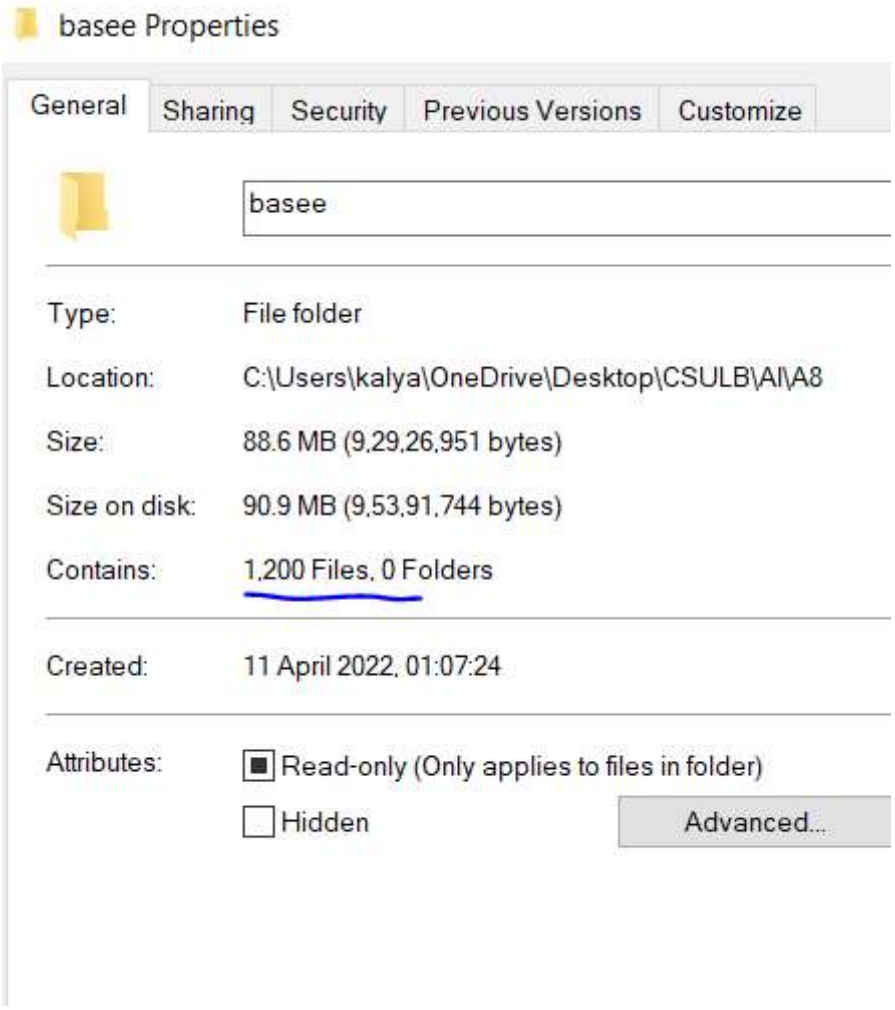
Report
Name: Pavan Sai Kalyan
ID: 029358574

Part D(PartD.ipynb) - Using selected_ids and Identity_Celeb i had extracted 1200 images to folder as show in PartD.ipynb



PartE - By using pre-trained YOLO V3 and Facenet model, implement a Python program image2vect.py.

