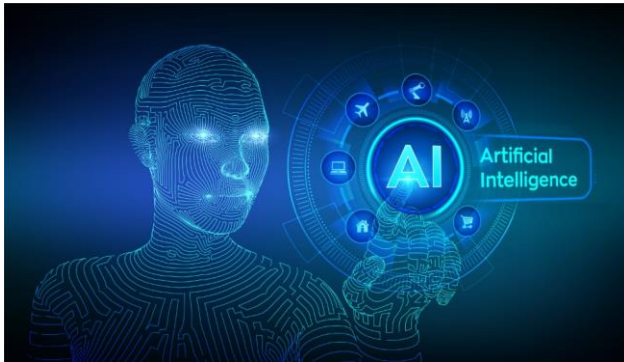


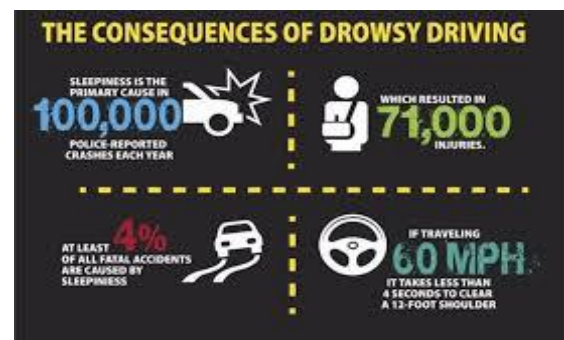
AI, the problem solver.....



Nowadays, Roadways is one of the most crucial means of transport in these days, On an average, Indians spend 7% of their day in commuting to office, but here is a question for all of us? Is office commutation is the only means of transportation. Not absolutely, there are hell lot of people who spend most of their lives spending by driving. So there will be lot of day-night journey been carried by many truck, bus drivers. Definetely these drivers feel tired when they keep driving all through the day but drivers don't really get alerted when they get fatigued.

So how can we address this problem and give a solution? Yes there is a solution for this through technology. Through technology we solve lot of problem nowadays and such a solution is here to alert the drivers when they fall asleep, get drowsed and fatigued. How are we going to do that through technology???? Keep your excitement up and here it is.

Fatigue/Sleep detection is an Artificial Intelligence based solution which can be used to alert all these drivers who get drowsed or fatigued. a computer vision system that can automatically detect driver drowsiness in a real-time video stream and then play an alarm if the driver appears to be drowsy.



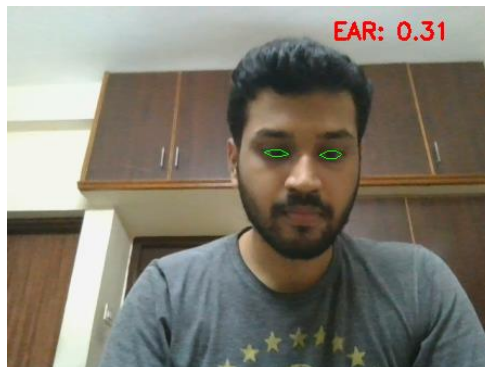
The drowsiness detector algorithm:

The general flow of our drowsiness detection algorithm is fairly straightforward.

- First, we'll setup a camera that monitors for faces.
- If a face is found, we apply facial landmark detection and extract the eye regions.



- Now that we have the eye regions, we can compute the eye aspect ratio to determine if the eyes are closed. (The detail of how we compute eye aspect ratio is mentioned below)



- If the eye aspect ratio indicates that the eyes have been closed for a sufficiently long enough amount of time, we'll sound an alarm to wake up the driver.



Building the drowsiness detector with OpenCV

To start our implementation, open up a new file, name it detect_drowsiness.py.

