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

**COLLEGE OF ENGINEERING AND ARCHITECTURE**
**ELECTRONICS ENGINEERING DEPARTMENT**

**COE 005 – Prediction and Machine Learning**
**ECE41S11**

# Neural Style Transfer

**Homework 2**

**Submitted to:**

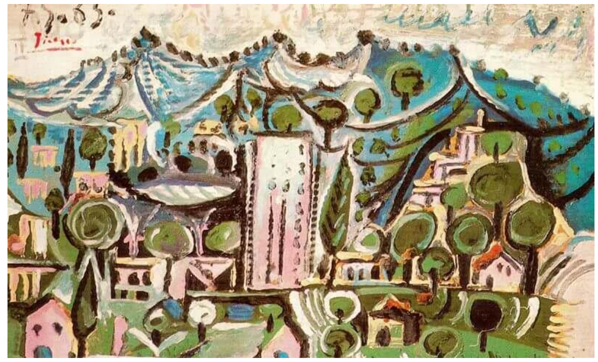
**Engr. Christian Lian Paulo P. Riofloride, MSEE**

**Submitted by:**
**John Andrei Mercado**
**BSECE – ECE41S11**

**Submitted on:**
**October 15, 2022**

Content and Style Images used for Neural Style Transfer

Arno Valley Landscape, 1473
by Leonardo Da Vinci



Landscape Mougins, 1965
by Pablo Picasso

Technological Institute of the Philippines
Quezon City
Photo A

Content Image with transferred style from source image

A.)

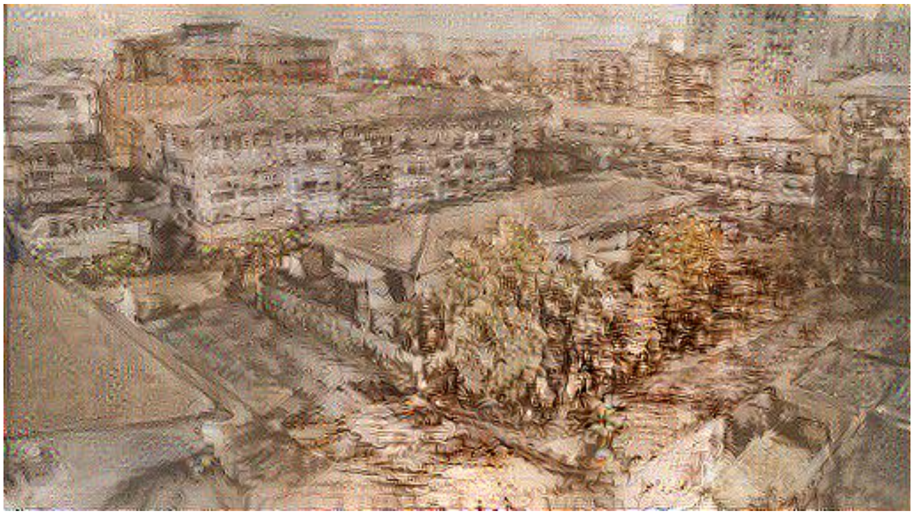


Photo A with Arno Valley Landscape art style

B.)

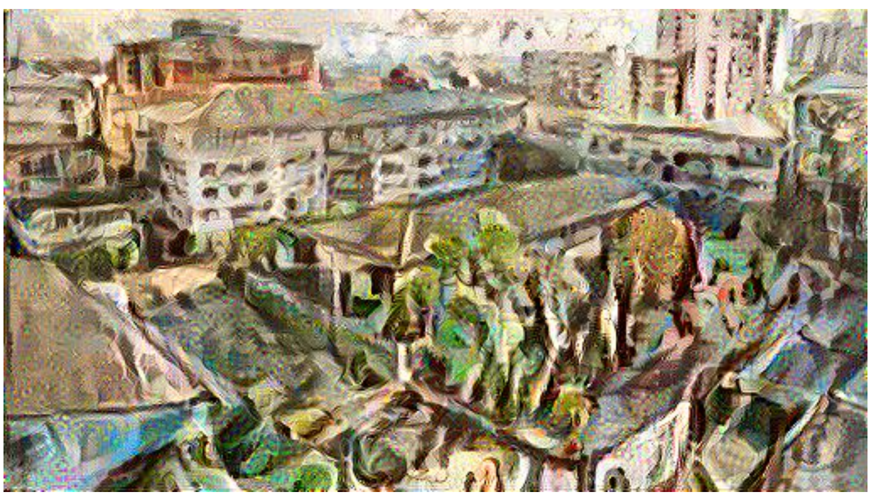


Photo A with Landscape Mougins Art Style

In this activity, a demonstration of neural style transfer was conducted. Neural Style Transfer is an optimization technique which uses deep learning to extract a style from an image called “style image”, then superimposes it to the targeted content image which results to the content image being “drawn” in the art style of the style image. In this activity, 2 style images, from Pablo Picasso and Leonardo Da Vinci, with the goal of using Neural Style Transfer to superimpose it to the photo of the Technological Institute of the Philippines – Quezon City Campus.

