics Engineering Department

Homework II

NEURAL STYLE TRANSFER

Submitted by:
Corcuera, Vincent Cyrus C.
ECE41S1

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

October 2022

We are tasked to optimize an output image to blend 2 images / pictures which is the content image which in this case. The picture of Technological Institute of the Philippines.

I choose 2 artists and their names are **Vincent Van Gogh** and **Benedicto Cabrera**

The first thing we do is to define, import and configure libraries that we need for the project.

```
[ ] 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
```

Setup the tensor, download the content image and the style image from the chosen artist. In this case, I use Vincent Van Gogh and Benedicto Cabrera's style

```
[ ] 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/Technological_Institute_of_the_Philippines_Quezon_City.jpg', 'https://dr
style_path = tf.keras.utils.get_file('/content/download.jpg','https://drive.google.com/drive/folders/1__mjFqKw5Z3UhNoyumK
```

Next step will be printing of two image to visualize the output. In order for us to achieve this, we need to define first a function to load each image and limit the dimension of the pixels.

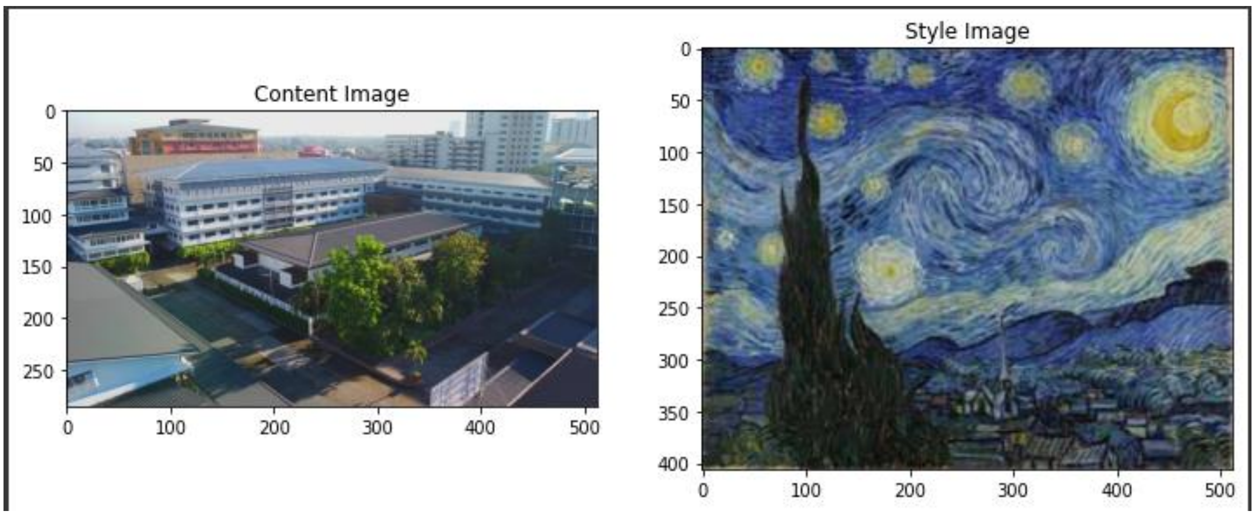
```
def load_img(path_to_img):
  max_dim = 512
  img = tf.io.read_file(path_to_img)
  img = tf.image.decode_image(img, channels=3)
  img = tf.image.convert_image_dtype(img, tf.float32)

  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
```

