# **Face Detection Application**

In this app we are detecting faces in images, videos, and the webcam of your computer.

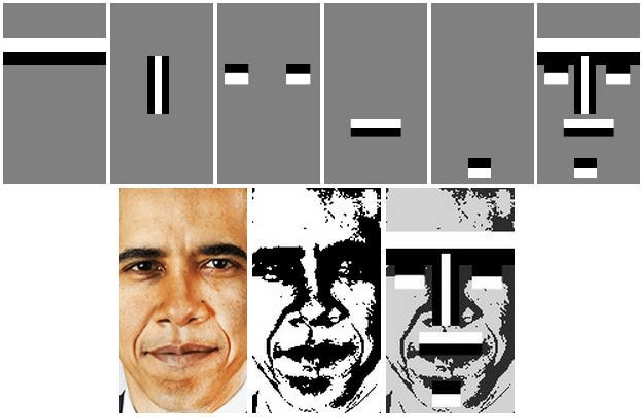
**Libraries used:**

* OpenCV
* Random

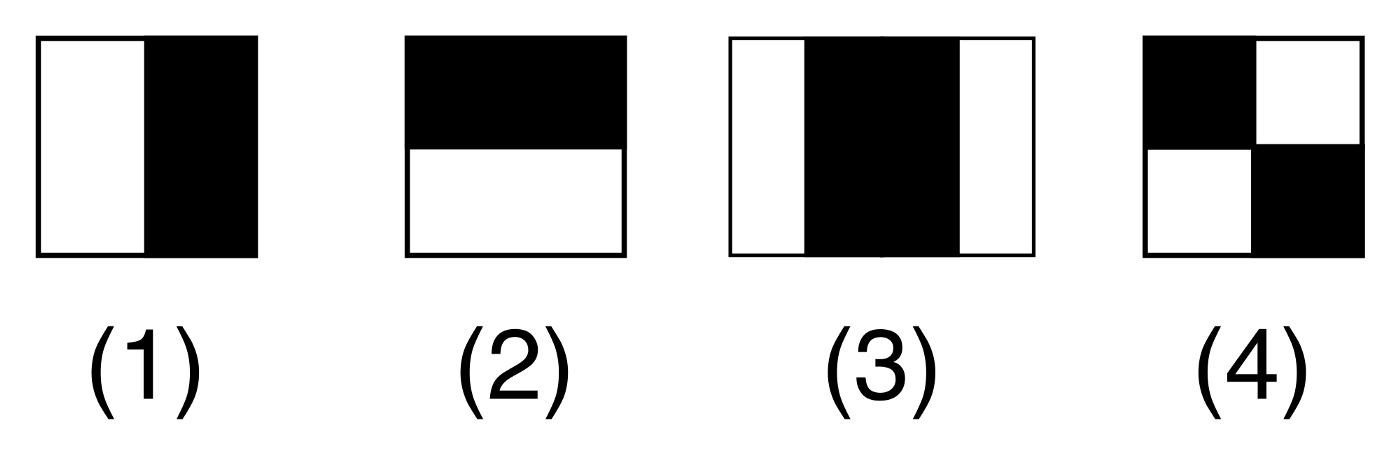
**TRAINING THE ALGORITHM TO DETECT FACES:**

How does an algorithm detect faces?

**Haar cascade-**



Basically, it finds relation between adjacent pixels. As we see in the image above, we converted the image to greyscale and then just checked the lighting. For example, the bridge of the nose is lighter than the areas on its left or right. It takes a haar feature and keeps on moving it across the image to check if it is a positive match



The haar features.

Every HF will give us a number- a number for the white pixels and a number for the black ones and then we find the difference between the two.

We can set a threshold for it. For example, if we set the threshold to be 50 then the difference between the white and the black pixel number should be above 50 to classify it as an actual facial feature.

Once we have found our first 1000 winner haar features, we chain them together into our face detector. Then a cascade of HFs is made and that is what is stored in the .xml file that we used in our program.

**OPENCV MAKES THE JOB EASIER-**

OpenCV already has a pre-trained classifier that contains all the haar features that are best to make a frontal face.

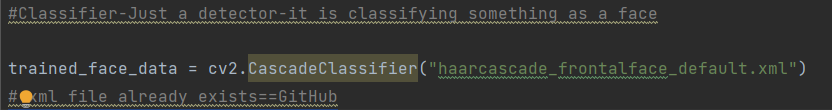
After it is classified, we pass a sliding window of the image into the classifier. It is run through all the haar cascades. If it reaches all the way to the end, it is a face.

For visualization:

<https://www.youtube.com/watch?v=hPCTwxF0qf4>

**PROGRAM EXPLANATION**

**Detecting faces in images-**

****

**Classifier-** Detects a face. “Classifies” something as a face.

**Cascade-** Goes block by block in an image and each block is pushed through the haarcascade algorithm to check if it is a facial feature.



Here *face\_coordinates* is using the algorithm to detect the coordinates of what it will classify as face.

