Face Detection and Recognition using OpenCV and Python

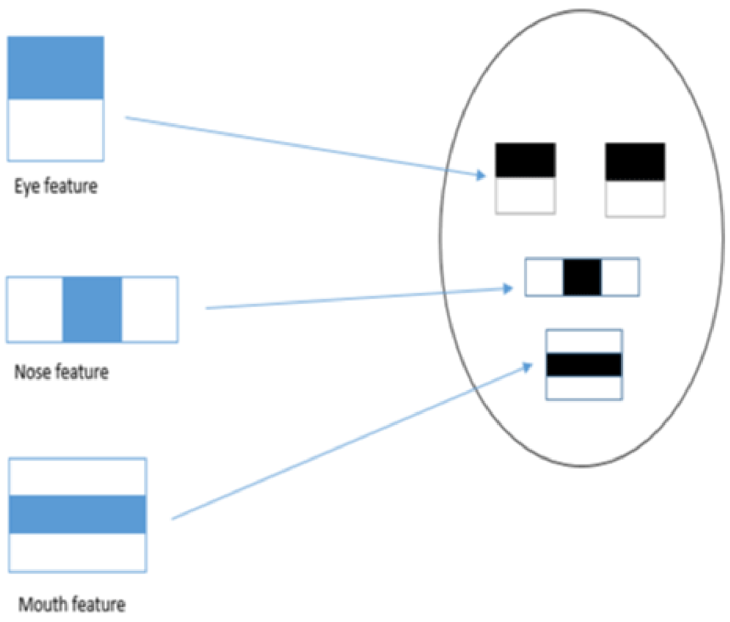
Abstract

This project gives an ideal way of detecting and recognizing human face using OpenCV, and python which is part of deep learning. This report contains the ways in which deep learning an important part of computer science field can be used to determine the face using several libraries in OpenCV along with python. This report will contain a proposed system which will help in the detecting the human face in real time. This implementation can be used at various platforms in machines and smartphones, and several software applications.

Key Words: *Python, OpenCV, Face detection.*

1. INTRODUCTION

Face recognition is the technique in which the identity of a human being can be identified using ones individual face. Such kind of systems can be used in photos, videos, or in real time machines. The objective of this article is to provide a simpler and easy method in machine technology. With the help of such a technology one can easily detect the face by the help of dataset in similar matching appearance of a person. The method in which with the help of python and OpenCV in deep learning is the most efficient way to detect the face of the person. This method is useful in many fields such as the military, for security, schools, colleges and universities, airlines, banking, online web applications, gaming. This system uses powerful python algorithm through which the detection and recognition of face is very easy and efficient.



*The above **fig.** represents the steps taking place in face recognition. There are three steps: face detection, face extraction and face recognition.*

2. OpenCV

*OpenCV is the most popular library for computer vision. Originally written in **C/C++**, it now provides bindings for **Python**. OpenCV uses machine learning algorithms to search for faces within a picture. Because faces are so complicated, there isn't one simple test that will tell you if it found a face or not. Instead, there are thousands of small patterns and features that must be matched. The algorithms break the task of identifying the face into thousands of smaller, bite-sized tasks, each of which is easy to solve. These tasks are also called classifiers. **OpenCV uses "cascades"**.*

- *What is cascade?*

  "a waterfall or series of waterfalls."

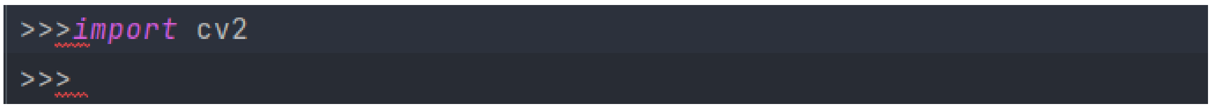
Like a series of waterfalls, the OpenCV cascade breaks the problem of detecting faces into multiple stages. For each block, it does a very rough and quick test. If that passes, it does a slightly more detailed test, and so on. The algorithm may have 30 to 50 of these stages or cascades, and it will only detect a face if all stages pass.

The advantage is that the majority of the picture will return a negative during the first few stages, which means the algorithm won't waste time testing all 6,000 features on it. Instead of taking hours, face detection can now be done in real time.