As seen above, 2 style images with different art style were used. The criteria personally used in choosing the 2 styles were opposing styles. Da Vinci’s art style was more on utilizing pen and eraser sketch techniques to create distinct details while recreating the scenery. On the other hand, Picasso’s art for this chosen image was using distinctive water color techniques to recreate the scenery. Objects were highlighted through the use of color to create depth.

The following are the machine learning code used to perform Neural Style Transfer with Google Colab as the compiler. (See bottom for reference).

Initial part of the code is to call the general libraries to perform the different functions later on related to Neural Style Transfer.

```
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
```

Next, the source images were also imported. Functions used to process these content images were also defined.

```
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 = "tipqc.jpg"
style_path= "leo.jpg"
style_path2= "pablo1.jpg"
```

```
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
```

The source images were then assigned to their syntax and is then displayed using matplotlib.

```
content_image = load_img(content_path)
style_image = load_img(style_path)
style_image2 = load_img(style_path2)


plt.subplot(1, 3, 1)
imshow(content_image, 'Content Image')

plt.subplot(1, 3, 2)
imshow(style_image, 'Style Image')

plt.subplot(1, 3, 3)
imshow(style_image2, 'Style Image 2')
```


Now, VGG19 network architecture will be used in processing the images through multiple model layers. With the initial layers dedicated for the low-level features such as edges and textures, with every succeeding layer capturing higher feature levels as it goes on.

```
[ ] 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_kernels.h5
    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', 0.107877925),
     ('dam', 0.09949322),
     ('pier', 0.07090873),
     ('garbage_truck', 0.046317235),
     ('crane', 0.038242817)]
```

The layers are then assigned to the corresponding to the inputs that they will be receiving such as content and style layers. These are important in order for the algorithm to understand the image which will play a vital role later in conducting the style transfer.

```
[ ] 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_kernels_notop.h5
    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)
```

The model is then defined and created.

```
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()
```

Then, the style of the image is described using this function.

```
[ ]  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)
```

The functions for extracting the features of the content and style images are then defined. This is used later on for transferring the features.

```
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())
```

The process of neural style transfer is now performed using the extract function. The optimizer is also defined.

```
[ ] style_targets = extractor(style_image2)['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
```

Now, Gradient Tape function is used to update the content image with the extracted features from the style image. A few steps were performed to test if the function is properly working. This resulted to an image with a poor style transfer accuracy.

```
@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))
```

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

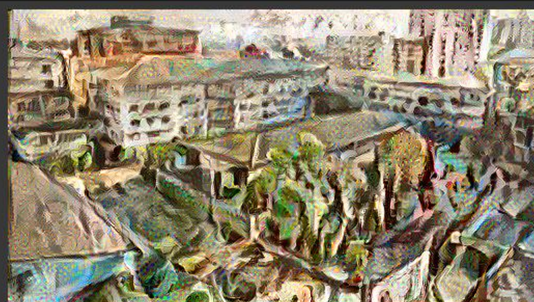
A function to conduct multiple steps over multiple epochs is created to further optimize the neural transfer, this resulted to a better style transfer over iterations as seen from the photo below.

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

    epochs = 50
    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: 5000
Total time: 297.7
```

In conclusion, Neural Style Transfer uses tensorflow to process images into numerical data that machines may understand. With the proper defined functions, Neural Style Transfer is performed with the capability of training the model over continuous iterations in order to produce a much more accurate and “eye-pleasing” style transfer. With the evolution of technology, Machine Algorithms and AI have started to integrate itself to the industry of art. With algorithms producing its own different styles and techniques learned using the art from way before, it is quite intriguing how the art industry will react when such techniques start to overwhelm the traditional art we’ve always been accustomed to.

References:

[https://www.tensorflow.org/tutorials/generative/style_transfer](https://www.tensorflow.org/tutorials/generative/style_transfer)
[https://www.leonardodavinci.net/landscape-drawing-for-santa-maria-della-neve.jsp](https://www.leonardodavinci.net/landscape-drawing-for-santa-maria-della-neve.jsp)
https://www.pablopicasso.org/landscape-mougins.jsp