haarcascade_frontalface_default. xml
is a haar cascade designed by OpenCV to detect the frontal face. This haar cascade is available on github . A Haar Cascade works by training the cascade on thousands of negative images with the positive image superimposed on it.
Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

cv2.CascadeClassifier
detect faces and eyes in an image. First, a cv::CascadeClassifier is created and the necessary XML file is loaded using the cv::CascadeClassifier::load method. Afterwards, the detection is done using the cv::CascadeClassifier::detectMultiScale method, which returns boundary rectangles for the detected faces or eyes.

cam.read()
read imaage

cv2.rectangle
cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
creates a rectange about the portion required
or

```
cvRound( (eyes[j].width + eyes[j].height)*0.25 );
circle( frame, eye_center, radius, Scalar( 255, 0, 0 ), 4 );
```

circle around face amd eyes

cv2.imwrite("TrainingImage\ "+name +"."+Id +'.'+ str(sampleNum) + ".jpg", gray[y:y+h,x:x+w])
saving captured image to training folder

StudentDetails\StudentDetails.csv
student id and name

```
with open('StudentDetails\StudentDetails.csv','a+') as csvFile:
        writer = csv.writer(csvFile)
        writer.writerow(row)
csvFile.close()
```

open csv file

//
recognizer = cv2.face.LBPHFaceRecognizer_create()

Use of OpenCV's LBPH Face Recognizer to train the dataset that outputs trainingData.yml file that we'll be using for face recognition.

```
static Ptr<LBPHFaceRecognizer>
cv::face::LBPHFaceRecognizer::create
                                    (  int     radius = 1,
                                       int     neighbors = 8,
                                       int     grid_x = 8,
                                       int     grid_y = 8,
                                       double  threshold = DBL_MAX
                                    )
```

Python:

```
retval = cv.face.LBPHFaceRecognizer_create( [, radius[, neighbors[, grid_x[, grid_y[, threshold]]]]] )
```

Parameters

radius	The radius used for building the Circular Local Binary Pattern. The greater the radius, the smoother the image but more spatial information you can get.
neighbors	The number of sample points to build a Circular Local Binary Pattern from. An appropriate value is to use `8` sample points. Keep in mind: the more sample points you include, the higher the computational cost.
grid_x	The number of cells in the horizontal direction, 8 is a common value used in publications. The more cells, the finer the grid, the higher the dimensionality of the resulting feature vector.
grid_y	The number of cells in the vertical direction, 8 is a common value used in publications. The more cells, the finer the grid, the higher the dimensionality of the resulting feature vector.
threshold	The threshold applied in the prediction. If the distance to the nearest neighbor is larger than the threshold, this method returns -1.

The Circular Local Binary Patterns (used in training and prediction) expect the data given as grayscale images, use cvtColor to convert between the color spaces.

faces,Id = getImagesAndLabels("TrainingImage")
taking image one by one and returning face and id of each image taken during train

//
[os.path.join(path,f) for f in os.listdir(path)]
Python method listdir() returns a list containing the names of the entries in the directory given by path. The list is in arbitrary order. It does not include the special entries '.' and '..' even if they are present in the directory.

Parameters

- **path** − This is the directory, which needs to be explored.

Return Value

This method returns a list containing the names of the entries in the directory given by path.

```
def getImagesAndLabels(path):
    #get the path of all the files in the folder
    imagePaths=[os.path.join(path,f) for f in os.listdir(path)]
    #print(imagePaths)

    #create empty face list
    faces=[]
    #create empty ID list
    Ids=[]
    #now looping through all the image paths and loading the Ids and the images
    for imagePath in imagePaths:
        #loading the image and converting it to gray scale
        pilImage=Image.open(imagePath).convert('L')
        #Now we are converting the PIL image into numpy array
        imageNp=np.array(pilImage,'uint8')
        #getting the Id from the image
        Id=int(os.path.split(imagePath)[-1].split(".")[1])
        # extract the face from the training image sample
        faces.append(imageNp)
        Ids.append(Id)
    return faces,Ids
```

//
createLBPHFaceRecognizer()

gray=cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)

