FACE RECOGNITION USING PYTHON

Siddharth Trivedi

60004180106

TE COMPS

Introduction

Facial recognition is the process of identifying or verifying the identity of a person using their face. It captures, analyses, and compares patterns based on the person's facial details.

Facial biometrics continues to be the preferred biometric benchmark. That's because it's easy to deploy and implement. There is no physical interaction required by the end-user. Moreover, face detection and face match processes for verification/identification are speedy.

In the Modern day, Facial recognition finds it self highly applicable to real life problems covering Security, Health, Marketing and Retail sectors, to name a few.

Scope

This project focuses on recognizing the faces from a real time picture stream. The real time stream can be provided as a video file or from a webcam. The model finds the faces in the stream and makes a prediction and labels the face with it after comparing it with the known faces from the dataset. In this specific project, we have built a dataset of images of a few characters from 'The Office' and have tried to recognize and label them using face recognition on a few images and a clip (video file) from the show.

Working Methodology

We implement Face Recognition by using deep metric learning algorithms in python and with OpenCV, the most popular library for computer vision.

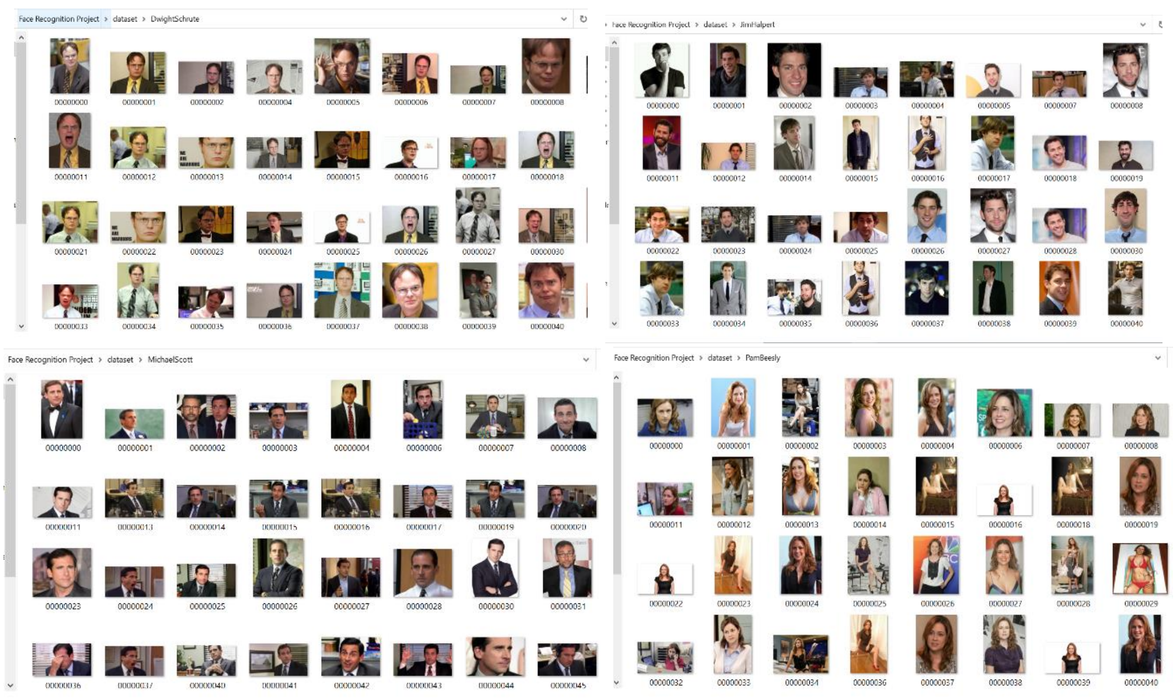
Key **Libraries/Packages** used and required:

- OpenCV
- dlib
- face_recognition (built using dlib)
- imutils

The Implementation of this project basically happens in the following steps:

1.) Building up the Dataset:

So, to apply face recognition on my favourite characters (*Michael Scott, Jim Halpert, Dwight Schrute and Pam Beesly*) from the show, '*The Office*', we first build up the dataset of their images by downloading images corresponding to the characters and labelling them accordingly.
This was implemented by using Microsoft's super convenient *Bing Image Search API*. We create a python script(*search_bing_api.py*) to download the images very quickly based upon the search query argument passed through command line and the images are fetched and saved to the output directory passed as a command line argument.