3. Installing OpenCV

First, you need to find the correct setup file for your operating system*. I found that installing OpenCV was the hardest part of the task. If you get strange unexplainable errors, it could be due to library clashes, 32/64 bit differences, and so on. I found it easiest to just use a Linux virtual machine and install OpenCV from scratch.*

Once you have completed the installation, you can test whether or not it works by firing up a Python session and typing:

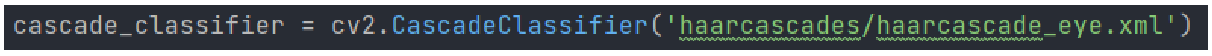


```
>>>import cv2
>>>
```

If you don't get any errors, you can move on to the next part.

4. Implementation of Haarcascades

*The **haarcascade_frontalface_default.xml** file is our pre-trained face detector, provided by the developers and maintainers of the OpenCV library.*



```
cascade_classifier = cv2.CascadeClassifier('haarcascades/haarcascade_eye.xml')
```

Haarcascade files

OpenCV comes with a lot of pre-trained classifiers. For instance, there are classifiers for smile, eyes, face, etc. These come in the form of xml files and are located in the opencv/data/haarcascades/ folder. However, to make things simple, you can also access them from https://github.com/chinni-py/chinni-py.git.

*Download the xml files and place them in the data folder in the same working directory as folder: **haarcascades***

Face detection

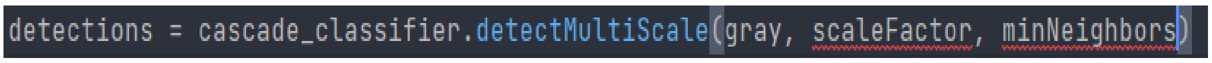
We shall be using the detect Multiscale module of the classifier. This function will return a rectangle with coordinates(x,y,w,h) around the detected face. This function has two important parameter which have to be tuned according to the data.

- ***Scale-factor***

  In a group photo, there may be some faces which are near the camera than others. Naturally, such faces would appear more prominent than the ones behind. This factor compensates for that.

- ***Min-Neighbors***

  This parameter specifies the number of neighbors a rectangle should have to be called a face.

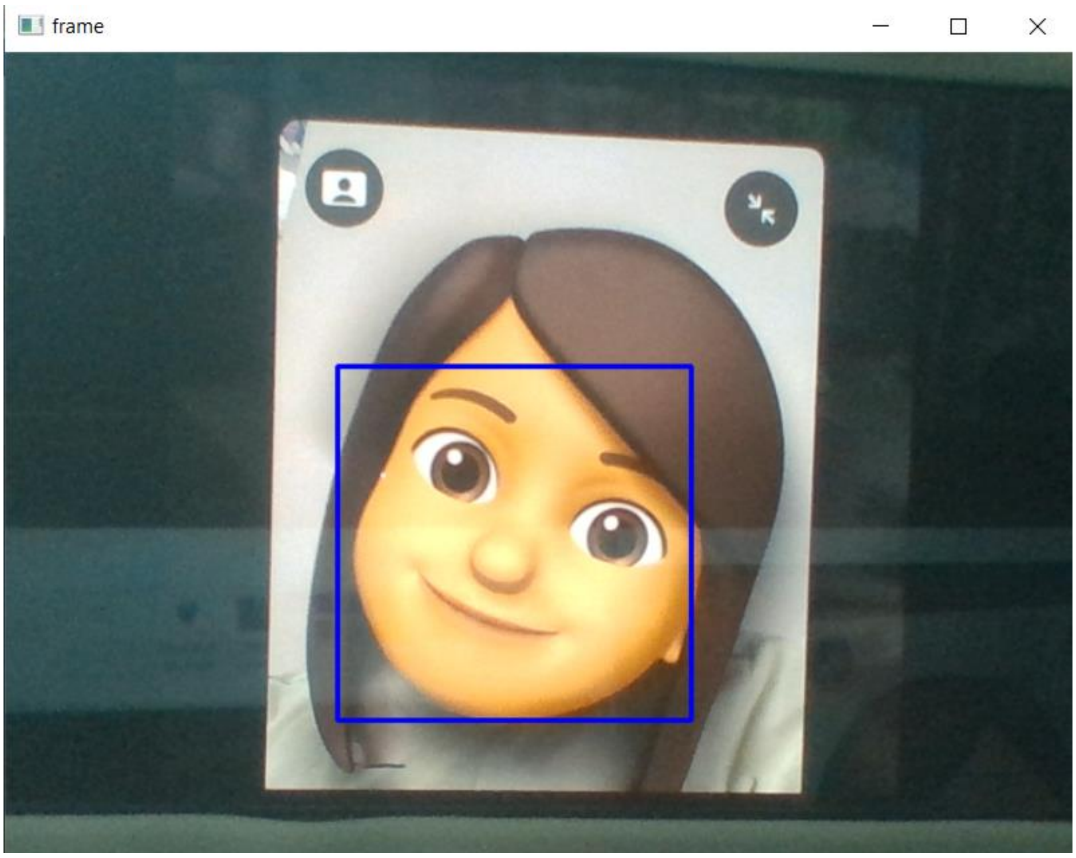


```
detections = cascade_classifier.detectMultiScale(gray, scaleFactor, minNeighbors)
```

