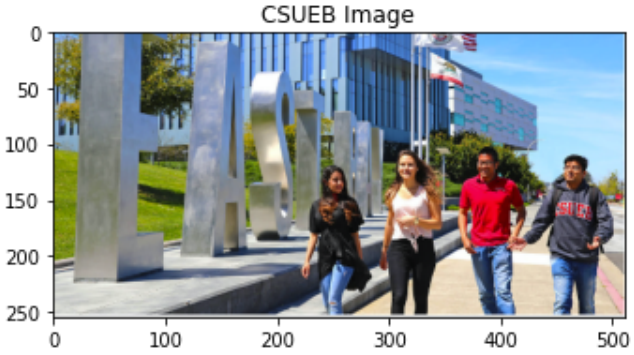
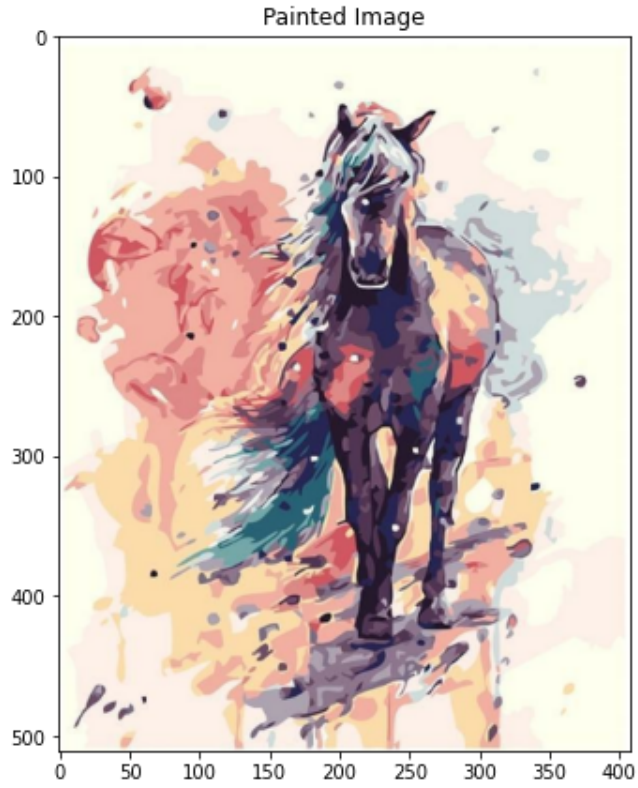
```
import os
import tensorflow as tf
# Load compressed models from tensorflow_hub
os.environ['TFHUB_MODEL_LOAD_FORMAT'] = 'COMPRESSED'
import IPython.display as display
import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.rcParams['figure.figsize'] = (12,12)
mpl.rcParams['axes.grid'] = False
import numpy as np
import PIL.Image
import time
import functools


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)


from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mou
```

```
csueb_path = '/content/students-walking-on-campus.png'
painted_path = '/content/horse_painting.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

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(csueb_path)
style_image = load_img(painted_path)
plt.subplot(1, 2, 1)
imshow(content_image, 'CSUEB Image')
plt.subplot(1, 2, 2)
imshow(style_image, 'Painted Image')
```






```
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)
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
predicted_top_5 = tf.keras.applications.vgg19.decode_predictions(prediction_probabilities.num
[(class_name, prob) for (number, class_name, prob) in predicted_top_5]
print()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg1
574717952/574710816 [==============================] - 5s 0us/step
```

```
574726144/574710816 [==============================] - 5s 0us/step
Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/image
40960/35363 [================================] - 0s 0us/step
49152/35363 [==========================================] - 0s 0us/step
```

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

```
input_1
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
flatten
fc1
fc2
predictions
```

```
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)
```

```
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()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg1
80142336/80134624 [==============================] - 1s 0us/step
80150528/80134624 [==============================] - 1s 0us/step
block1_conv1
  shape:  (1, 511, 408, 64)
  min:  0.0
  max:  875.37885
  mean:  35.754337

block2_conv1
  shape:  (1, 255, 204, 128)
  min:  0.0
  max:  4549.171
  mean:  197.31772

block3_conv1
  shape:  (1, 127, 102, 256)
  min:  0.0
  max:  11595.237
  mean:  185.18872

block4_conv1
  shape:  (1, 63, 51, 512)
  min:  0.0
  max:  24262.912
  mean:  680.86816

block5_conv1
  shape:  (1, 31, 25, 512)
  min:  0.0
  max:  2637.521
  mean:  45.93024
```

```
def gram_matrix(input_tensor):
  result = tf.linalg.einsum('bijc,bijd->bcd', input_tensor, 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(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())
```