Source code(image2vect):

```
from yoloface.utils import post_process, get_outputs_names
from facenet_pytorch import MTCNN, InceptionResnetV1
from PIL import Image

import cv2
import numpy as np
import torch

# Give the configuration and weight files for the model and load the network
# using them.
net = cv2.dnn.readNetFromDarknet("yoloface/cfg/yolov3-face.cfg", "model-weights/yolov3-wider_16000.weights")
net.setPreferableBackend(cv2.dnn.DNN_BACKEND_OPENCV)
net.setPreferableTarget(cv2.dnn.DNN_TARGET_CPU)

CONF_THRESHOLD = 0.8
NMS_THRESHOLD = 0.4
IMG_WIDTH = 416
IMG_HEIGHT = 416

def get_image_vect(input_image_path):
  """
  Returns ndarray of cropped image.
  """
  frame = cv2.imread(input_image_path)

  # Create a 4D blob from a frame.
    blob = cv2.dnn.blobFromImage(frame, 1 / 255, (IMG_WIDTH, IMG_HEIGHT),
```

```
 [0, 0, 0], 1, crop=False)

 # Sets the input to the network
 net.setInput(blob)

 # Runs the forward pass to get output of the output layers
 outs = net.forward(get_outputs_names(net))

 # Remove the bounding boxes with low confidence
     faces = post_process(frame, outs, CONF_THRESHOLD,
NMS_THRESHOLD)

 # Ensure there is only 1 face in each image.
 if (len(faces) > 1):
         print("WARNING: More than 1 face detected. File: ",
input_image_path)

 face = faces[0]

 left, top, width, height = face
 # Crop the image.
 cropped_frame = frame[top:top+height, left:left+width]
 # Flip the image to convert to RGB channel
 cropped_frame = np.flip(cropped_frame, axis=-1)
 # print(cropped_frame.shape)

                                          PIL_image                    =
Image.fromarray(np.uint8(cropped_frame)).convert('RGB')
 # print(PIL_image.size)

 # If required, create a face detection pipeline using MTCNN:
 mtcnn = MTCNN()

 # Create an inception resnet (in eval mode):
 resnet = InceptionResnetV1(pretrained='vggface2').eval()
```

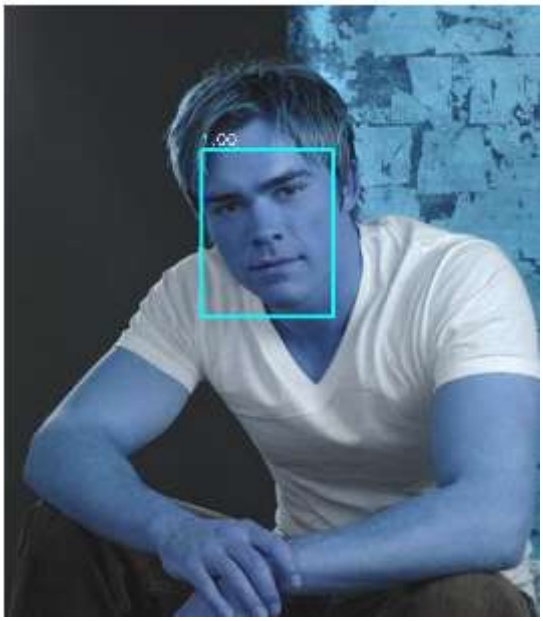
```
  # Get cropped and prewhitened image tensor
  img_cropped = mtcnn(PIL_image)

  # print(img_cropped)
  # Calculate embedding (unsqueeze to add batch dimension)
  img_embedding = resnet(img_cropped.unsqueeze(0))

                                        normalized_embedding                =
torch.nn.functional.normalize(img_embedding)
  return normalized_embedding.detach().numpy().flatten()
```

```
img_embedding.shape
```

```
torch.Size([1, 512])
```



```
[130] vect = get_image_vect("/content/drive/MyDrive/Selected_images/007472.jpg")
      print(vect.shape)

      WARNING: More than 1 face detected. File:  /content/drive/MyDrive/Selected_images/007472.jpg
      (177, 156, 3)
      (177, 156, 3)
      (156, 177)
      torch.Size([3, 177, 156])
      (512,)
```

PartF - imageFinder.py.

```
from pathlib import Path
import numpy as np
from image2vect import get_image_vect


def calculate_image_vectors(images_path):
  images = Path(images_path).glob("*.jpg")
  image_strings = [str(p) for p in images]
  print(image_strings)

  # For each image, get vector
  #print(image_strings)
  print(image_strings)
  vect_dict = {}
  for i in image_strings:
    vect_dict[i] = get_image_vect(i)
```

```
  print(len(vect_dict))
  return vect_dict


def get_matching_images(input_file, threshold, vect_dict):
  """
  """
  # Get vector for input_file
  input_img_vect=get_image_vect(input_file)

  similar_images = []
  for image_path in vect_dict:
    vec = vect_dict[image_path]
    distance=eucledian_distance(input_img_vect, vec)
    if(distance < threshold):
      similar_images.append(image_path)

  print(str(len(similar_images)) + " similar images found")

def eucledian_distance(vect1, vect2):
  return np.linalg.norm(vect1 - vect2)
```

```python
from image_finder import get_matching_images
from image2vect import get_image_vect

#IMAGES_PATH = "/content/drive/MyDrive/1200_images"
#vect_dict = calculate_image_vectors(IMAGES_PATH)
get_matching_images("/content/drive/MyDrive/1200_images/000109.jpg",1,vect_dict)
```

