Image feature Extraction Transfer Learning

* Tapan Kumar Patro
* tapankumarpatro05@gmail.com

Work Items:

1.Image Load

2.Feature Extraction for all Image

3.Save into A pickel file

4.Load the pickel file

5.Extract feature from given image

6.Find the minimum distance and return the images

Importing Libraries

```
from __future__ import absolute_import, division, print_function, unicode_litera
# from google_images_download import google_images_download

try:
  # The %tensorflow_version magic only works in colab.
  %tensorflow_version 2.x
except Exception:
  pass
import tensorflow as tf

import os
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

```
tf.__version__
```

```
'2.3.0'
```

```
import os
from tqdm import tqdm
from tensorflow.keras import models, layers
from tensorflow.keras.models import Model
from tensorflow.keras.layers import BatchNormalization, Activation, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2
from tensorflow.keras.layers import Dense,AveragePooling2D,BatchNormalization,Co
```

```
from tensorflow.keras.layers import Dense,AveragePooling2D,BatchNormalization,Co
from time import time
from datetime import datetime
from tensorflow.python.keras.callbacks import TensorBoard
import cv2
```

Setup Google Colab for importing Dataset

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

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

```
# Navigating to Dataset folder in my drive
path = 'drive/My Drive/PocketApps/Avantari/dataset'
os.chdir(path)

```

```
!pwd
```

```
/content/drive/My Drive/PocketApps/Avantari/dataset
```

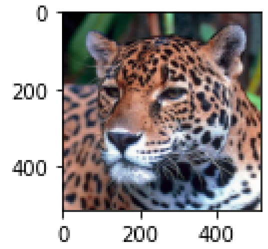
```
# !ls
```

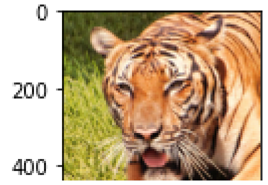
```
# creating a list of all images
all_images = os.listdir()
```

```
# Defining Image Size given in requirement
IMAGE_SIZE = 512
```

```
from PIL import Image, ImageOps
```

```
# Showing some random 3 images to visualize
for i, val in enumerate(all_images[10:13]):
    plt.subplot(1, 3, i+1)
    image_data = Image.open(val)
    plt.imshow(image_data)
    plt.show()
```





Create the base model from the pre-trained convnets

Create the base model from the **MobileNet V2** model developed at Google, and pre-trained on the ImageNet dataset, a large dataset of 1.4M images and 1000 classes of web images.

First, pick which intermediate layer of MobileNet V2 will be used for feature extraction. A common practice is to use the output of the very last layer before the flatten operation, the so-called "bottleneck layer". The reasoning here is that the following fully-connected layers will be too specialized to the task the network was trained on, and thus the features learned by these layers won't be very useful for a new task. The bottleneck features, however, retain much generality.

Let's instantiate an MobileNet V2 model pre-loaded with weights trained on ImageNet. By specifying the `include_top=False` argument, we load a network that doesn't include the classification layers at the top, which is ideal for feature extraction.

```
# Creating Base Model


# Defining Image Shape
IMG_SHAPE = (IMAGE_SIZE, IMAGE_SIZE, 3)


# Create the base model from the pre-trained model MobileNet V2
base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
                                               include_top=False,
                                               weights='imagenet')
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicat
9412608/9406464 [==============================] - 0s 0us/step
```

Feature extraction

You will freeze the convolutional base created from the previous step and use that as a feature extractor, we will train extra 20 more layers to get the features out.

```
# base_model.trainable = False
```

```
base_model.trainable = True

# Let's take a look to see how many layers are in the base model
print("Number of layers in the base model: ", len(base_model.layers))

# Fine tune from this layer onwards
fine_tune_at = 20

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable =  False
```

```
Number of layers in the base model:  155
```

Creating sequestial model

```
# Creating Sequential model with MobileNetV2 Base model
model = tf.keras.Sequential([
  base_model,
  tf.keras.layers.Conv2D(64, 3, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.GlobalAveragePooling2D(), #Adding Pooling layer to better feat
])
```

Compile the model

You must compile the model before training it. Since there are two classes, use a binary cross-entropy loss.

