Object Detection : Face Detection using Haar Cascade Classfiers

bogotobogo.com/python/OpenCV_Python/python_opencv3_Image_Object_Detection_Face_Detection_Haar_Cascade_Classifiers.php

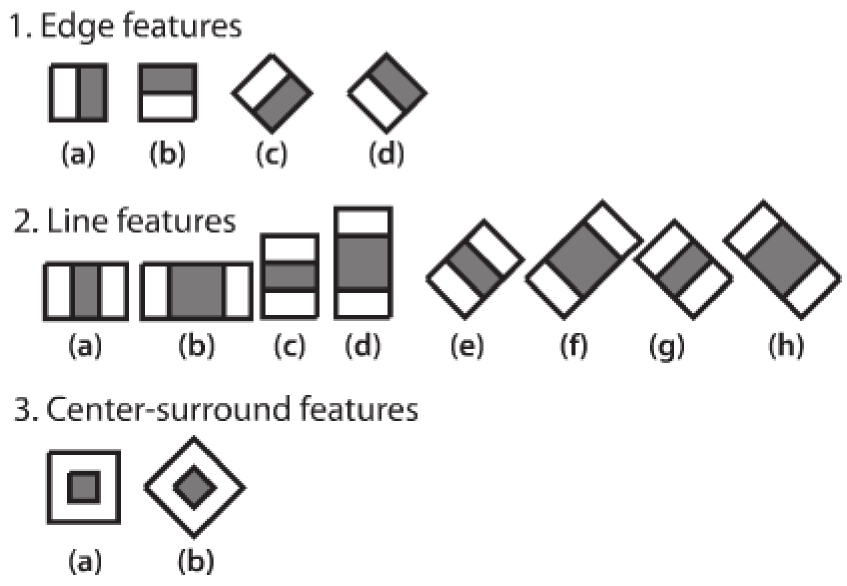
bogotobogo.com site search:

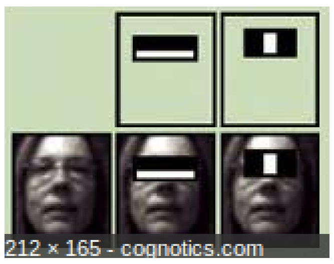
Face Detection

"Face detection is a computer technology that determines the locations and sizes of human faces in arbitrary (digital) images. It detects facial features and ignores anything else, such as buildings, trees and bodies. Face detection can be regarded as a more general case of face localization. In face localization, the task is to find the locations and sizes of a known number of faces (usually one)." - wiki - Face detection

Haar features

OpenCV's algorithm is currently using the following Haar-like features which are the input to the basic classifiers:

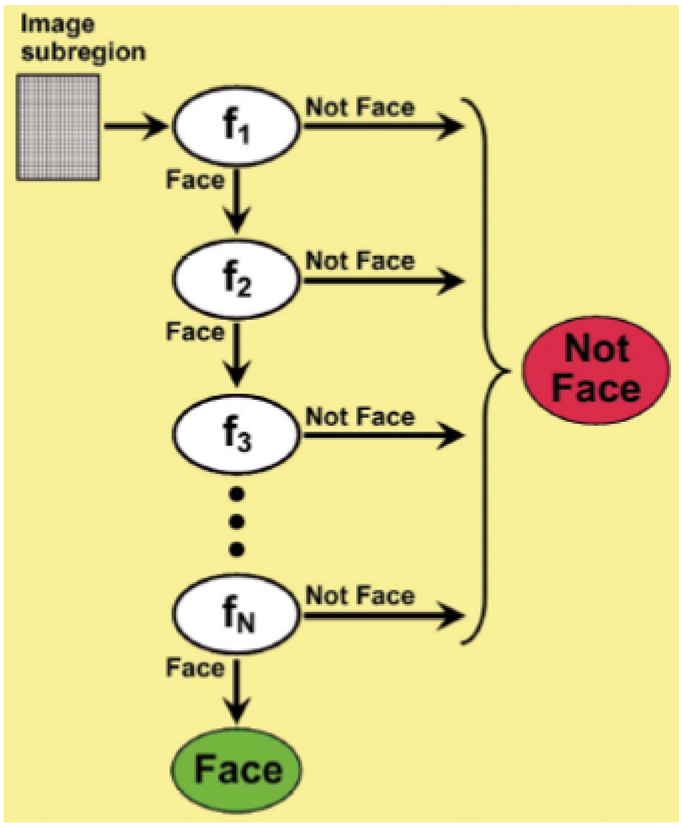




Picture source: How Face Detection Works

Cascade of Classifiers

"Instead of applying all the 6000 features on a window, group the features into different stages of classifiers and apply one-by-one. (Normally first few stages will contain very less number of features). If a window fails the first stage, discard it. We don't consider remaining features on it. If it passes, apply the second stage of features and continue the process. The window which passes all stages is a face region." - Face Detection using Haar Cascades.



