Fake IDs Detection using Face Recognition and OpenCV

It is the era of technology. We can build connections in a fraction of a second due to widely spread social media.
No matter, you want to make new friends or to connect with the older ones, it's very trivial nowadays.
With this growing popularity of social media, there is a problem you need to look after.
The problem is the misuse of our photograph.
We love to click pictures and post them on social media. But these pictures are misused by different persons.
Fake profiles are a real issue on social media that we are facing. Fake profiles of celebrities are rampant, as are also profiles with fake names and details.
Burglars are creating fake ids and targeting potential victims. People also use fake ids of girls to take revenge and disgrace their images.
As a result of these fake ids, people are going through depression or committing suicide.

The solution

Social media provides us a facility to report an account. We need to go to their profile and tap on the report.
But the main problem is to find all those fake profiles that are using our photos without our concern.

A simple approach

We can develop a feature with the help of face recognition technology to extract the ids that are using our photographs.

How does face_recognition work?
By giving a picture, the face recognition extracts facial features like the position of eyes, width of the nose, lips location, etc., and feed it to a mathematical calculation, which gives us a unique face signature.
This facial signature is then compared to a list of known faces and extract the name or id.
to know more, please visit
https://en.wikipedia.org/wiki/Facial_recognition_system
https://pypi.org/project/face-recognition/

How can we use face recognition to extract the fake ids?
Step1: companies need to encode the pre-uploaded pictures with face recognition technology.
Step 2: They can save the encodings and corresponding labels in a file.
Step 3: When a user uploads a picture, the model should encode that picture and update the file.
Step 4: The user needs to upload a picture that he/she wants to search.

Step 5: The model encodes the picture and compares it to the already existing encodings.
Step 6: If the facial signature matches, it will extract the corresponding labels, which will be the ids.

Handling boundary cases.

1. For better performance, we should not encode or search group pictures.
2. We need to delete duplicates. Otherwise, it will give one name multiple times.
3. For false-positive cases, it will not matter much because we need to report the accounts manually.
4. We can set the tolerance level for better matching.

A dummy project on this.

Installation and Requirments

1. Python: Go to https://www.python.org/downloads/ to download python in your system.
2. dlib: https://pypi.org/search/?q=dlib
3. face_recognition: https://pypi.org/project/face-recognition/

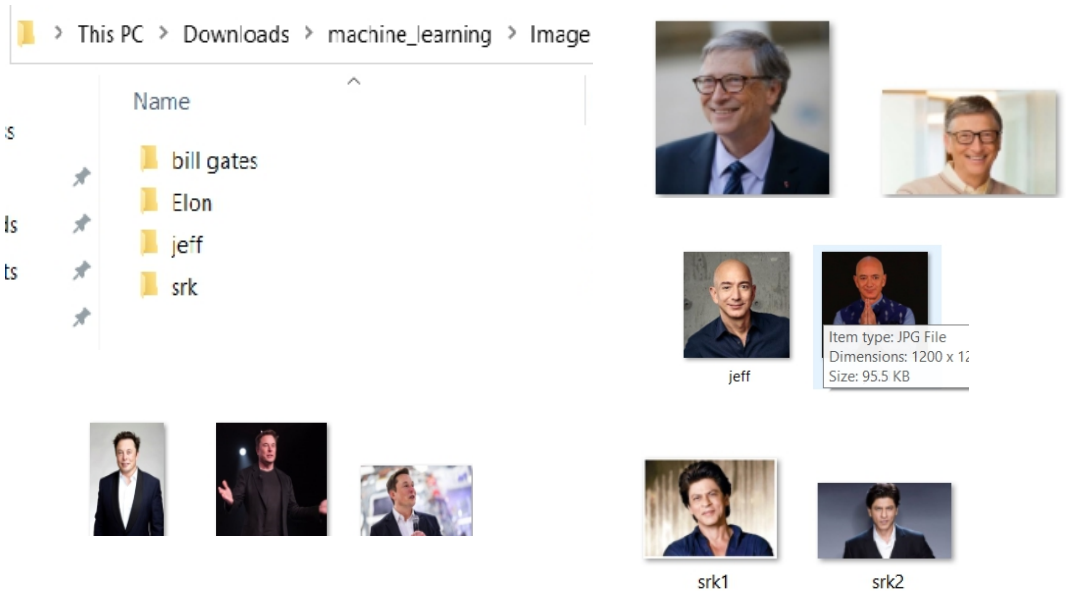
For windows users, the installation process is a bit different.