Face Recognition Project > dataset > DwightSchrute
Face Recognition Project > dataset > JimHalpert
Face Recognition Project > dataset > MichaelScott
Face Recognition Project > dataset > PamBeesly

2.) Embedding (encoding) the images in our Dataset and Face Detection:

Before applying face detection or recognition on our images we need to quantify the faces in our training dataset. We use a pre-trained deep convolutional neural network to generate 128 measurements (128-d embeddings) for each face.

We use the *HOG (Histogram of Oriented Gradients)* algorithm/model for detecting the faces that are to be embedded. In the HOG model, we started by making our image B&W since we don't need colour to find faces. The model looks at every single pixel in the image and figures out how dark the current pixel is compared to the pixels directly surrounding it. Then it replaces the pixel with an arrow pointing towards the direction in which the image is getting darker. These arrows are called gradients. To find faces in the HOG image, all we have to do is find the part of our image that looks the most similar to a known HOG pattern that was extracted from a bunch of other training faces.

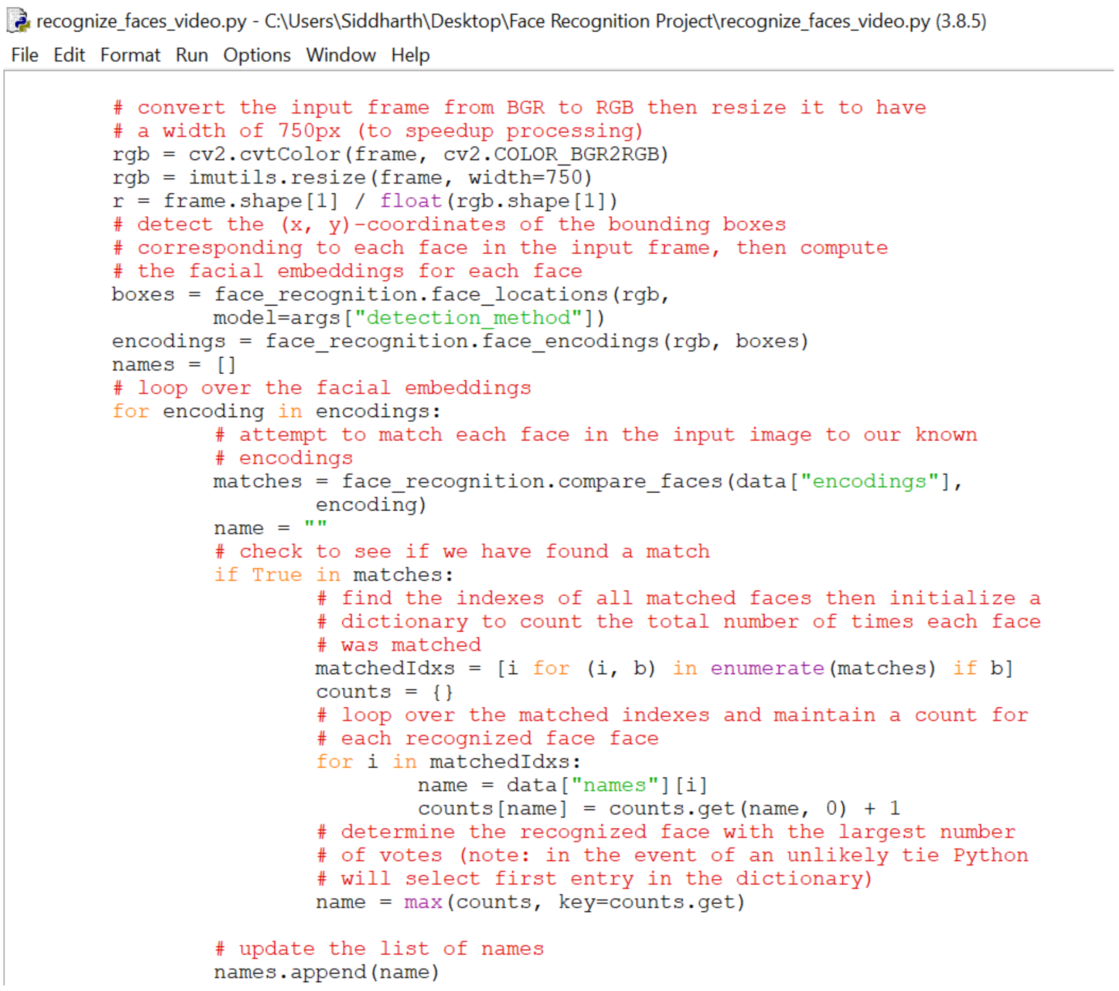
We perform the encoding for each face in our dataset and store the encodings and store them in a python pickle file so that we can use the encodings further in the project.

```
# grab the paths to the input images in our dataset
print("[INFO] quantifying faces...")
imagePaths = list(paths.list_images(args["dataset"]))
# initialize the list of known encodings and known names
knownEncodings = []
knownNames = []

# loop over the image paths
for (i, imagePath) in enumerate(imagePaths):
        # extract the person name from the image path
        print("[INFO] processing image {}/{}".format(i + 1,
                len(imagePaths)))
        name = imagePath.split(os.path.sep)[-2]
        # load the input image and convert it from BGR (OpenCV ordering)
        # to dlib ordering (RGB)
        image = cv2.imread(imagePath)
        rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        # detect the (x, y)-coordinates of the bounding boxes
        # corresponding to each face in the input image
        boxes = face_recognition.face_locations(rgb,
                model=args["detection_method"])
        # compute the facial embedding for the face
        encodings = face_recognition.face_encodings(rgb, boxes)
        # loop over the encodings
        for encoding in encodings:
                # add each encoding + name to our set of known names and
                # encodings
                knownEncodings.append(encoding)
                knownNames.append(name)

# dump the facial encodings + names to disk
print("[INFO] serializing encodings...")
data = {"encodings": knownEncodings, "names": knownNames}
f = open(args["encodings"], "wb")
f.write(pickle.dumps(data))
f.close()
```