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

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

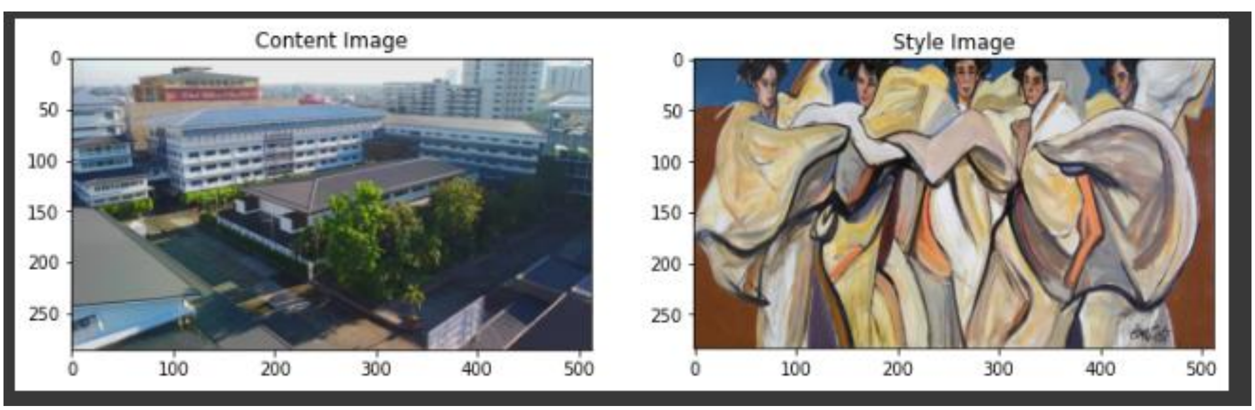
```
[ ] content_image = load_img(content_path)
    style_image = load_img(style_path)

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

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

The pictures below are the result of the content that we are doing in the previous steps.

The below pictures are the Technological Institute of the Philippines and Vincent Van Gogh's Starry Night.



The below pictures are the Technological Institute of the Philippines and Ben Cabrera's The Enduring Legacy.



After that, Use VGG19 network architecture to test and run with the images for us to make sure that it is the right picture or image.

```
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
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf
574710816/574710816 [==============================] - 3s 0us/step
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]
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/imagenet_class_index.json
35363/35363 [==============================] - 0s 0us/step
[('dock', 0.10787788),
 ('dam', 0.09949343),
 ('pier', 0.07090867),
 ('garbage_truck', 0.046317216),
 ('crane', 0.038242985)]
```

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

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

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_
80134624/80134624 [==============================] - 1s 0us/step

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
```

```
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)
```

For building the model, use Keras functional API (tf.keras.applications). It will extract the intermediate layer values.

We also must specify the inputs since we will use the functional API. That is the reason why we put model = Model(inputs, outputs).

```
[ ] 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
```

```
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()
```

This will give us an output of:

```
block1_conv1
  shape:  (1, 406, 512, 64)
  min:  0.0
  max:  669.6255
  mean:  20.270468

block2_conv1
  shape:  (1, 203, 256, 128)
  min:  0.0
  max:  2234.299
  mean:  119.44729

block3_conv1
  shape:  (1, 101, 128, 256)
  min:  0.0
  max:  6964.2783
  mean:  113.765366

block4_conv1
  shape:  (1, 50, 64, 512)
  min:  0.0
  max:  14064.77
  mean:  455.06958

block5_conv1
  shape:  (1, 25, 32, 512)
  min:  0.0
  max:  3005.2104
  mean:  37.969402
```

Use tf.linalg.einsum function for our system to implement the Gram matrix calculation concisely.

```
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)
```

For extracting the style and content, build a model that returns the tensors of contents and style.

```
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}
```

```
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:  0.052376863
    max:  15855.283
    mean:  505.8719

   block2_conv1
    shape:  (1, 128, 128)
    min:  0.0
    max:  111882.47
    mean:  15729.449

   block3_conv1
    shape:  (1, 256, 256)
    min:  0.0
    max:  383264.16
    mean:  16511.375

   block4_conv1
    shape:  (1, 512, 512)
    min:  0.0
    max:  5120678.5
    mean:  235370.88

   block5_conv1
    shape:  (1, 512, 512)
    min:  0.0
    max:  131512.8
    mean:  1749.0945
```

```
Contents:
   block5_conv2
    shape:  (1, 17, 32, 512)
    min:  0.0
    max:  1082.0116
    mean:  16.22038
```

```
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
```

Also, you can see above that we implement style transfer algorithm by running gradient descent. These algorithms will initialize the matching of images.

```
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
```

```python
@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))
```

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

Run a much longer optimization to get the initial output. To clean up the output check for total variation loss and rerun the optimization.

