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

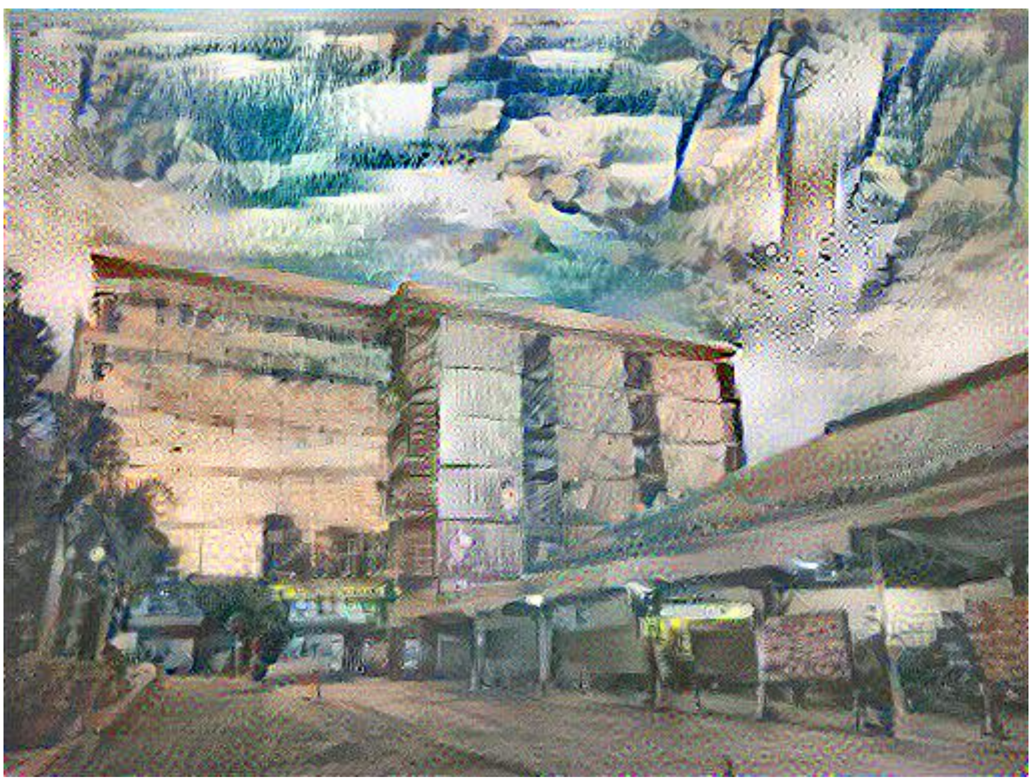
Homework 2
Neural Style Transfer
Submitted to:
Engr. Christian Lian Paulo Riofloridio

Submitted on:
10/17/2022

Submitted by:
John Michael Valad-on

Author: Ukiyo-e
Title: From Thirty-six Views of Mount Fuji

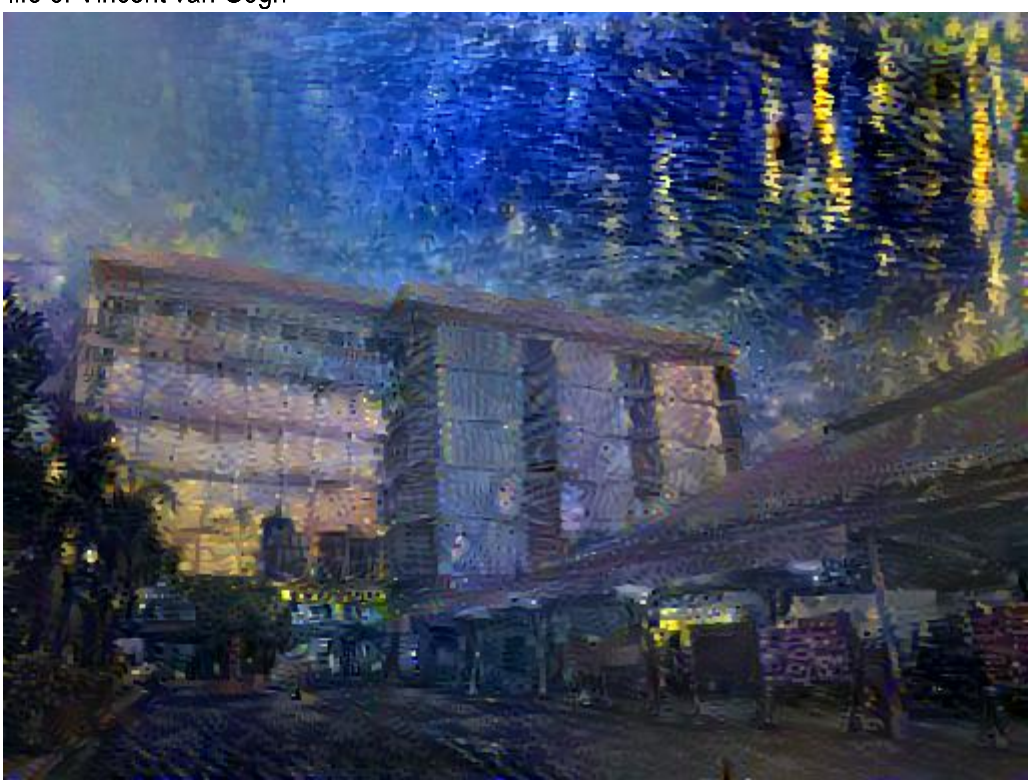
Content Image (TIP Building)



