Facial detection and recognition project

Theme:

Python Keras/Opencv/C++ Facial recognition

Name:

Pop Razvan Valentin

Supervising professors : Giurgiu Mircea & Drobut Alexandra

Table Of contents

Introduction

It is required to recognize faces in a video or an image , based on an already know face which is given.The entire projects is based around the face_recognition library of python and opencv (for video streams like camera and file)

At first I started with recognizing faces in only one image ,and then drawing a rectangle around the coordinates outputed by the .face_locations() function that uses the hog algorithm , wich is less accurate on general use but it has shown great results in this case, I also used hog option of face_location because I wanted to base my computations on the CPU , making it more available to environments that do not have a dedicated GPU.

Theoretical background

Technologies used

Figure 1 OpenCv

Figure 2 Python

As main technologies used , are , python 3.6 and OpenCv.I used python for it’s large collection of libraries , as OpenCv which is used in this project for its video editing capabilities , and multiple stream support.

Libraries

The main library used,which made this project possible, is face_recognition from python,released in 2017, incorporates a small set of instructions for facial detection , in this project I used 4 out of the 7 functions implemented in the library.

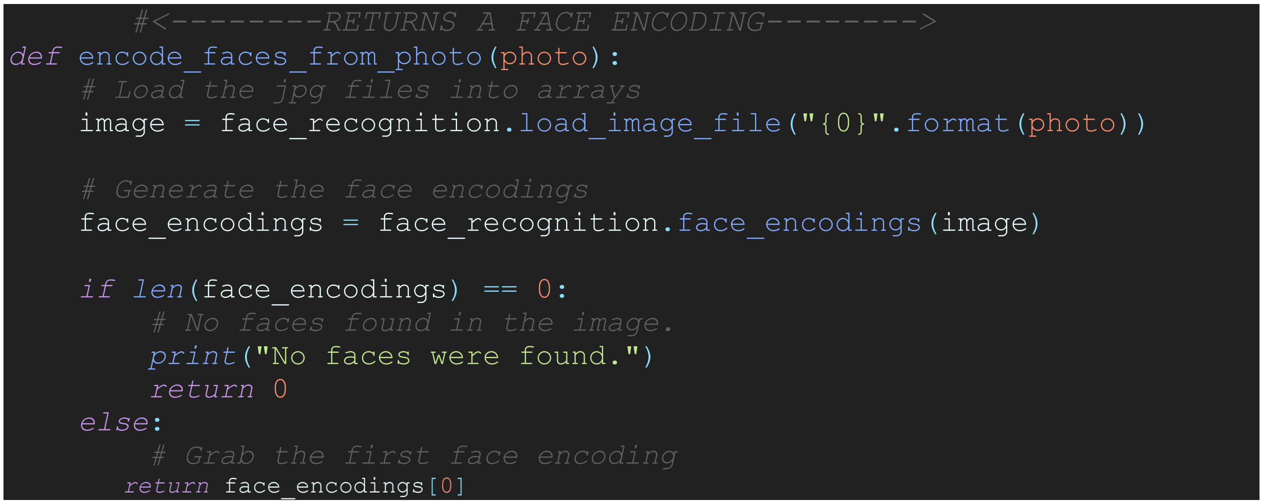
The glob module finds all the pathnames matching a specified pattern according to the rules used by the Unix shell, although results are returned in arbitrary order. No tilde expansion is done, but *, ?, and character ranges expressed with [] will be correctly matched.[1]

Implementation

Image importing and face encoding

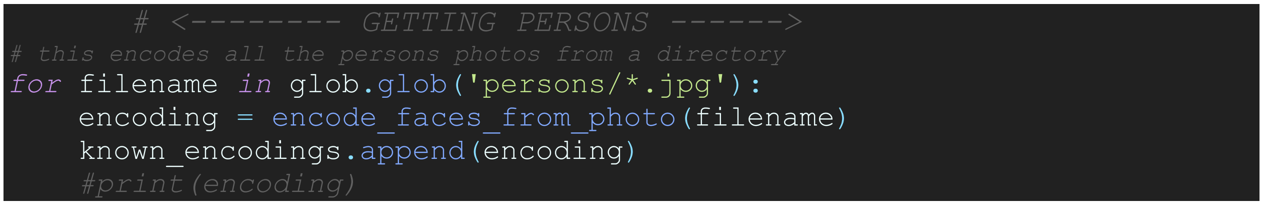
First of all , I had to know which faces I have , so that I can compare them to the faces that are gonna show on the photo/video.(the first tests and implementations were done on single photos , and after that I upgraded my code to support video input).