```
# Compiling Sequential Model
model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
# Summary of the new Model
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mobilenetv2_1.00_224 (Functi (None, 16, 16, 1280)      2257984
_________________________________________________________________
conv2d (Conv2D)              (None, 14, 14, 64)        737344
```

```
print('Number of trainable variables = {}'.format(len(model.trainable_variables)
```

```
Number of trainable variables = 139
```

```
path+"/"+all_images[0]
```

```
'drive/My Drive/PocketApps/Avantari/dataset/1082.jpg'
```

```
# Image processing and getting the features
image = Image.open(all_images[0])
# Expanding array shape so that we can get the array
image = np.expand_dims(image, axis=0)
# Making the data computation easy
image = image/127.5
image = image - 1.0
# Extracting features with our model
feature = model.predict(image)
```

```
print(feature[0])
```

```
[0.49994224 0.44996613 0.27758083 0.29522327 0.38929397 0.3438073
 0.195894   0.5861151  0.4681566  0.3366901  0.403672   0.24890278
 0.37545297 0.51397645 0.34870008 0.12072756 0.45931536 0.31429037
 0.24065983 0.7767942  0.1341397  0.09324072 0.28004178 0.5572336
 0.6312104  0.12128512 0.38273776 0.29121804 0.6402675  0.22053409
 0.36824292 0.17402567 1.0168102  0.513898   0.12440684 0.4089893
 0.32928032 0.51452655 0.22989284 0.29747814 0.7697754  0.42973357
 0.16185778 0.08714455 0.3863834  0.30866814 0.38226187 0.3141222
 0.09446295 0.11780533 0.1220076  0.16386393 0.36600053 0.5211232
 0.7306636  0.3152943  0.47405928 0.55595654 0.3969729  0.07280312
 0.16172086 0.57698613 0.09400336 0.1174629 ]
```

```
import pickle
from tqdm.notebook import tqdm
```

```
# Creating function for doing the feature extraction
def cal_feature(image_data):
    image = Image.open(image_data)
    image = np.expand_dims(image, axis=0)
    image = image/127.5
    image = image - 1.0
    feature = model.predict(image)
    return feature

# Created function for saving the extracted feature
def pickle_stuff(filename, stuff):
```

```
    save_stuff = open(filename, "wb")
    pickle.dump(stuff, save_stuff)
    save_stuff.close()

```

```
# Computing features for all images

precompute_features = []

for image_name in tqdm(all_images_listed):
    name = image_name
    features = cal_feature(image_name)
    precompute_features.append({"name": name, "features": features})

```

```
# Saving the Computed features for all images into pickle file
pickle_stuff("precompute_img_features.pickle", precompute_features)
```

Now as the feature are saved .. now Need to load and find out Similar Images

```
# Loading pickle file

def load_stuff(filename):
    saved_stuff = open(filename, "rb")
    stuff = pickle.load(saved_stuff)
    saved_stuff.close()
    return stuff
```

```
precompute_features = load_stuff("precompute_img_features.pickle")
```

```
# How the pickle file looks like ?

precompute_features[:1]
```

```
[{'features': array([[0.83356875, 0.11934735, 0.5703318 , 0.44154346, 0.536055
          0.3937406 , 0.20401797, 0.5865561 , 0.4903536 , 0.32962668,
          0.11365715, 0.48013547, 0.8012603 , 0.43802613, 0.34844816,
          0.3768936 , 0.528533  , 0.27429342, 0.47127384, 0.28969875,
          0.43658295, 0.40675595, 0.26151603, 0.36329055, 0.5200927 ,
          0.07504987, 0.22237611, 0.11110853, 0.27769855, 0.48667377,
          0.53858685, 0.4242892 , 0.2046688 , 0.16038375, 0.7619277 ,
          0.18771447, 0.18673845, 0.21487917, 0.19756776, 0.16124476,
          0.59538907, 0.586797  , 0.22485739, 0.23783682, 0.2680364 ,
          0.14924428, 0.5562108 , 0.24824658, 0.2541831 , 0.22938392,
          0.37973905, 1.0174744 , 0.4327505 , 0.42731556, 1.1012273 ,
          0.15701164, 0.36008227, 0.18594375, 0.06878649, 1.2418958 ,
          0.29685545, 0.44102025, 0.40507662, 0.63583595]], dtype=float32),
  'name': '1050.jpg'}]
```

```
import scipy as sp
from scipy import spatial
from scipy.spatial import distance
from heapq import nsmallest
```

```
# Finding Similar Images
def find_similar_image(path, count):
    distances = []
    image_name_list = []


    feature = cal_feature(path)

    for each_image_data in precompute_features:
        image_feature = each_image_data.get("features")
        eucl_dist = distance.euclidean(image_feature, feature)
        # eucl_dist = np.linalg.norm(image_feature, feature)
        distances.append(eucl_dist)