The coordinates are basically tuples inside a tuple which contain (x,y,w,h)

* x- The x coordinate of the top left corner of the square to be drawn around the face.
* y- The y coordinate of the top left corner of the square to be drawn around the face.
* w- The width of the box
* h- The height of the box

cv2.rectangle(img,(x,y),(x+w,y+h),(randrange(128,256),randrange(128,256),randrange(128,256)),2)

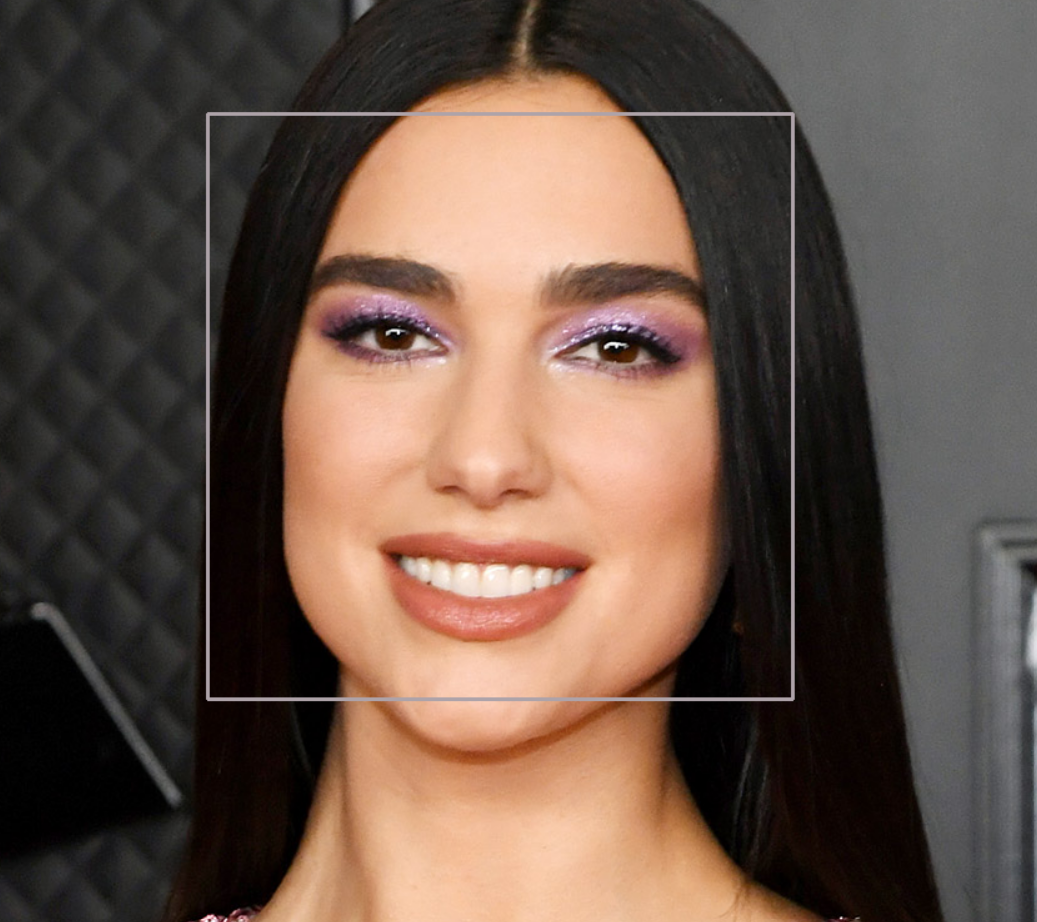
img= Image that we are using.

(x,y)= Starting coordinates. Coordinates of top left corner

(x+w,y+h)= Coordinates of bottom right corner.

(randrange (128,256),…)= Colour of the box. From colour code 128 to 255 any random colour is chosen for the box.

2= Is the width of the lines of the box.

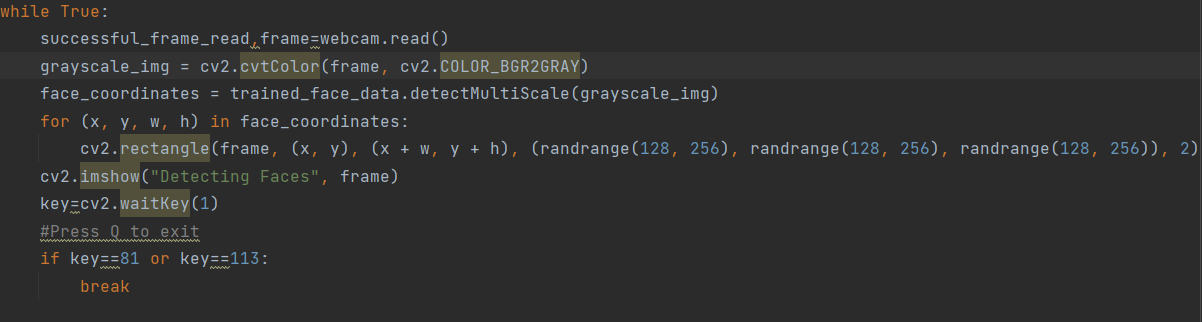


**Detecting faces in videos-**

Videos are basically multiple frames of images being displayed at a very fast rate, so we will treat every frame as an image and keep on checking those frames using a while loop. It’s like we are checking multiple images at once.



s1 = The file path to the video. If s1=0 then it will use the webcam.



* successful\_frame\_read = just bool to check if the frame was read successfully. Won’t use it but necessary to write.
* frame is like “img” in the face detection in images program. We treat this as an image to check for a face.
* waitKey(1) = Uses real time webcam or video footage. If we don’t write 1 then it will wait for you to press a key to check the next frame.
* To exit we need to break the while loop. Therefore ‘q’ and ‘Q’ were used (Q = Quit).

**Detecting faces from webcam-**

webcam=cv2.VideoCapture(0)

Only this is changed rest is same as the program used to detect faces in videos.

**Limitations:**

* The algorithm we are using can only be used to for frontal faces. We can use other algorithms if we want to detect side profile, masked faces etc.
* The algorithm might not be very accurate.
* This will not be able to detect masked faces.