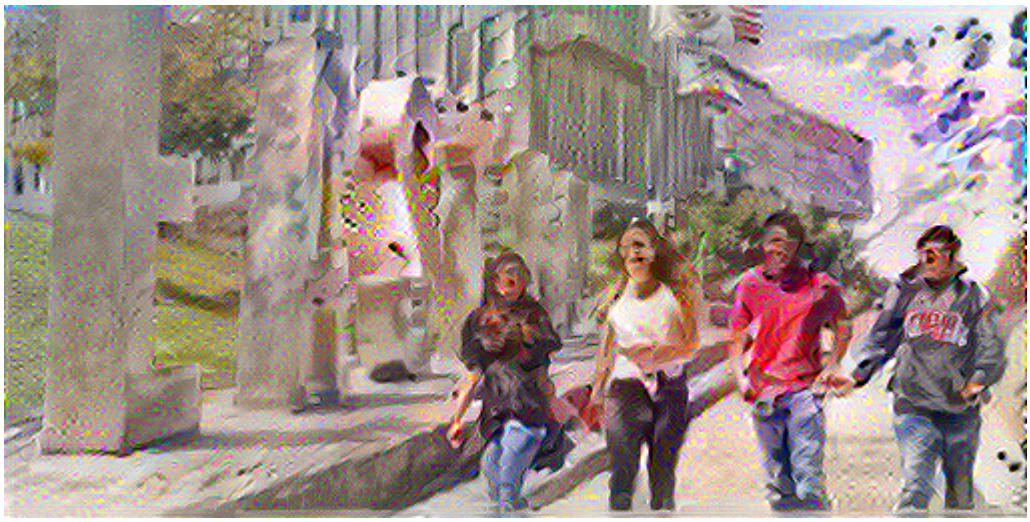
```
      Styles:
         block1_conv1
          shape:  (1, 64, 64)
          min:  0.2309468
          max:  31140.148
          mean:  984.14825

         block2_conv1
          shape:  (1, 128, 128)
          min:  0.0
          max:  239140.47
          mean:  31675.844

         block3_conv1
          shape:  (1, 256, 256)
          min:  0.0
          max:  948065.1
          mean:  40235.055

         block4_conv1
          shape:  (1, 512, 512)
          min:  0.0
          max:  8885830.0
          mean:  542932.6

         block5_conv1
          shape:  (1, 512, 512)
          min:  0.0
          max:  476421.28
          mean:  5435.8745

      Contents:
         block5_conv2
          shape:  (1, 16, 31, 512)
          min:  0.0
          max:  3961.6167
          mean:  29.52333
```

```
  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.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))


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

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

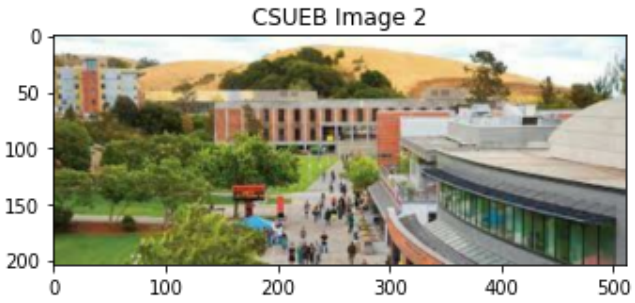
```
start = time.time()

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))
```

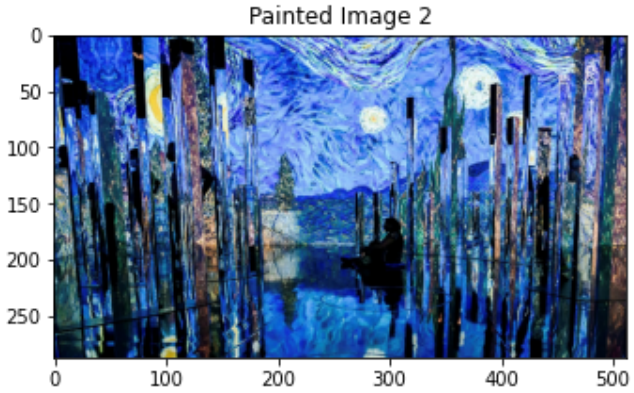