(Style Transfer output 1)

Author: Vincent Van Gogh
Title: The life of Vincent van Gogh

Content Image (TIP Building)



(Style Transfer output 2)

Screenshot of Codes

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

```python
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)
```

(Import Necessary Libraries)

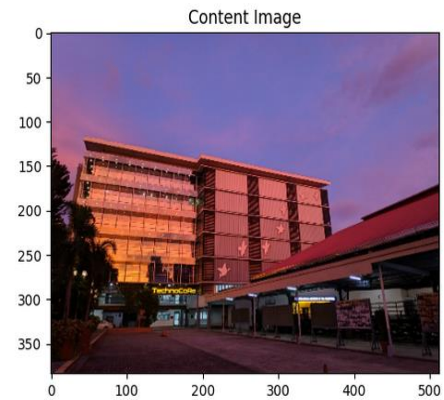
PIL.image and matplotlib is necessary to visualize the data while tensorflow was used for image processing. functools also is to allow to fixed the values of parameters of images and time is to represent the time of training of epochs. On the other hand, Tensor to image is one of the essential codes for this program since it transforms the image into a tensor to be able to train our models the content image to style image. we can also manipulate the image in a way that is useful to us.

```python
content_path = "tip1.jpg"
content2_path = "tip2.jpg"
style_path =   "hok.jpg"
style2_path = "hok2.jpg"
```

(Gather and import the content and style 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
```

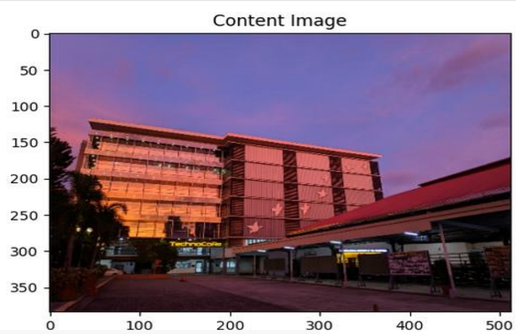
```python
content_image = load_img(content_path)
style_image = load_img(style_path)

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

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

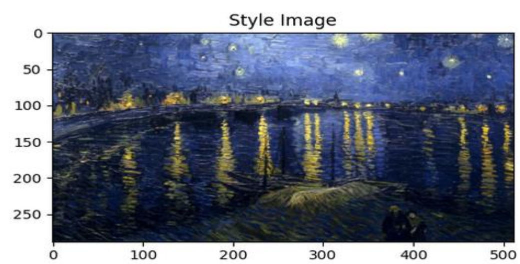



```python
def show_image(image,title=None):
    if len(image.shape)>3:
        image=tf.squeeze(image,axis=0)
    plt.imshow(image)
    if title:
        plt.title=title
```

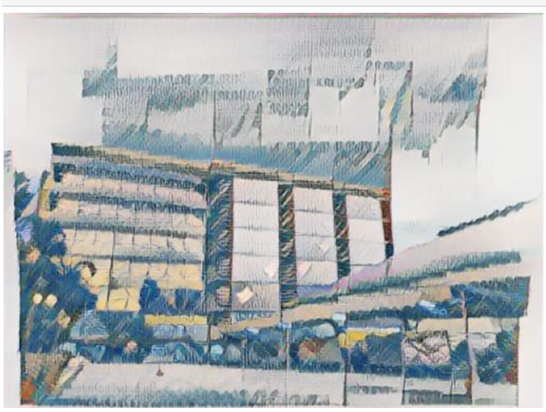
```python
def imshow(image, title=None):
  if len(image.shape) > 3:
    image = tf.squeeze(image, axis=0)

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

