DROWSINESS DETECTION-SYNOPSIS

PREPROCESSING STEPS ON THE IMAGE

Libraries Referred : OpenCV, dlib, imutils, NumPy(Python)

1-Reading the image

2-Converting it to Grayscale

In many objects, colour isn't necessary to recognize and interpret an image. Grayscale can be good enough for recognizing certain objects.

Because colour images contain more information than black and white images, they can add unnecessary complexity and take up more space in memory (colour images are represented in 3 channels(r,g,b) while a grayscale image contains only 1 channel, which means that converting it to grayscale reduces the number of pixels that need to be processed).

Colour adds unnecessary noise to it is easier to detect faces in a grayscale image.

(There are situations where colour will be required to get information)

3-Manupilating Images

After that, the image manipulation used, in which the resizing, rotating ,cropping, blurring and sharpening of the images done if needed .

There are cases where an algorithm only reads an image of a fixed defined size and hence all the images have to be resized in the given dataset before the algorithm is implied.

TECHNIQUES TO DETERMINE EYES IN THE ENTIRE IMAGE

Before detecting the eyes in the image, we first need to detect the face.

There are many ways to localize a face in an image: Using OpenCV's in-built Haar Cascades, HOG + Linear SVM, CNN, DNN etc.

By using the HOG (Histogram of oriented Gradients) + Linear SVM (Support Vector Machines) we detect faces in the image first.

The SVM algorithm creates a line or a hyperplane which separates the data into classes. According to the SVM algorithm we find the points closest to the line from both the classes.

These points are called support vectors. Now, we compute the distance between the line and the support vectors. This distance is called the margin. Our goal is to maximize the margin. The hyperplane for which the margin is maximum is the optimal hyperplane.

The basic idea of HOG is dividing the image into small connected cells, like a grid, then computes histogram for each cell. Thereafter, all histograms together form a feature vector i.e., it forms one histogram from all small histograms which is unique for each face.

[This detection can be done by dlib's inbuilt function get_frontal_face_detector() with just one line of code.]

Then we bound the face by a bounding box which makes it easier afterward to detect the eyes.

The bounding box is made by the rectangle() function in OpenCV by passing the (x,y) co-ordinates of the diagonals of the rectangle to be bounded by the face.

[Dlib has its own attributes to extract the bounding box coordinates. They are

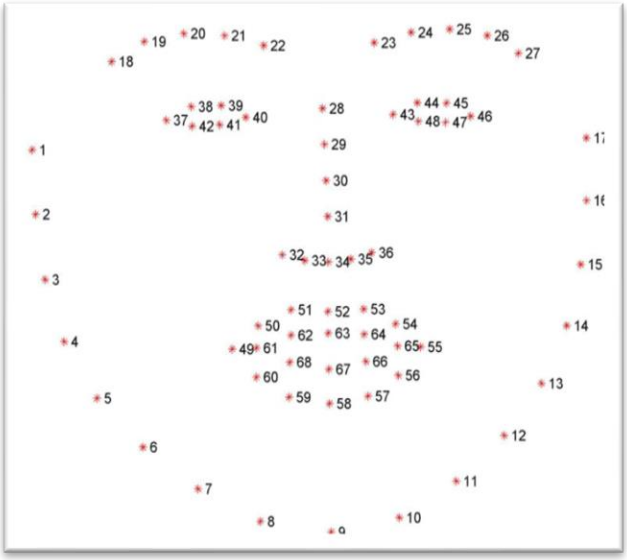
left(),top(),right() and bottom()]

DETECTING THE EYE

After finding the face we the look for the pair of eyes in the image using dlib's facial landmark detector.

The facial landmark detector implemented inside dlib produces 68 *(x, y)*-coordinates that map to *specific facial structures*. These 68-point mappings were obtained by training a shape predictor.

Below we can visualize what each of these 68 coordinates map to:



In this image we can clearly see every facial landmark has a specific set of co-ordinates which can then be accessed later to find various other information.

In our case we will only use the co-ordinates of the eyes for further calculations.

Here we use the points (37-48) for the eyes.

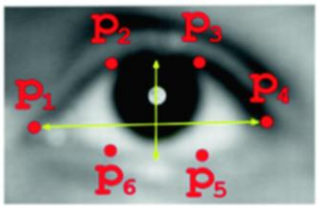
We use the NumPy library to then get the co-ordinates of these points.[Can be done by using imutills'inbuilt shape_to_np() function which returns a NumPy array].

Since we do not want the other facial landmarks we can remove them.

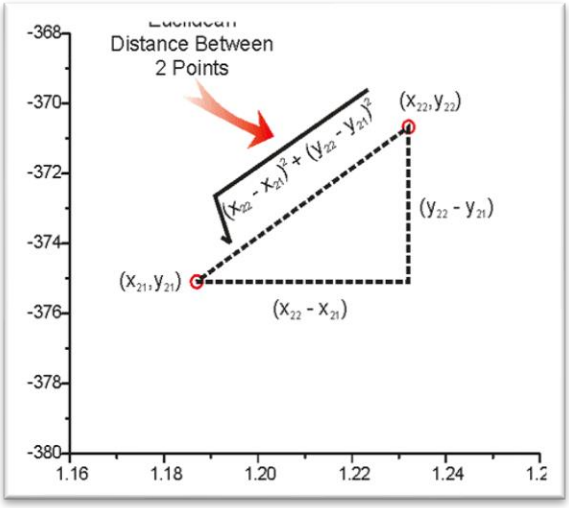
TECHNIQUES TO DETECT DROWSINESS

We use the co–ordinates of the eyes found earlier to calculate the **EYE ASPECT RATIO(EAR).**