Script from *encoding_faces.py*

3.) Face Recognition in Image and Video Files:

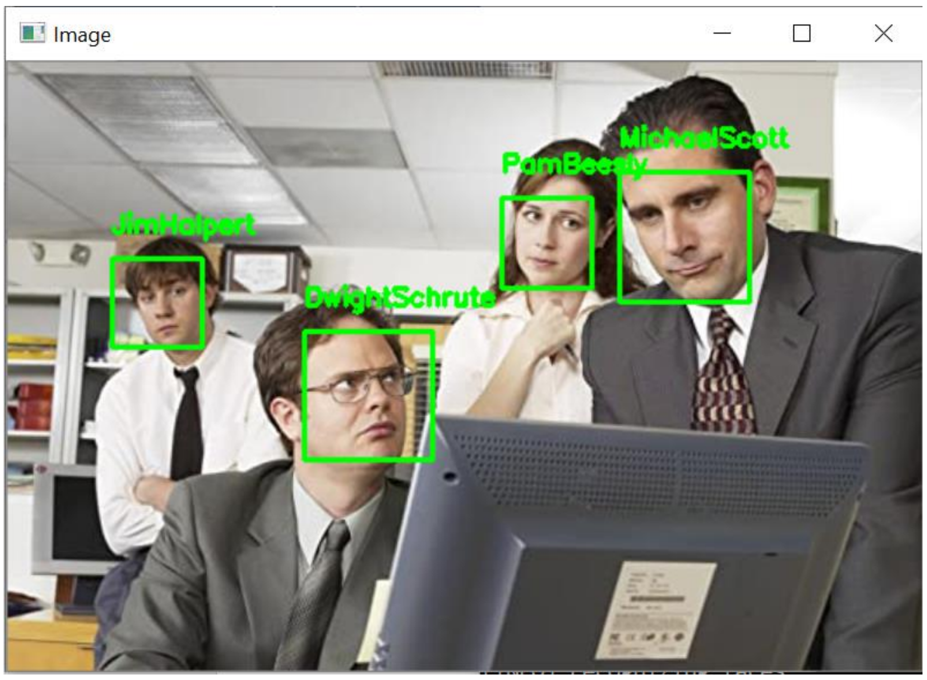
The process of face recognition in images and video files is almost the same with few variations being in the input method. A video file is converted into multiple number of frames and then each frame is then treated as a separate image and face detection and recognition is applied to that frame.
After loading the encodings from the previous step and loading the image, or grabbing the frames and looping over them for each step as a single image, HOG face detection is applied to it and 128-d encodings are produced for the same. The encoding produced is then compared to each of the encodings of our dataset images using the *compare_faces* method from *face_recognition* module which produced a Boolean value for each of the comparison. The recognized face is determined with the largest number of votes from the Boolean outputs of compare_faces method.

