Image Search By an Artistic Style

This notebook demonstrates how images can be searched by style. One typical business case is an eCommerce webiste that allows to search a poster based on the example uploaded by the user. A user would usually expect to get results that are similar in terms of artisitc style. A real search system would typically mix style similarity with other similarity score such as image subject or category (landscape, still life, etc.) that can also be obtained using deep neural networks.

The implementation is based on a TensorFlow tutorial on Neural Style Transfer [1] which is in turn based on a seminal paper A Neural Algorithm of Artistic Style by Gatys et al. [2].

Data

We use simple dataset with 32 images from **tensor-house-data** repository.

References

1. https://www.tensorflow.org/tutorials/generative/style_transfer
2. Gatys L., Ecker A., Bethge M. -- A Neural Algorithm of Artistic Style, 2015


```
import tensorflow as tf
from tensorflow.keras.applications.vgg19 import preprocess_input
from tensorflow.keras.models import Model
print(tf.__version__)

import matplotlib.pyplot as plt
import numpy as np
from tqdm import tqdm

import glob
import ntpath
import cv2

from sklearn.metrics.pairwise import cosine_similarity
import scipy as sc
```

```
2.4.1
```

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

```
Mounted at /content/drive
```

```
#image_paths = glob.glob('https://github.com/ikatsov/tensor-house-data/tree/master/search/images-by-style/*.jpg')
#image_paths = glob.glob('https://github.com/ikatsov/tensor-house-data/blob/26b650abb0e4750b3c0a240411312c0f28d5e3b9/search/images-by
image_paths = glob.glob('/content/drive/MyDrive/ADM/images-by-style/*.jpg')
print(image_paths)
```

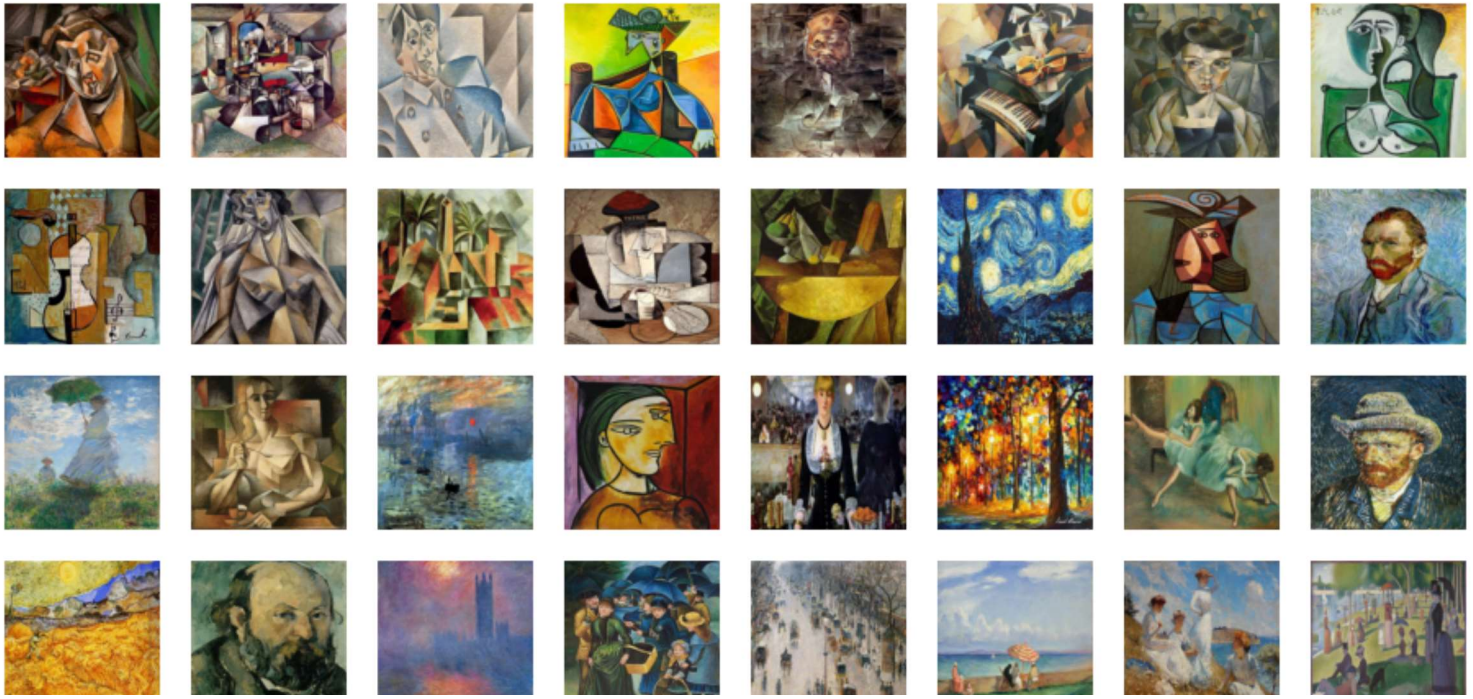
```
['/content/drive/MyDrive/ADM/images-by-style/s_cubism-11.jpg', '/content/drive/MyDrive/ADM/images-by-style/s_cubism-06.jpg', '/
```

```
image_paths = glob.glob('/content/drive/MyDrive/ADM/images-by-style/*.jpg')
print(f'Founnd [{len(image_paths)}] images')


images = {}
for image_path in image_paths:
    image = cv2.imread(image_path, 3)
    b,g,r = cv2.split(image)           # get b, g, r
    image = cv2.merge([r,g,b])         # switch it to r, g, b
    image = cv2.resize(image, (200, 200))
    images[ntpath.basename(image_path)] = image


n_col = 8
n_row = int(len(images)/n_col)
f, ax = plt.subplots(n_row, n_col, figsize=(16, 8))
for i in range(n_row):
    for j in range(n_col):
        ax[i, j].imshow(list(images.values())[n_col*i + j])
        ax[i, j].set_axis_off()
```

```
Founnd [32] images
```