Our next step is to loop over all the coordinates it returned and draw rectangles around them using Open CV. We will be drawing a green rectangle with a thickness of 2.

```
(x,y,w,h) = detections[0]
frame = cv2.rectangle(frame,(x,y),(x+w,y+h),(255,0,0),2)
```

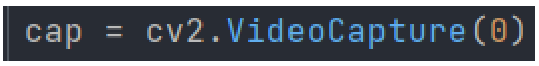
Finally, let's check weather it displays the original image in colored to see if the face has been detected correctly or not.



This is the sample output for face recognition or face detection.

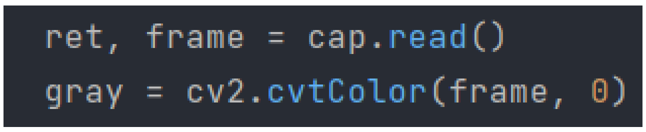
5. Face Detection with generalized function

- *It recognizes the video from system.*

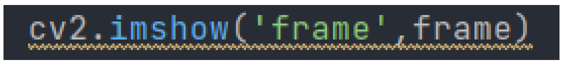


```
cap = cv2.VideoCapture(0)
```

- *It reads the video frames and color of the face.*

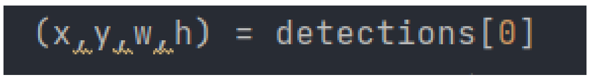


```
ret, frame = cap.read()
gray = cv2.cvtColor(frame, 0)
```



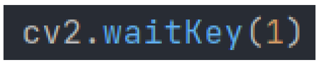
```
cv2.imshow('frame',frame)
```

- *It detects the rectangle box over the face as given below.*



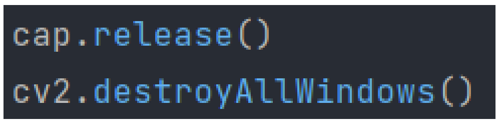
```
(x,y,w,h) = detections[0]
```

- *The **waitkey()** is a keyword binding function and it only accepts time in milliseconds as an argument. When you add any time as an argument , then it waits for the specified time and then the program continues. If 0 is passed , it waits indefinitely until a key is pressed.*



```
cv2.waitKey(1)
```

- *And for closing the window.*



```
cap.release()
cv2.destroyAllWindows()
```

6. Advantages and Disadvantages

*The **advantages** of the face recognition system include faster processing, automation of the identity, breach of privacy, massive data storage, best results, enhanced security, real time face recognition of students in schools and colleges, employees at corporate offices, smartphone unlock and many more in day to day life.*

*Few **disadvantages** in this system include the costing, or the funding, very good cameras of high definition are required, poor image quality may limit the effectiveness of this system, size of the image will matter because it becomes difficult to recognize the face in small images, Face angles can limit the face recognition reliability, massive storage is required for this system to work effectively.*

7. Conclusion

Face recognition systems are currently associated with many top technological companies and industries making the work of face recognition easier. The use of python programming and OpenCV makes it an easier and handy tool or system which can be made by anyone according to their requirement. The proposed system discussed in this project will be helpful for many as it is user friendly and cost_ efficient system. Hence by the use of python and OpenCV the face recognition system can be designed for various purposes.