```python
content2_image = load_img(content2_path)
style2_image = load_img(style2_path)

plt.subplot(1, 2, 1)
imshow(content2_image, 'Content Image')

plt.subplot(1, 2, 2)
imshow(style2_image, 'Style Image')
```

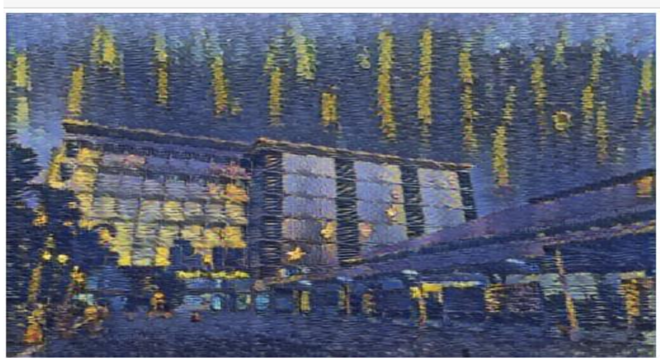





```python
import tensorflow_hub as hub
hub_model = hub.load('https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/2')
stylized_image = hub_model(tf.constant(content_image), tf.constant(style_image))[0]
tensor_to_image(stylized_image)
```



```python
import tensorflow_hub as hub
hub_model = hub.load('https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/2')
stylized2_image = hub_model(tf.constant(content2_image), tf.constant(style2_image))[0]
tensor_to_image(stylized2_image)
```



Visualization of the images

```
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', 0.2946157),
 ('library', 0.040501934),
 ('fire_engine', 0.039418653),
 ('planetarium', 0.03768352),
 ('freight_car', 0.032800265)]
```

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

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

```
content_layers = ['block5_conv2']

content2_layers = ['block5_conv2']

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

style2_layers = ['block1_conv1',
                'block2_conv1',
                'block3_conv1',
                'block4_conv1',
                'block5_conv1']

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

num_content2_layers = len(content2_layers)
num_style2_layers = len(style2_layers)
```

```
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
```

(Choosing the Model)

VGG19 network architecture is an intermediate layer that represents features like edges and textures in the first layer. It consists of 19 layers and 5 maxpool layers. These layers are where we load our image to trained from its database. VGG19 is a good model to use in this activity since it have 19 layers, pretrained network and million images from database compared to VGG16 however VGG19 is much slower to trained compared to VGG16 but produce high quality result.

```
extractor = StyleContentModel(style_layers, content_layers)

results = extractor(tf.constant(content_image))
```

```
extractor2 = StyleContentModel(style2_layers, content_layers)

results2 = extractor(tf.constant(content_image))
```

```
style_targets = extractor(style_image)['style']
content_targets = extractor(content_image)['content']
image1 = tf.Variable(content_image)

style2_targets = extractor2(style2_image)['style']
content2_targets = extractor2(content2_image)['content']
image2 = tf.Variable(content2_image)
```

```
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)
```

```
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(inpu
    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}
```

```
class StyleContentModel(tf.keras.models.Model):
  def __init__(self, style2_layers, content2_layers):
    super(StyleContentModel, self).__init__()
    self.vgg = vgg_layers(style2_layers + content2_layers)
    self.style2_layers = style2_layers
    self.content2_layers = content2_layers
    self.num_style2_layers = len(style2_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)
    style2_outputs, content2_outputs = (outputs[:self.num_style2_layers],
                                        outputs[self.num_style2_layers:])

    style2_outputs = [gram_matrix(style2_output)
                      for style2_output in style2_outputs]

    content2_dict = {content2_name: value
                     for content2_name, value
                     in zip(self.content2_layers, content2_outputs)}

    style2_dict = {style2_name: value
                   for style2_name, value
                   in zip(self.style2_layers, style2_outputs)}

    return {'content': content2_dict, 'style': style2_dict}
```

(Building of Model

The content of the images is represented by values as it can described by the means and correlation. A gram matrix is able to calculate the outer product of the vector of all location which is important to extract the style of the image to create a texture information for the data.

```
def clip_0_1(image1):
  return tf.clip_by_value(image1, clip_value_min=0.0, clip_value_max=1.0)

def clip_0_1(image2):
  return tf.clip_by_value(image2, 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

opt = tf.keras.optimizers.Adam(learning_rate=0.02, beta_1=0.99, epsilon=1e-1)
style2_weight=1e-2
content2_weight=1e4
def style2_content2_loss2(outputs2):
    style2_outputs2 = outputs2['style']
    content2_outputs2 = outputs2['content']
    style2_loss = tf.add_n([tf.reduce_mean((style2_outputs2[name]-style2_targets[name])**2)
                           for name in style2_outputs2.keys()])
    style2_loss *= style2_weight / num_style2_layers