1. You need to install Visual Studio with Cmake.
2. Install dlib
3. install faced_recognition

*you need to install it in a different environment.

I use pycharm IDE, but you can use others.

We have an image folder inside that we have folders of different persons. In this case, we have folders of Elon Musk, Bill Gates, Jeff Bezos, and SRK.



Part 1

Step1
Import all the necessary packages
Open cv
NumPy
Os
Face_recognition
pickle

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

Step 2: Extracting the labels

```
# fatching images and the labels from folder
data_path='images'
categories=os.listdir(data_path)
labels=[i for i in range(len(categories))]

label_dict=dict(zip(categories,labels))
```

Step 3: Here, we create a pickle file to store our dictionary, which contains our labels.

```
#pickle file to store the dictonary
dict_file = open("frDict.pkl", "wb")
pickle.dump(label_dict, dict_file)
dict_file.close()
```

Step 4: We fetch images from folders and encode them one by one. Then we append all the encoded values in a list.

```
# this loop is used for encoding of all images present inside the folders
for category in categories:
    folder_path = os.path.join(data_path, category)
    img_names = os.listdir(folder_path)

    for img_name in img_names:
        img_path = os.path.join(folder_path, img_name)
        img = cv2.imread(img_path)

        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        try:
            encode = face_recognition.face_encodings(img)[0]
            encodeList.append(encode)
            print('Encoding Complete')
            target.append(label_dict[category])
            #print(type(encodeList))
```

Step 5: In this final step, we save the encoded list and labels in a NumPy array file.

```
#save the encode list and label in a numpy array file.
np.save('encodeList',encodeList)
np.save('target',target)
```

Part 2

Step 1: The first step is to load the NumPy files and pickle file that we save in the previous part.

```
#load the numpy files
Data=np.load('encodeList.npy')
Target=np.load('target.npy')

#lode the dictonary
dict_file= open("frDict.pkl", "rb")
fr_dict = pickle.load(dict_file)
print(fr_dict)
```

Step 2: Upload the image that you want to search. This part takes the input from you, resize the image and convert it to RGB.

```
#read new image
img= cv2.imread('bill3.png')
#img = cv2.resize(img, (400, 400))
imgs = cv2.resize(img, (0, 0), None, 0.25, 0.25)
imgs = cv2.cvtColor(imgs, cv2.COLOR_BGR2RGB)
```

Step 3: Now, the model will find the face location of the new image and encode it.

```
#encode new image
locaNewface= face_recognition.face_locations(imgs)
encodeNewface = face_recognition.face_encodings(imgs, locaNewface)
```

Step 4: The next lines of code will compare the new image with the older ones. We can tune the tolerance. The result will be store in matches in boolean format.

```
#compare new image with pre encoded file
for encodeFace, faceLoc in zip(encodeNewface, locaNewface):
    matches = face_recognition.compare_faces(Data, encodeFace,tolerance=0.5)
    faceDis = face_recognition.face_distance(Data, encodeFace)
```

Step 5: This block of code checks the matches list and append those indexes to the name.
Here indexes are values, and labels are the keys. So we need to extract keys by giving the values.
If the matches is blank, it will generate an exception. So we use try expect block to handle it.

```
# list of matched files
count=0
try:
    for i in range(0,len(matches)):
        if matches[i] == True:
            name = Target[i]
            #extracting the key by giving values from a dictonary
            output = [number for number, id in fr_dict.items() if id == name]
            final_list.append(output)
            #print(output)
            count=count+1
except Exception as e:
    print("Please enter single pic")
    exit()
```