```
Train step: 1000
Total time: 3972.1
```

New Picture

```
csueb_path ='/content/Cal state east bay.jpg'
painted_path ='/content/painting_1.jpg'


content_image = load_img(csueb_path)
style_image = load_img(painted_path)
plt.subplot(1, 2, 1)
imshow(content_image, 'CSUEB Image 2')
plt.subplot(1, 2, 2)
imshow(style_image, 'Painted Image 2')
```






```
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)
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
predicted_top_5 = tf.keras.applications.vgg19.decode_predictions(prediction_probabilities.num
[(class_name, prob) for (number, class_name, prob) in predicted_top_5]
print()
```

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

```
input_4
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
        flatten
        fc1
        fc2
        predictions


    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)


    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()

        block1_conv1
          shape:  (1, 288, 512, 64)
          min:  0.0
          max:  824.818
          mean:  34.34097

        block2_conv1
          shape:  (1, 144, 256, 128)
          min:  0.0
          max:  4492.1304
          mean:  221.71385

        block3_conv1
          shape:  (1, 72, 128, 256)
          min:  0.0
          max:  9234.559
          mean:  234.34587

        block4_conv1
          shape:  (1, 36, 64, 512)
          min:  0.0
          max:  25270.893
          mean:  850.7247

        block5_conv1
```

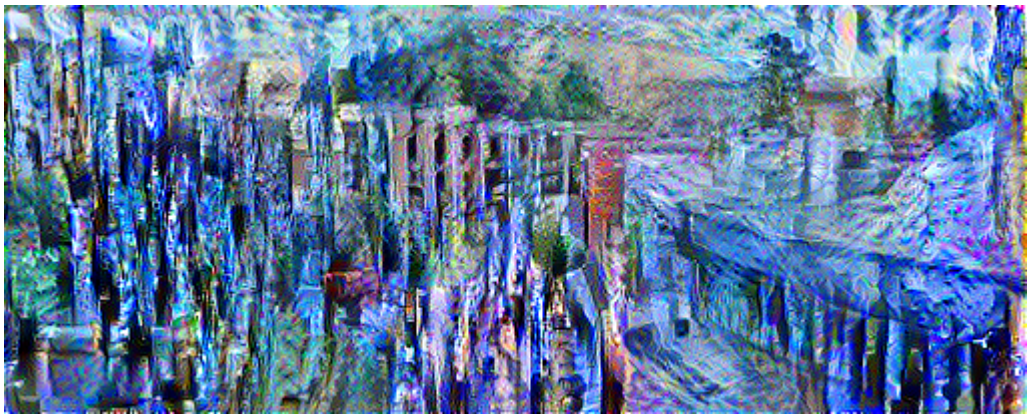
```
    shape:  (1, 18, 32, 512)
    min:  0.0
    max:  3231.2678
    mean:  58.030205
```

```
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:  0.10935415
    max:  36434.26
    mean:  490.3711

   block2_conv1
    shape:  (1, 128, 128)
    min:  0.0
    max:  109473.48
    mean:  17873.855

   block3_conv1
    shape:  (1, 256, 256)
    min:  0.0
    max:  518084.62
    mean:  16948.277

   block4_conv1
    shape:  (1, 512, 512)
    min:  0.0
    max:  4669337.0
    mean:  242333.19

   block5_conv1
```

```
            shape:  (1, 512, 512)
            min:  0.0
            max:  112971.82
            mean:  1811.5908

         Contents:
            block5_conv2
            shape:  (1, 12, 31, 512)
            min:  0.0
            max:  1096.959
            mean:  16.499222


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


    image2 = tf.Variable(content_image)


    opt = tf.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))


    train_step(image2)
    train_step(image2)
    train_step(image2)
    tensor_to_image(image2)
```

```
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(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: 3192.3
```

✓ 53m 12s completed at 9:48 PM