Build Face Recognition Attendance

In this project we are going to learn how to perform Facial recognition with high accuracy. I will first briefly go through the theory and learn the basic implementation. Then I will create project that will use webcam to detect faces and record the attendance live in an excel sheet.

Installations

The installation process for this project is a bit more than usual. First I download a C++ compiler. I tried installing Visual Studios. I downloaded the community version for free from their website (https://visualstudio.microsoft.com/downloads/). Once the installer I run it and select the 'Desktop development with C++'.
The download and installation took some time as it is a few Gbs.

After completing and restarting the computer, I head on to Pycharm project. Here, I install the required packages. Below is the list.

1. cmake
2. dlib
3. face_recognition
4. numpy
5. opencv python

Understanding the problem

Although many face recognition algorithms have been developed over the years, their speed and accuracy balance has not been quiet optimal. But some recent advancements have shown promise. A good example is Facebook, where they are able to tag you and your friends with just a few images of training and with accuracy as high as 98%. So how does this work. Today I will try to replicate similar results using a face recognition library developed by Adam Geitgey. Let's look at the 4 problems he explained in his article (https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learningc3cffc121d78).

Face recognition is a series of several problems:

1. First, look at a picture and find all the faces in it
2. Second, focus on each face and be able to understand that even if a face is turned in a weird direction or in bad lighting, it is still the same person.
3. Third, be able to pick out unique features of the face that you can use to tell it apart from other people— like how big the eyes are, how long the face is, etc.
4. Finally, compare the unique features of that face to all the people you already know to determine the person's name.

Face Attendance Project

Now using the methods, I developed an attendance system where the user is automatically logged when they are detected in the camera. I stored the name along with the time when they appeared.

Importing

First, I import the relevant libraries

```
import face_recognition
import cv2
import numpy as np
```

1. Loading Images and Converting to RGB.

```
imgElon = face_recognition.load_image_file('ImagesBasic/Elon Musk.jpg')
imgElon = cv2.cvtColor(imgElon,cv2.COLOR_BGR2RGB)
imgTest = face_recognition.load_image_file('ImagesBasic/Elon Test.jpg')
imgTest = cv2.cvtColor(imgTest,cv2.COLOR_BGR2RGB)
```

2. Find Faces Locations and Encodings

```
faceLoc = face_recognition.face_locations(imgElon)[0]
encodeElon = face_recognition.face_encodings(imgElon)[0]
cv2.rectangle(imgElon,(faceLoc[3],faceLoc[0]),(faceLoc[1],faceLoc[2]),(255,0,255),2) # top, right, bottom, left
```

```
faceLocTest = face_recognition.face_locations(imgTest)[0]
encodeTest = face_recognition.face_encodings(imgTest)[0]
cv2.rectangle(imgTest,(faceLocTest[3],faceLocTest[0]),(faceLocTest[1],faceLocTest[2]),(255,0,255),2)
```

3. Compare Faces and Find Distance

```
results = face_recognition.compare_faces([encodeElon], encodeTest)
faceDis = face_recognition.face_distance([encodeElon], encodeTest)
cv2.putText(imgTest,f'{results} {round(faceDis[0],2)} ',(50,50),cv2.FONT_HERSHEY_COMPLEX,1,(255,0,255),3)
```

If we run this with the test image we get the value True, indicating that the face found is of Elon Mask. The is distance between the faces is 0.44. The lower the distance the better the match.

Importing Images

When I have multiple images, importing them individually can become messy. Therefore, I write a script to import all images in eachfolder at once. For this I need the os library so we will import that first. I will store all the images in one list and their names in another.

```
import face_recognition
import cv2
import numpy as np
import os
```

```
path = 'ImagesAttendance'
images = [ ]
className = [ ]
myList = os.listdir(path)
print("Total Classes Detected:",len(myList))
for cl in enumerate(myList):
    curImg = cv2.imread(f'{path}/{cl}')
    photos.append(curImg)
    className.append(os.path.splitext(cl)
```

Compute Encodings

Now that we have a list of images, we can iterate through those and create a corresponding encoded list for known faces. To do this I create a function. As earlier I will first convert it into RGB and then find its encoding using the face_encodings() function. Then I append each encoding to our list.

```
def findEncodings(images):
    encodeList = []
    for img in images:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        encode = face_recognition.face_encodings(img)
        encodeList.append(encode)
    return encodeList
```