Step 6: There is a high chance of duplicity, so we have to delete the duplicate elements and store the rest in our final list.

```
# create a final list by deleting the duplicate elements
final_output=[]
[final_output.append(x) for x in final_list if x not in final_output]
```

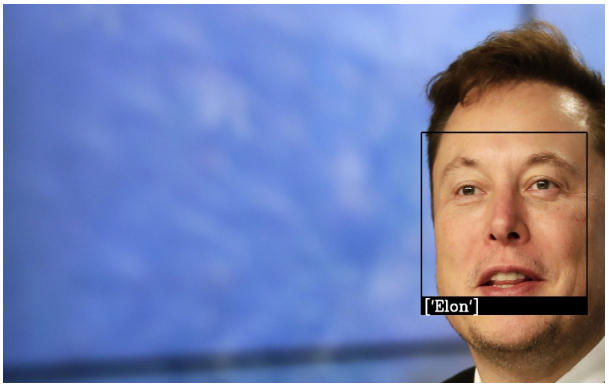
Step 7: if the counter is 0, there is no face similar to the image we provide. In this case, the program will return a message, " No face found"

```
if count==0:
    y1, x2, y2, x1 = faceLoc
    # print(x1, x2, y1, y2)
    y1, x2, y2, x1 = y1 * 4, x2 * 4, y2 * 4, x1 * 4
    cv2.rectangle(img, (x1, y1), (x2, y2), (0, 0, 0), 2)
    cv2.rectangle(img, (x1, y2 - 35), (x2, y2), (0, 0, 0), cv2.FILLED)
    cv2.putText(img, "No Result Found", (x1 + 6, y2 - 6), cv2.FONT_HERSHEY_COMPLEX, 1, (255, 255, 255), 1)
    cv2.imshow('img', img)
    cv2.waitKey(0)
```


Step 7: Else it will extract all the names that are using our photograph.

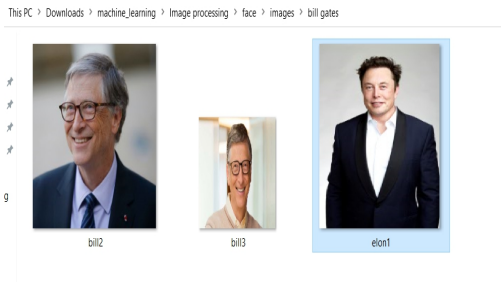
```
for final in final_output:
    y1, x2, y2, x1 = faceLoc
        #print(x1, x2, y1, y2)
    y1, x2, y2, x1 = y1 * 4, x2 * 4, y2 * 4, x1 * 4
    cv2.rectangle(img, (x1, y1), (x2, y2), (0, 0, 0), 2)
    cv2.rectangle(img, (x1, y2 - 35), (x2, y2), (0, 0, 0), cv2.FILLED)
    cv2.putText(img,str(final), (x1 + 6, y2 - 6), cv2.FONT_HERSHEY_COMPLEX, 1, (255, 255, 255), 2)
    cv2.imshow('img', img)
    cv2.waitKey(0)
```

Input

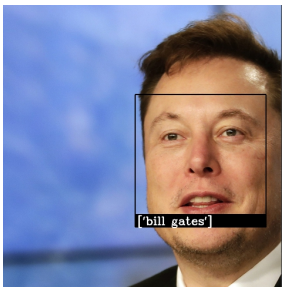


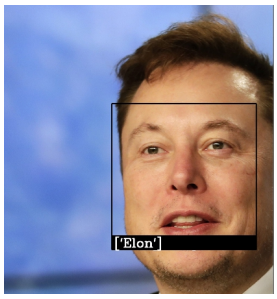
Output

What if one of our folder(Bill Gates) have Elon musk’s picture.



Input





Output