    content2_loss = tf.add_n([tf.reduce_mean((content2_outputs2[name]-content2_targets[name])**2)
                             for name in content2_outputs2.keys()])
    content2_loss *= content2_weight / num_content2_layers
    loss2 = style2_loss + content2_loss
    return loss2
```

(Run Gradient Descent)

Implementation of the style transfer algorithm by setting up target values of the style and content extractor.

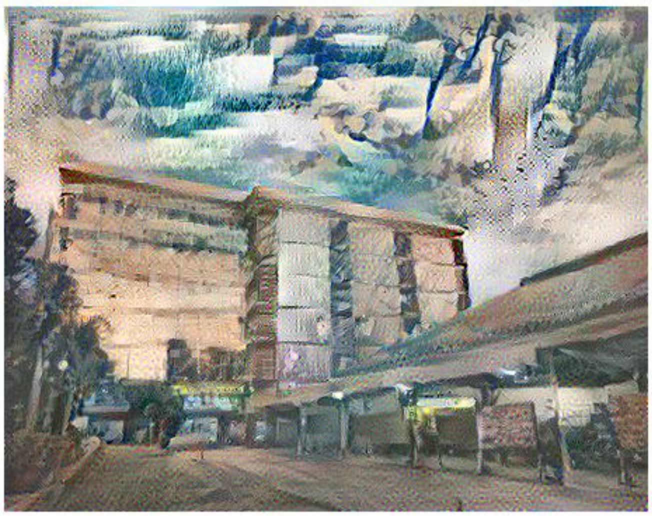
```
total_variation_weight=30
```

```
@tf.function()
def train1_step(image1):
  with tf.GradientTape() as tape:
    outputs = extractor(image1)
    loss = style_content_loss(outputs)
    loss += total_variation_weight*tf.image.total_variation(image)

  grad = tape.gradient(loss, image1)
  opt.apply_gradients([(grad, image1)])
  image1.assign(clip_0_1(image1))

@tf.function()
def train2_step(image2):
  with tf.GradientTape() as tape:
    outputs2 = extractor2(image2)
    loss2 = style2_content2_loss2(outputs2)
    loss2 += total_variation_weight*tf.image.total_variation(image2)

  grad2 = tape.gradient(loss2, image2)
  opt.apply_gradients([(grad2, image2)])
  image2.assign(clip_0_1(image2))
```

(optimization and train of images)

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

```
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
    train1_step(image1)
    print(".", end='', flush=True)
  display.clear_output(wait=True)
  display.display(tensor_to_image(image1))
  print("Train step: {}".format(step))

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



```
Train step: 1000
Total time: 158.8
```

```
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
    train2_step(image2)
    print(".", end='', flush=True)
  display.clear_output(wait=True)
  display.display(tensor_to_image(image2))
  print("Train step: {}".format(step))

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

```
Train step: 1000
Total time: 160.2
```

(Optimized Style and Content Images)

This shows that the final output result has better image output. The optimization is done by total variation loss wherein it optimizes the style and content image loss led to highly pixelated and noisy output of some part of the image. additionally, I observed that The higher the epoch the higher the quality of style image into content image however, the lower the epoch the lower also the accuracy of style transfer such as it will have blurred photo everywhere in the output image and also it will have glitches at some part.

Discussion

A neural style transfer combines two images, content and style images. As a result, the combined image will look like the content image. However, the representation of it will be the style image as both pictures have mixed and blended to be a new artwork. It is a fun and exciting technique that showcases the capabilities of the neural network in creating artistic images.

The implementation of the code is. First, I gathered data by choosing the style images I wanted to transfer to the content images. Then importing the libraries will be needed as well as pathing the images. Second, I prepared the data by visualizing images and getting content and style representation. In this case, I use VGG19, a relatively simple model that works better in style transfer since it can classify up to 1000 objects. The third is to build the model by configuring the optimizer and loss function, creating a gram matrix, vgg19 with TensorFlow Keras, picking intermediate layers to represent the style and content of the image, and defining the custom train step and training loop. Vgg19 consists of 5 blocks, and I use conv1 layers from every block for style. Next is the gram matrix for the loss calculation of the image. To create an extractor that contains content and style information output values. After that, is configure our model and create a custom loss function. Total variation weight for obtaining different style transfer results and GradientTape to record loss of input image. Lastly is the custom training loop, which can optimize by running the model in 10 epochs and 100 for steps_per_epoch, which means that the total training time is 1000 because of epoch x steps per epoch. The loop also will give us a temporarily stylized image until it gets to 1000 training time for the final output.

In the given activity, the first style I chose was painted by Ukiyo-e and titled From Thirty-six Views of Mount Fuji. The output is that it has the same background color and lighting between style images while having the exact contents of the content image. I also noticed a blurred image of a tree and mountain at the top of the image, similar to the stylized image. For the second style image, I chose the life of vincent van Gogh, painted by vincent van Gogh. The output result is the lightness, and background color of the picture have been similar to the stylized image. It also has some yellow light reflection that we can see at the top of the output results coming from the stylized image, which we can see in its middle part. Both output results look like a painting painted by the author of a stylized image.