Now we can simply call this function with the images list as the input arguments.

```
encodeListKnown = findEncodings(images)
print('Encodings Complete')
```

The While loop

The while loop is created to run the webcam. But before the while loop I had to create a video capture object so that I can grab frames from the webcam.

```
cap = cv2.VideoCapture(0)
```

Webcam Image

First, I read the image from the webcam and then resize it to quarter the size. This is done to increase the speed of the system. Even though the image being used is 1/4 th of the original, I will still use the original size while displaying. Next, I convert it to RGB.

```
while True:
    success, img = cap.read()
    imgS = cv2.resize(img, (0, 0), fx=0.25, fy=0.25)
    imgS = cv2.cvtColor(imgS, cv2.COLOR_BGR2RGB)
```

Webcam Encodings

Once we have the webcam frame, we will find all the faces in our image. The face_locations function is used for this purpose. Later we will find the face_encodings as well.

```
facesCurFrame = face_recognition.face_locations(imgS)
encodesCurFrame = face_recognition.face_encodings(imgS, facesCurFrame)
```

Find Matches

Now we can match the current face encodings to our known faces encoding list to find the matches. I also compute the distance. This is done to find the best match in case more than one face is detected at a time.

```
for encodeFace,faceLoc in zip(encodesCurFrame,facesCurFrame):
    matches = face_recognition.compare_faces(encodeListKnown, encodeFace)
    faceDis = face_recognition.face_distance(encodeListKnown, encodeFace)
```

Once we have the list of face distances we can find the minimum one, as this would be the best match.

```
matchIndex = np.argmin(faceDis)
```

Now based on the index value we can determine the name of the person and display it on the original Image.

```
if matches [matchIndex]:
    name = classNames[matchIndex].upper()
    y1,x2,y2,x1=faceLoc
    y1, x2, y2, x1 = y1*4,x2*4,y2*4,x1*4
    cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2)
    cv2.rectangle(img, (x1, y2 - 35), (x2, y2), (0, 255, 0), cv2.FILLED)
    cv2.putText(img, name, (x1 + 6, y2 - 6), cv2.FONT_HERSHEY_DUPLEX, 1.0, (255, 255, 255), 1)
```

Marking Attendance

Lastly, added the automated attendance code. I wrote a function that requires only one input which is the name of the user. First, we open our Attendance file which is in csv format. Then we read all the lines and iterate through each line using a for loop. Next, we can split using comma ‘,’. This will allow us to get the first element which is the name of the user. If the user in the camera already has an entry in the file, then nothing will happen. On the other hand, if the user is new then the name of the user along with the current time stamp will be stored. We can use the datetime class in the date time package to get the current time.

```
def markAttendance(name):
    with open('Attendance.csv','r+') as f:
        myDataList = f.readlines()
        nameList =[ ]
        for line in myDataList:
            entry = line.split(',')
            nameList.append()
        if name not in  line:
            now = datetime.now()
            dt_string = now.strftime("%H:%M:%S")
            f.writelines(f'n{name},{dt_string}')
```

Labeling Unknown faces as well

To find the unknown faces I replaced

```
if matches [matchIndex]:
    name = classNames[matchIndex].upper()
    #print(name)
    y1,x2,y2,x1 = faceLoc
    y1, x2, y2, x1 = y1*4,x2*4,y2*4,x1*4
    cv2.rectangle(img,(x1,y1),(x2,y2),(0,255,0),2)
```

```
    cv2.rectangle(img,(x1,y2-35),(x2,y2),(0,255,0),cv2.FILLED)
    cv2.putText(img,name,(x1+6,y2-6),cv2.FONT_HERSHEY_COMPLEX,1,(255,255,255),2)
    markAttendance(name)
```

with this

```
if faceDis[matchIndex] < 0.60:
    name = classNames[matchIndex].upper()
    markAttendance(name)
else: name = 'Unknown'
#print(name)
y1,x2,y2,x1 = faceLoc
y1, x2, y2, x1 = y1*4,x2*4,y2*4,x1*4
cv2.rectangle(img,(x1,y1),(x2,y2),(0,255,0),2)
cv2.rectangle(img,(x1,y2-35),(x2,y2),(0,255,0),cv2.FILLED)
cv2.putText(img,name,(x1+6,y2-6),cv2.FONT_HERSHEY_COMPLEX,1,(255,255,255),2)
```

All this does is to check if the distance to our min face is less than 0.6 or not. If it's not, then this means the person is unknown, so we change the name to unknown and don't mark the attendance.