OpenCV's pre-trained classifiers

OpenCV already contains many pre-trained classifiers for face, eyes, smile etc. Those XML files are stored in **opencv/data/haarcascades/** folder:

```
~/OpenCV/opencv/data/haarcascades$ ls


haarcascade_eye_tree_eyeglasses.xml   haarcascade_mcs_leftear.xml
haarcascade_eye.xml                   haarcascade_mcs_lefteye.xml
haarcascade_frontalface_alt2.xml      haarcascade_mcs_mouth.xml
haarcascade_frontalface_alt_tree.xml  haarcascade_mcs_nose.xml
haarcascade_frontalface_alt.xml       haarcascade_mcs_rightear.xml
haarcascade_frontalface_default.xml   haarcascade_mcs_righteye.xml
haarcascade_fullbody.xml              haarcascade_mcs_upperbody.xml
haarcascade_lefteye_2splits.xml       haarcascade_profileface.xml
haarcascade_lowerbody.xml             haarcascade_righteye_2splits.xml
haarcascade_mcs_eyepair_big.xml       haarcascade_smile.xml
haarcascade_mcs_eyepair_small.xml     haarcascade_upperbody.xml
```

OpenCV's face detection

Let's load the required XML classifiers.

```
face_cascade =
cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
```

Then, we need to load input image in grayscale mode:

```
img = cv2.imread('xfiles4.jpg')

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

We use **v2.CascadeClassifier.detectMultiScale()** to find faces or eyes, and it is defined like this:

```
cv2.CascadeClassifier.detectMultiScale(image[, scaleFactor[,
minNeighbors[, flags[, minSize[, maxSize]]]]])
```

Where the parameters are:

1.**image** : Matrix of the type CV_8U containing an image where objects are detected.

2.**scaleFactor** : Parameter specifying how much the image size is reduced at each image scale.

Picture source: Viola-Jones Face Detection
This scale factor is used to create scale pyramid as shown in the picture. Suppose, the scale factor is 1.03, it means we're using a small step for resizing, i.e. reduce size by 3 %, we increase the chance of a matching size with the model for detection is found, while it's expensive.

3.**minNeighbors** : Parameter specifying how many neighbors each candidate rectangle should have to retain it. This parameter will affect the quality of the detected faces: higher value results in less detections but with higher quality. We're using 5 in the code.

4.**flags** : Parameter with the same meaning for an old cascade as in the function cvHaarDetectObjects. It is not used for a new cascade.

5.**minSize** : Minimum possible object size. Objects smaller than that are ignored.

6.**maxSize** : Maximum possible object size. Objects larger than that are ignored.

If faces are found, it returns the positions of detected faces as Rect(x,y,w,h).

```
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
```

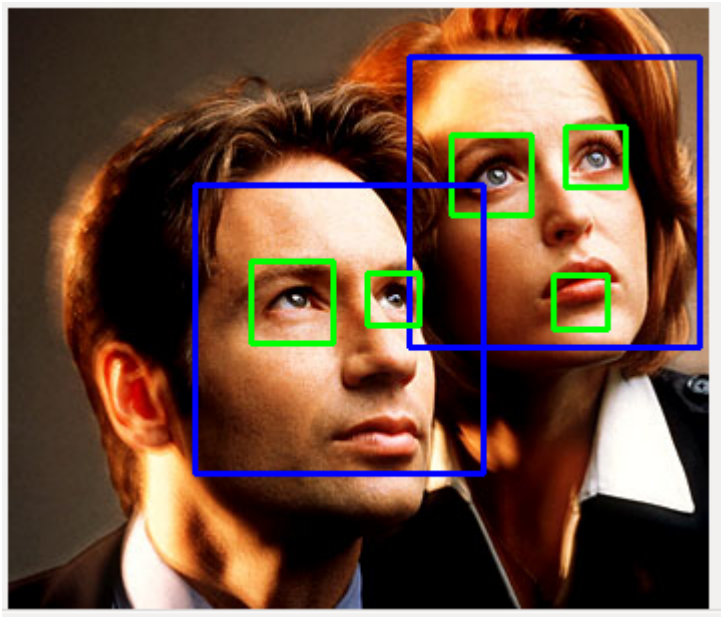
Once we get these locations, we can create a ROI for the face and apply eye detection on this ROI.

The Code

```
import numpy as np

import cv2

from matplotlib import pyplot as plt



face_cascade =
cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')



img = cv2.imread('xfiles4.jpg')

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)



faces = face_cascade.detectMultiScale(gray, 1.3, 5)




for (x,y,w,h) in faces:

    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)

    roi_gray = gray[y:y+h, x:x+w]

    roi_color = img[y:y+h, x:x+w]
```

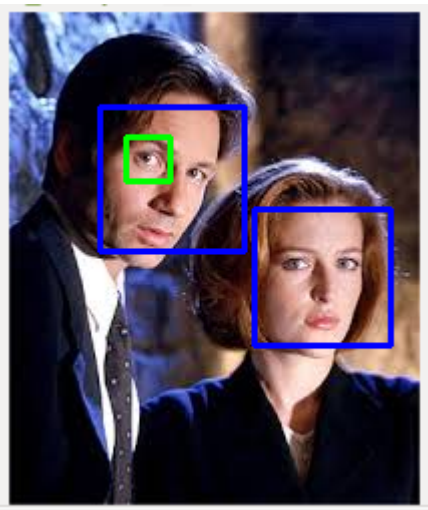
```
    eyes = eye_cascade.detectMultiScale(roi_gray)

    for (ex,ey,ew,eh) in eyes:

        cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)



cv2.imshow('img',img)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

Output 1



We're almost there except we got an additional eye.

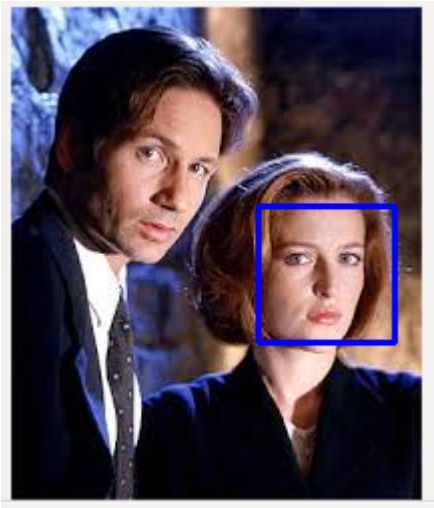
Output 2



I got the image using:

```
faces = face_cascade.detectMultiScale(gray, 1.03, 3)
```

But if with scaleFactor = 3 and minNeighbors = 5, I got this:



https://medium.com/@ankit.bhadoriya/face-recognition-using-open-cv-part-3-574a0766cd84

//kdrgfui thriutruhtu4hnnu65h985h49tn534thnerijvi
faceCascade.detectMultiScale(gray, 1.2,5)

`scaleFactor` – Parameter specifying how much the image size is reduced at each image scale.

Basically, the scale factor is used to create your scale pyramid. More explanation, your model has a fixed size defined during training, which is visible in the XML. This means that this size of the face is detected in the image if present. However, by rescaling the input image, you can resize a larger face to a smaller one, making it detectable by the algorithm.
1.05 is a good possible value for this, which means you use a small step for resizing, i.e. reduce the size by 5%, you increase the chance of a matching size with the model for detection is found. This also means that the algorithm works slower since it is more thorough. You may increase it to as much as 1.4 for faster detection, with the risk of missing some faces altogether.

- `minNeighbors` – Parameter specifying how many neighbors each candidate rectangle should have to retain it.

This parameter will affect the quality of the detected faces. Higher value results in fewer detections but with higher quality. 3~6 is a good value for it.

- `minSize` – Minimum possible object size. Objects smaller than that are ignored.

This parameter determines how small size you want to detect. You decide it! Usually, [30, 30] is a good start for face detection.

- `maxSize` – Maximum possible object size. Objects bigger than this are ignored.

This parameter determines how big size you want to detect. Again, you decide it! Usually, you don't need to set it manually, the default value assumes you want to detect without an upper limit on the size of the face.

ts = time.time()
date = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d')
timeStamp = datetime.datetime.fromtimestamp(ts).strftime('%H:%M:%S')

```
from datetime import timedelta
>>> delta = timedelta(
...     days=50,
...     seconds=27,
...     microseconds=10,
...     milliseconds=29000,
...     minutes=5,
...     hours=8,
...     weeks=2
... )
>>> # Only days, seconds, and microseconds remain
>>> delta
datetime.timedelta(days=64, seconds=29156, microseconds=10)
```

`days`	Between -999999999 and 999999999 inclusive
`seconds`	Between 0 and 86399 inclusive
`microseconds`	Between 0 and 999999 inclusive