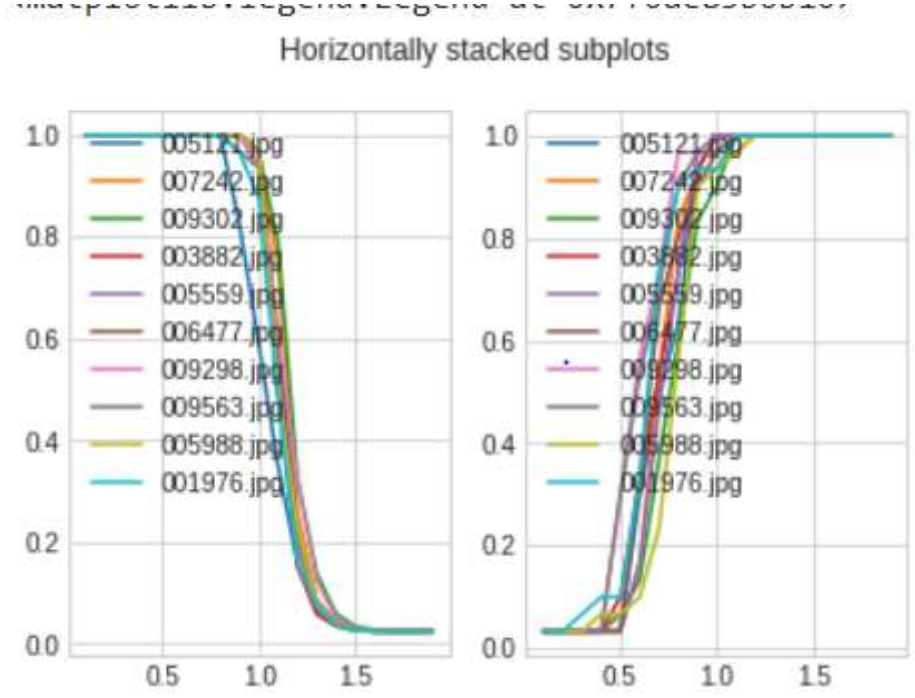
```
Processing:  /content/drive/MyDrive/1200_images/000109.jpg
Saved to output0.jpg
[[156, 182, 359, 525]]
156 182 359 525
(525, 359, 3)
torch.Size([3, 160, 160])
23 similar images found
['/content/drive/MyDrive/1200_images/007667.jpg', '/content/drive/MyDrive/1200_images/0
/content/drive/MyDrive/1200_images/000109.jpg
```

Plotting the Image:

```
fig, (p_plot, r_plot) = plt.subplots(1, 2)
fig.suptitle('Horizontally stacked subplots')
precisions_list = []
recalls_list = []

paths                                                                    =
list(Path("/content/drive/MyDrive/1200_images").glob("*.jpg"))[:10]
for i in range(len(paths)):
  path = str(paths[i])
       precisions,   recalls   =   get_precision_recall(path,   vect_dict,
inv_training_dict, training_images)
  x = list(precisions.keys())
  y = list(precisions.values())
  y2 = list(recalls.values())
  precisions_list.append(y)
  recalls_list.append(y2)
  p_plot.plot(x, y, label=os.path.basename(path))
  r_plot.plot(x, y2, label=os.path.basename(path))

p_plot.legend(loc="upper left")
r_plot.legend(loc="upper left")
```



References:

- Pre-trained YOLOFace model:

  https://drive.google.com/file/d/1xYasjU52whXMLT5MtF7RCPQkV66993oR/view

- Pre-trained FaceNet model:

  https://drive.google.com/file/d/1EXPBSXwTaqrSC0OhUdXNmKSh9qJUQ55-/view

- https://github.com/ultralytics/yolov3
- https://github.com/sotheanith/Detecting-Face-with-YoloV3-and-Facenet
- https://github.com/timesler/facenet-pytorch

- https://matplotlib.org/stable/gallery/subplots_axes_and_figures/subplots_demo.html