    # distances = distances.sort()
    min_distance_value = min(distances)
    print("The lowest distance for given Image {}".format(min_distance_value))
    min_distance_index = distances.index(min_distance_value)
    print("The lowest distance for given Image index {}".format(min_distance_ind
    print("The lowest distance for given Image name {}".format(precompute_featur

    for dis in nsmallest(3, distances):
        each_index = distances.index(dis)
        image_name = precompute_features[each_index].get("name")
        image_name_list.append(image_name)

    return image_name_list


```

```
image_list = find_similar_image(all_images[11], 3)
```

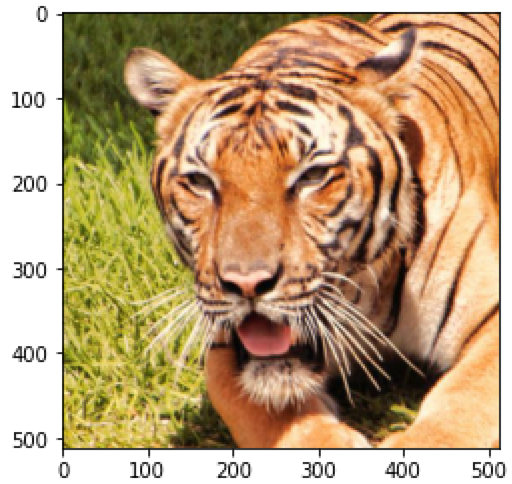
```
The lowest distance for given Image 1.9969264268875122
The lowest distance for given Image index 2377
The lowest distance for given Image name 1895.jpg
```

```
image_list
```

```
['1895.jpg', '937.jpg', '3276.jpg']
```

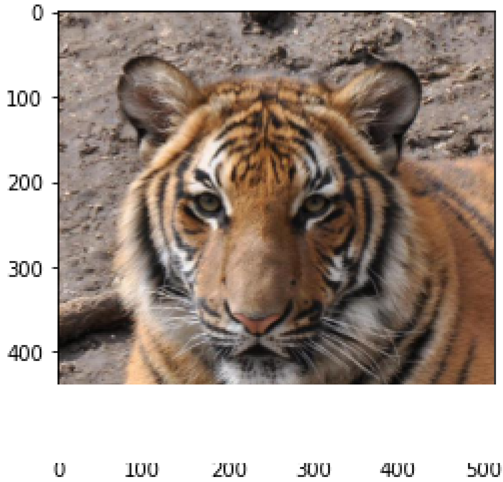
Lets take a sample Image

```
image_data = Image.open(all_images[11])
plt.imshow(image_data)
plt.show()
```



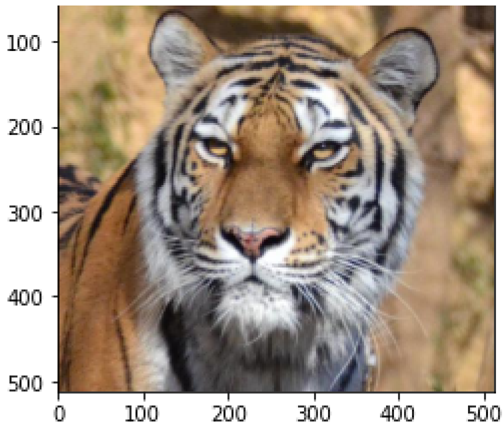
Here are the similar Images What it gets

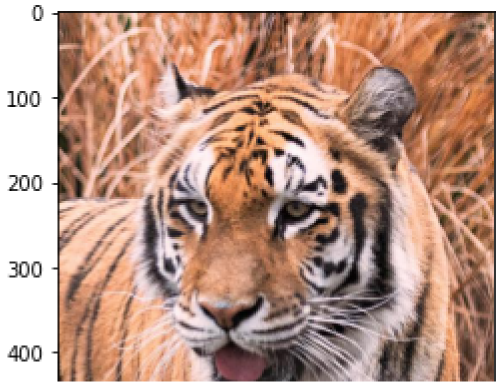
```
for img in list(image_list):
    image_data = Image.open(img)
    plt.imshow(image_data)
    plt.show()
```


0
100
200
300
400
0 100 200 300 400 500

1

1


100
200
300
400
500
0 100 200 300 400 500


0
100
200
300
400