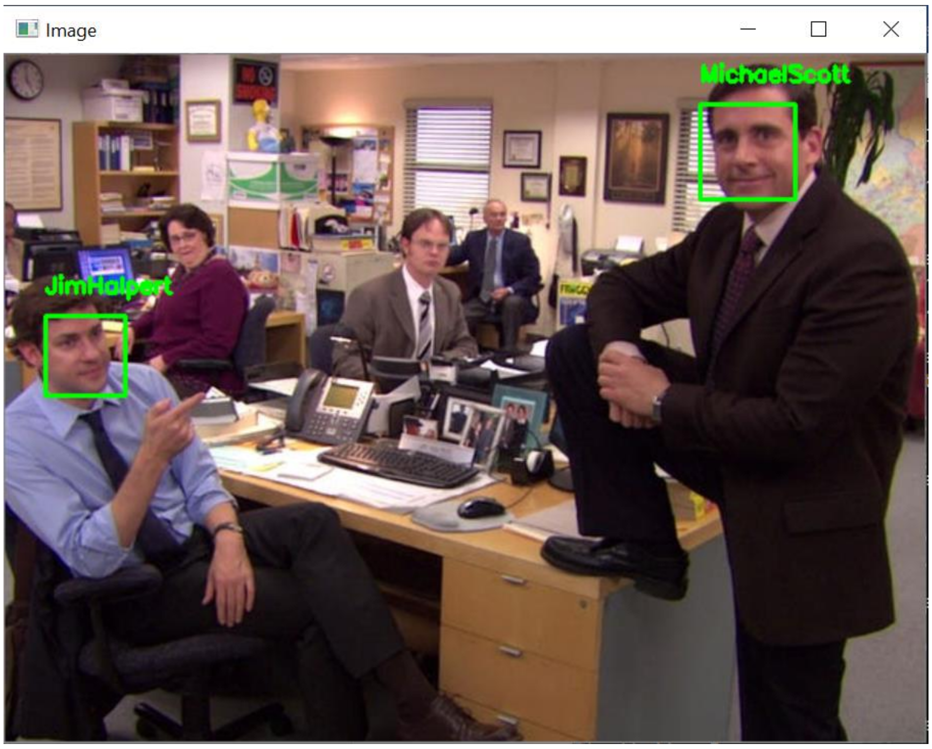

recognize_faces_video.py - C:\Users\Siddharth\Desktop\Face Recognition Project\recognize_faces_video.py (3.8.5)

```
        # convert the input frame from BGR to RGB then resize it to have
        # a width of 750px (to speedup processing)
        rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        rgb = imutils.resize(frame, width=750)
        r = frame.shape[1] / float(rgb.shape[1])
        # detect the (x, y)-coordinates of the bounding boxes
        # corresponding to each face in the input frame, then compute
        # the facial embeddings for each face
        boxes = face_recognition.face_locations(rgb,
                model=args["detection_method"])
        encodings = face_recognition.face_encodings(rgb, boxes)
        names = []
        # loop over the facial embeddings
        for encoding in encodings:
                # attempt to match each face in the input image to our known
                # encodings
                matches = face_recognition.compare_faces(data["encodings"],
                        encoding)
                name = ""
                # check to see if we have found a match
                if True in matches:
                        # find the indexes of all matched faces then initialize a
                        # dictionary to count the total number of times each face
                        # was matched
                        matchedIdxs = [i for (i, b) in enumerate(matches) if b]
                        counts = {}
                        # loop over the matched indexes and maintain a count for
                        # each recognized face face
                        for i in matchedIdxs:
                                name = data["names"][i]
                                counts[name] = counts.get(name, 0) + 1
                        # determine the recognized face with the largest number
                        # of votes (note: in the event of an unlikely tie Python
                        # will select first entry in the dictionary)
                        name = max(counts, key=counts.get)

                # update the list of names
                names.append(name)
```

Script from recognize_faces_video.py

Then the label/name associated with the recognized face is stored.
Finally, a box is drawn around the recognized face using OpenCV and the name of the person with the face is labelled around the box.





REFERENCES

- https://www.pyimagesearch.com/2018/06/18/face-recognition-with-opencv-python-and-deep-learning/
- https://www.pyimagesearch.com/2018/04/09/how-to-quickly-build-a-deep-learning-image-dataset/
- https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78
- https://face-recognition.readthedocs.io/en/latest/readme.html
- https://realpython.com/face-recognition-with-python/