To obtain the facial encodings of the faces I used **.face_encodings()** and **load_image_file()** to import the image in a known format for the face_recognition() library, which resulted an array for each face.



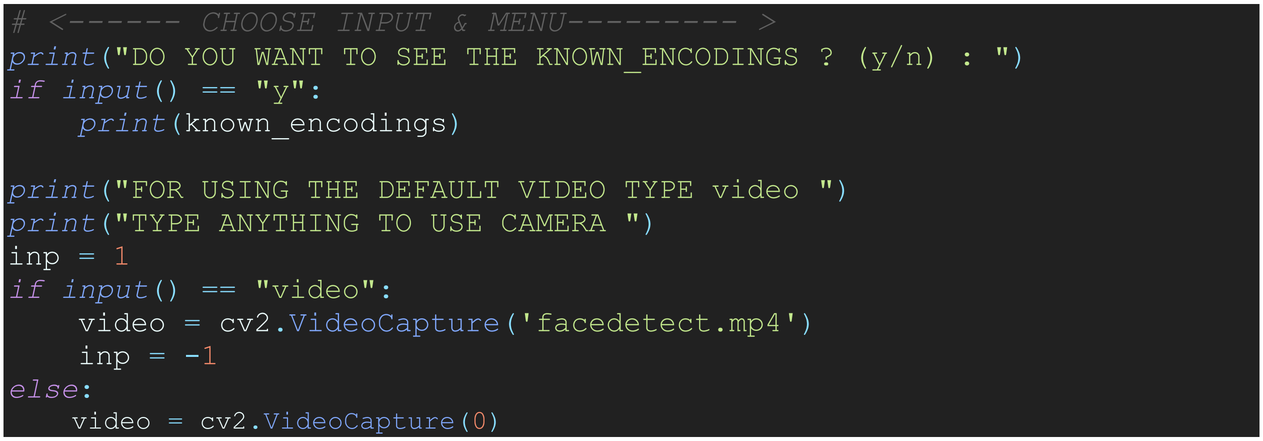
```
        #<--------RETURNS A FACE ENCODING-------->
def encode_faces_from_photo(photo):
    # Load the jpg files into arrays
    image = face_recognition.load_image_file("{0}".format(photo))

    # Generate the face encodings
    face_encodings = face_recognition.face_encodings(image)

    if len(face_encodings) == 0:
        # No faces found in the image.
        print("No faces were found.")
        return 0
    else:
        # Grab the first face encoding
        return face_encodings[0]
```

To get all the images given in a file I used the glob library and iterate through all the filenames found in the directory which had the extension .**jpg**.



```
        # <-------- GETTING PERSONS ------>
# this encodes all the persons photos from a directory
for filename in glob.glob('persons/*.jpg'):
    encoding = encode_faces_from_photo(filename)
    known_encodings.append(encoding)
    #print(encoding)
```

After that I implemented a mini-menu for the user to be able to choose the input , a given video file or the stream from the camera of the laptop/pc .



```
# <------ CHOOSE INPUT & MENU--------- >
print("DO YOU WANT TO SEE THE KNOWN_ENCODINGS ? (y/n) : ")
if input() == "y":
    print(known_encodings)

print("FOR USING THE DEFAULT VIDEO TYPE video ")
print("TYPE ANYTHING TO USE CAMERA ")
inp = 1
if input() == "video":
    video = cv2.VideoCapture('facedetect.mp4')
    inp = -1
else:
    video = cv2.VideoCapture(0)
```

If the input is from a video the frames will be upsized to a 800x600 resolution , if the input is from the camera it will be upsized to 600x400 , that being done in the final stage of the frame , afther the faces were detected and recognized, as default , in the beginning stage,it does not matter which the input is because they are resized to a 320x240 dimension for performance reasons.