New Section

```
def load_image(image):
  image = plt.imread(image)
  img = tf.image.convert_image_dtype(image, tf.float32)
  img = tf.image.resize(img, [400, 400])
  img = img[tf.newaxis, :] # shape -> (batch_size, h, w, d)
  return img

# content layers describe the image subject
content_layers = ['block5_conv2']

# style layers describe the image style
# we exclude the upper level layes to focus on small-size style details
style_layers = [
        'block1_conv1',
        'block2_conv1',
        'block3_conv1'
```

```
        'block3_conv1',
        #'block4_conv1',
        #'block5_conv1'
    ]

def selected_layers_model(layer_names, baseline_model):
  outputs = [baseline_model.get_layer(name).output for name in layer_names]
  model = Model([vgg.input], outputs)
  return model

# style embedding is computed as concatenation of gram matrices of the style layers
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 StyleModel(tf.keras.models.Model):
  def __init__(self, style_layers, content_layers):
    super(StyleModel, self).__init__()
    self.vgg =  selected_layers_model(style_layers + content_layers, vgg)
    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):
    # scale back the pixel values
    inputs = inputs*255.0
    # preprocess them with respect to VGG19 stats
    preprocessed_input = preprocess_input(inputs)
    # pass through the reduced network
    outputs = self.vgg(preprocessed_input)
    # segregate the style and content representations
    style_outputs, content_outputs = (outputs[:self.num_style_layers],
                                      outputs[self.num_style_layers:])

    # calculate the gram matrix for each layer
    style_outputs = [gram_matrix(style_output)
                     for style_output in style_outputs]

    # assign the content representation and gram matrix in
```

```
    # a layer by layer fashion in dicts
    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}

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

def image_to_style(image_tensor):
    extractor = StyleModel(style_layers, content_layers)
    return extractor(image_tensor)['style']

def style_to_vec(style):
    # concatenate gram matrics in a flat vector
    return np.hstack([np.ravel(s) for s in style.values()])

# compute styles
image_style_embeddings = {}
for image_path in tqdm(image_paths):
    image_tensor = load_image(image_path)
    style = style_to_vec( image_to_style(image_tensor) )
    image_style_embeddings[ntpath.basename(image_path)] = style
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kerne
80142336/80134624 [==============================] - 0s 0us/step
100%|██████████| 32/32 [01:08<00:00,  2.13s/it]
```

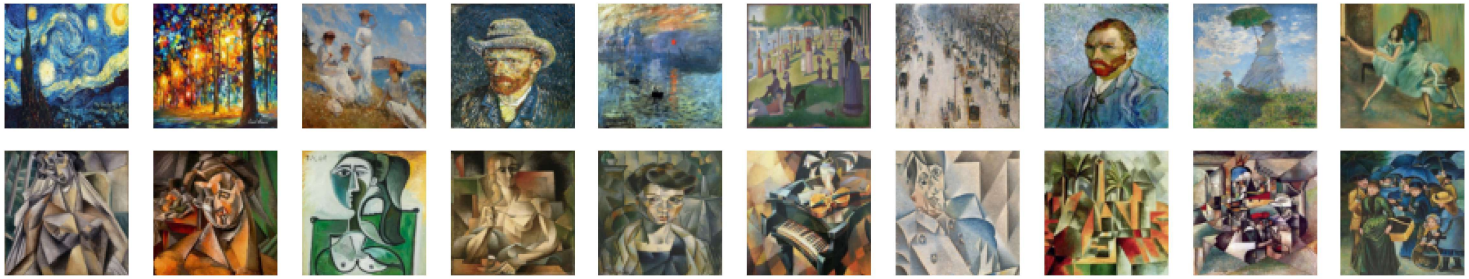
```
def search_by_style(reference_image, max_results=10):
    v0 = image_style_embeddings[reference_image]
    distances = {}
    for k,v in image_style_embeddings.items():
        d = sc.spatial.distance.cosine(v0, v)
        distances[k] = d

    sorted_neighbors = sorted(distances.items(), key=lambda x: x[1], reverse=False)
```

```
    f, ax = plt.subplots(1, max_results, figsize=(16, 8))
    for i, img in enumerate(sorted_neighbors[:max_results]):
        ax[i].imshow(images[img[0]])
        ax[i].set_axis_off()

    plt.show()

# images mostly match the reference style, although not perfectly
search_by_style('s_impressionist-02.jpg')
search_by_style('s_cubism-02.jpg')
```



✓ 1s completed at 5:04 PM