EAR, as the name suggests, is the ratio of the length of the eyes to the width of the eyes. The length of the eyes is calculated by averaging over two distinct vertical lines across the eyes as illustrated in the figure below.



$$\text{EAR} = \frac{\|p_2 - p_6\| + \|p_3 - p_5\|}{2\|p_1 - p_4\|}$$



We calculate the Euclidean Distance between the vertical set of points [p2 ,p6] and [p3,p5] and the horizontal distance between points p1 and p4 w.r.t the above image.

Then we take the average of the vertical distances and divide it by the horizontal distance.

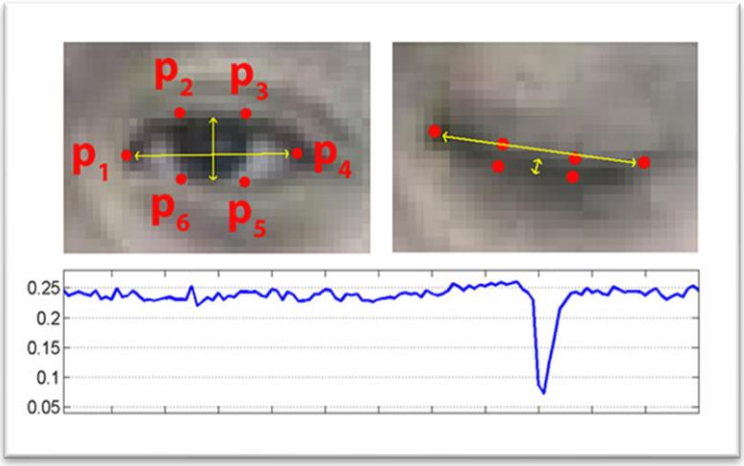
[Can be done using np.linalg.norm(pt2-pt1) in NumPy]

This is the EAR of one eye. We follow the same procedure for the other and the mean EAR of both the eyes is the required EAR for the given set of eyes.

If the EAR value is less than a certain threshold value for more than

100-400ms (the average time of a blink) then we can say that the person is drowsy.

The threshold value for a normal opened eye is about 0.25 . If this value is lower, we can say that the person is drowsy. The above graph represents the EAR values at particular intervals if time.



OTHER METHODS

1-We can get more accurate results if we compute for the **Pupil Circularity(PUC)** of the eyes.

PUC is a measure complementary to EAR, but it places a greater emphasis on the pupil instead of the entire eye. For example, someone who has their eyes half-open or almost closed will have a much lower pupil

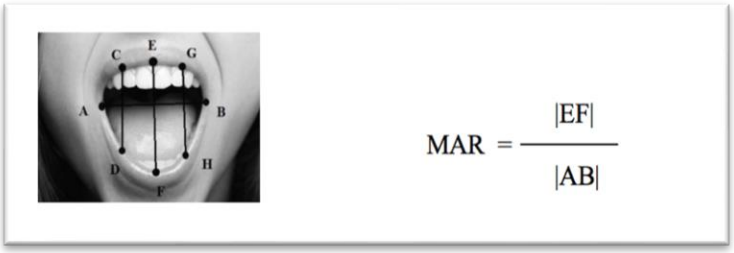
$$Circularity = \frac{4 * \pi * Area}{perimeter^2} \qquad Area = \left(\frac{\text{Distance(p2, p5)}}{2}\right)^2 * \pi$$

$$Perimeter = Distance(p1,p2) + Distance(p2,p3) + Distance(p3,p4) + Distance(p4,p5) + Distance(p5,p6) + Distance(p6,p1)$$

circularity value versus someone who has their eyes fully open due to the squared term in the denominator.

2-To see whether a person is drowsy in general(not just the eyes) we can compute the **Mouth Aspect Ratio (MAR).**

Computationally like the EAR, the MAR, as you would expect, measures the ratio of the length of the mouth to the width of the mouth. As an individual becomes drowsy, they are likely to yawn and lose control over their mouth, making their MAR to be higher than usual in this state.



(Using MAR and EAR we can calculate the Mouth to Eye Ratio(MOE) which also provides accurate results.)

CONCLUSION

I chose the HOG + Linear SVM Classifier over solutions because SVMs are simpler to train and HOG provides accurate images of the face (in this case).

SVM is used for both regression and classification problems. In this case it is a classification problem where the we can say if a person is drowsy or not by looking at the state of his eyes.

There are popular algorithms like the Haar Cascade Algorithm which is inbuilt in OpenCV but is not accurate enough and sometimes does not give the desired results. Moreover, it is an old algorithm and it requires fetching of .xml files for different facial landmarks.

Another popular algorithm is CNN (Convolutional Neural Network). This algorithm is actually better than the one used as detects faces even when they are not perfectly frontal to a good extend but, Unfortunately, it is not suitable as it is meant to be executed on a GPU. To get the same speed as the HOG based detector you might need to run on a powerful Nvidia GPU which is quite expensive and hence not efficient for a drowsiness detector.