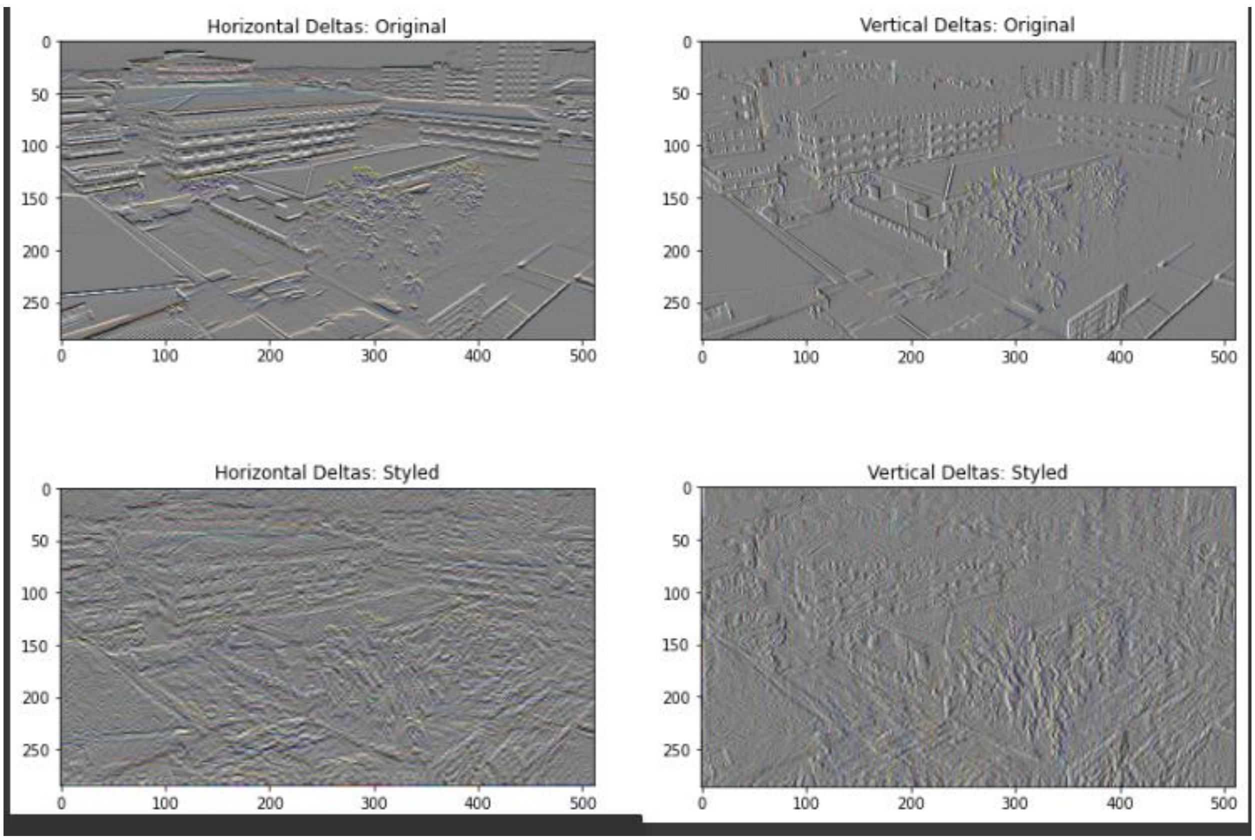
```python
import time
start = time.time()

epochs = 10
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))
```

Train step: 1000
Total time: 4607.4

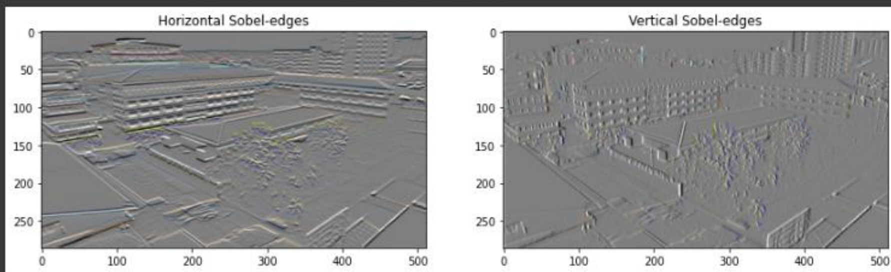
Train step: 1000
Total time: 4206.8

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

  return x_var, y_var
```

```
[ ] 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()
```

```
66701.3
```

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

```
array([66701.3], dtype=float32)
```

```
##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)
```

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

epochs = 10
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))
```

Lastly, Use this to save output.

```
Train step: 1000
Total time: 4634.2
```

```python
file_name = 'VanGogh.png'
tensor_to_image(image).save(file_name)
```

```
Train step: 400
......................
```

```python
file_name = 'Bencab.png'
tensor_to_image(image).save(file_name)
```