Face detection and recognition

In this part of the process I made use of .**face_locations()** and **.face_encodings()** and **.compare_faces()** , first of all I had to detect all the faces in the frame and iterate through them , after that put a box over them , so that I can see that they are correctly detected.

As an improving method I use 4 variables to store the coordonites of the previous frame detected face , so that I know that is not a different face and I do not have to recognize it again , successfully speeding the program.

```
        # <--------- WE DETECT FACES -------->
frame_detect = face_recognition.face_locations(gray_frame_res,1,"hog")

for face_location in frame_detect:
    skip = True
    top, right, bottom, left = face_location
    if  previous_top != top and previous_right != right and previous_bottom != bottom and previous_left != left:
        print(" COORD : Top: {}, Left: {}, Bottom: {}, Right: {}".format(top, left, bottom, right))
        skip = False
    previous_top = top
    previous_right = right
    previous_bottom = bottom
    previous_left = left
    #<--------- BOX ON FACE--------->
    cv2.rectangle(gray_frame_res, (left, top), (right, bottom), (255, 0, 0), 1)
```

If the skip variable is set to True , this means that I have a new list of coordinates and I need to recognize the face , and if the number of faces detected is bigger than 1 skip the entire recognizing process , for speed improvement.

Then I compare the face_encodings that I got from the actual frame to the ones that I have extracted from the file and finally displaying , if the case , the name of the recognized person.

```
number_of_faces = len(frame_detect)
if not skip and number_of_faces <= 1 :
    print("I found {} face(s) in this photograph.".format(number_of_faces))
    unknown_face_encodings = face_recognition.face_encodings(gray_frame_res,frame_detect)

    for unknown_face_encoding in unknown_face_encodings:

        # Test if this unknown face encoding matches any of the three people we know
        results = face_recognition.compare_faces(known_encodings,
unknown_face_encoding, tolerance=0.6)

        name = "Unknown"

        if results[1]==1:
         name = 'Barack Obama'
        if results[0]==1:
          name = 'it s me !'
        if results[2]==1:
            name = 'Henri Coanda'
        if results[3]==1:
            name = 'Jordan Peterson'

        print("Found {} in the photo!".format(name))
```

Frame showing

And in the last part I put the name of the person recognized and show the frame.If the key ‘q’ is pressed the execution ends.

```
if number_of_faces<=1:
    cv2.putText(gray_frame_res, name, (previous_left - 25,
previous_bottom + 25), font, 0.4, (255, 0, 0), 1)

gray_frame_enlarged = cv2.resize(gray_frame_res, dim_resized)
# gray_frame_enlarged = cv2.cvtColor(gray_frame_enlarged,
cv2.COLOR_RGB2GRAY)

cv2.imshow('Facial Recognition - Live Project -  faces',
gray_frame_enlarged)
# speed and quit
if cv2.waitKey(1) & 0xFF == ord('q'):
    break
```

Conclusions

This project was a challenging one and I can say that it still needs a lot of work to be made optimal , but it works decent , being able to detect multiple faces if the lighting conditions and the quality are good .

I think that the only way to improve the frame rate , from my knowledge at this point is to move the project from running exclusively on the CPU , to GPU , but that being said would go off my goal for the project,that was to be able to run it on any thing that can run python 3.6 and a camera , something like the Raspberry Pi even , if I did not tested it on yet.

Bibliograpy

[1] https://docs.python.org/3/library/glob.html

[2] https://ro.wikipedia.org/wiki/OpenCV

[3] https://face-recognition.readthedocs.io/en/latest/face_recognition.html

[4] https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_gui/py_video_display/py_video_display.html

[5] https://face-recognition.readthedocs.io/en/latest/usage.html

[6] https://realpython.com/face-recognition-with-python/