Picture source: How Face Detection Works

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:

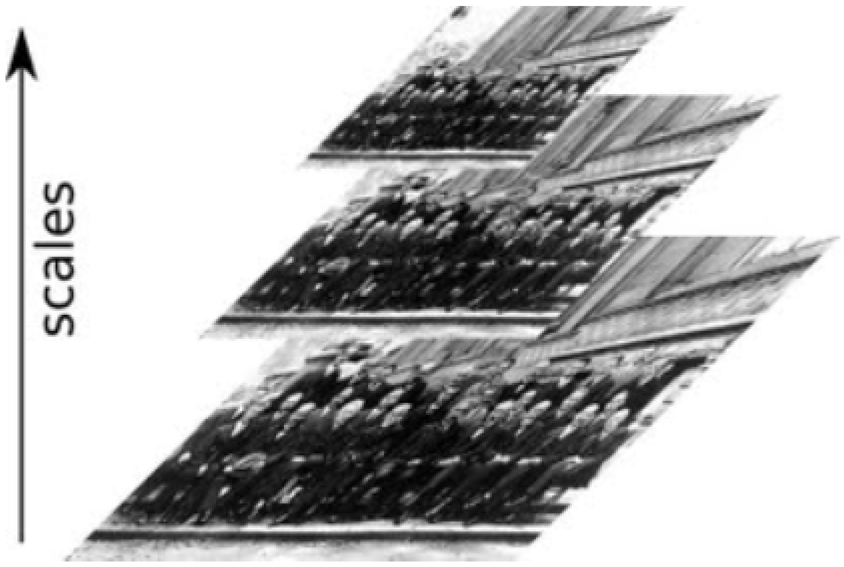
```
img = cv2.imread('xfiles4.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

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).

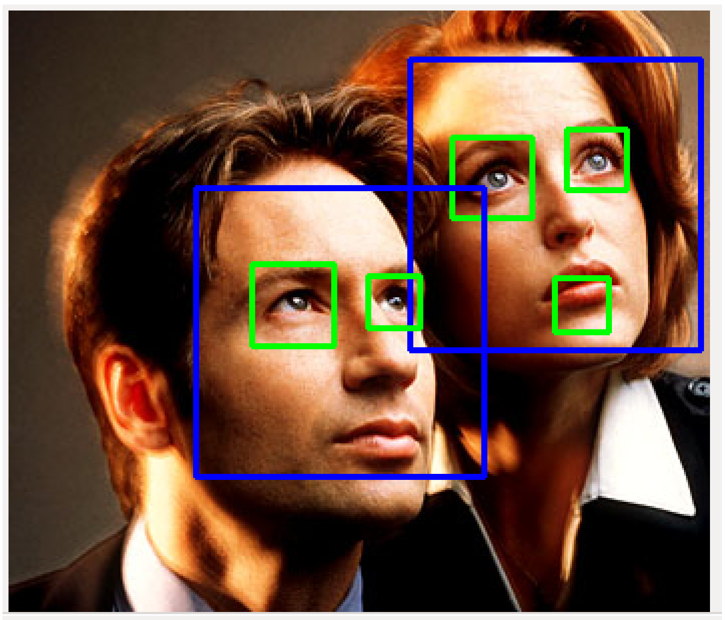
```
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
```

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



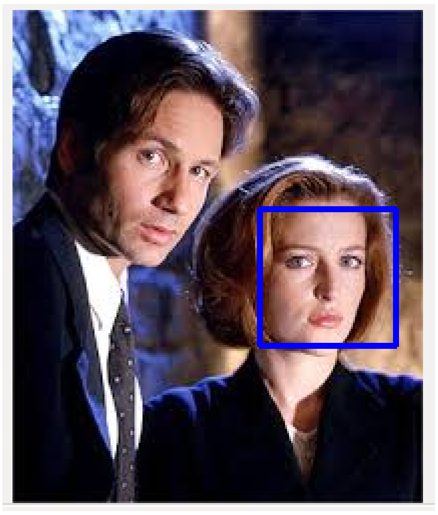
We're almost there except we got an additional eye.

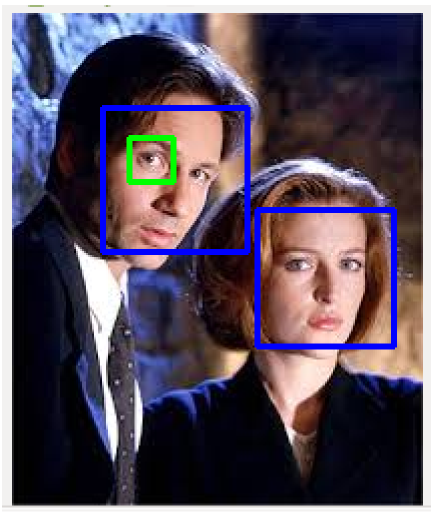
Output 2

I got the image using:

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

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





Ph.D. / Golden Gate Ave, San Francisco / Seoul National Univ / Carnegie Mellon / UC Berkeley / DevOps / Deep Learning / Visualization

Sponsor Open Source development activities and free contents for everyone.

Thank you.

- K Hong