Then import Necessary packages.(These are the modules which helps us to do our system)

```
#Import Necessary pack
from scipy.spatial import distance as dist
from imutils.video import VideoStream
from imutils import face_utils
from threading import Thread
import numpy as np
import pyglet
import argparse
import imutils
import time
import dlib
import cv2
from playsound import playsound
```

Import our required Python packages.

```
def sound_alarm(path):
    # play an alarm sound
    music = pyglet.resource.media(alarm.wav')
    music.play()
    pyglet.app.run()
```

This is used to import the alarm sound from our local system.

```
def eye_aspect_ratio(eye):
    # compute the euclidean distances between the two sets of
    # vertical eye landmarks (x, y)-coordinates
    A = dist.euclidean(eye[1], eye[5])
    B = dist.euclidean(eye[2], eye[4])
    # compute the euclidean distance between the horizon
    # eye landmark (x, y)-coordinates
    C = dist.euclidean(eye[0], eye[3])
    # compute the eye aspect ratio
    ear = (A + B) / (2.0 * C)
    # return the eye aspect ratio
    return ear
```

We also need to define the eye aspect ratio function which is used to compute the ratio of distances between the vertical eye landmarks and the distances between the horizontal eye landmark.

```
# define two constants, one for the eye aspect ratio to indicate
# blink and then a second constant for the number of consecutive
# frames the eye must be below the threshold for to set off the
# alarm
EYE_AR_THRESH = 0.3
EYE_AR_CONSEC_FRAMES = 48

# initialize the frame counter as well as a boolean used to
# indicate if the alarm is going off
COUNTER = 0
ALARM_ON = False
```

If the eye aspect ratio falls below this threshold (EYE_AR_THRESH), we'll start counting the number of frames the person has closed their eyes for.

If the number of frames the person has closed their eyes in exceeds, we'll sound an alarm.

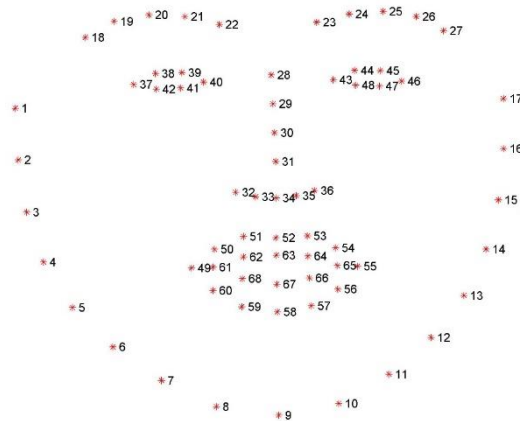
Experimentally, I've found that an EYE_AR_THRESH of 0.3 works well in a variety of situations

COUNTER, defines the total number of consecutive frames where the eye aspect ratio is below EYE_AR_THRESH. If COUNTER exceeds EYE_AR_CONSEC_FRAMES, then we'll update the boolean ALARM_ON.

The dlib library ships with a **Histogram of Oriented Gradients-based face detector** along with a **facial landmark predictor** (references for these two are attached at the end)— we instantiate both of these in the following code block:

```
# initialize dlib's face detector (HOG-based) and then create
# the facial landmark predictor
print("[INFO] loading facial landmark predictor...")
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("D:\cv\drowsiness-
detection\shape_predictor_68_face_landmarks.dat")
```

The facial landmarks produced by dlib are an indexable list, as I described here



Therefore, to extract the eye regions from a set of facial landmarks, we simply need to know the correct array slice indexes:

```
# grab the indexes of the facial landmarks for the left and
# right eye, respectively
(lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
(rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
```

Now we can start our core of our fatigue detector:

```
# start the video stream thread
print("[INFO] starting video stream thread...")
vs = VideoStream(src=args["webcam"]).start()
time.sleep(1.0)

# loop over frames from the video stream
while True:
    # grab the frame from the threaded video file stream, resize
    # it, and convert it to grayscale
    # channels)
    frame = vs.read()
    frame = imutils.resize(frame, width=450)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # detect faces in the grayscale frame
    rects = detector(gray, 0)
```

Here we instantiate our `VideoStream` using the supplied `–webcam` index, which then reads the next frame, which pre-process by resizing it to have a width of 450 pixels and converting it to grayscale, and then we apply `dlib`'s face detector to find and locate the face in the image.

The next step is to apply facial landmark detection to localize each of the important regions of the face:

```

# loop over the face detections
for rect in rects:
    # determine the facial landmarks for the face region, then
    # convert the facial landmark (x, y)-coordinates to a NumPy
    # array
    shape = predictor(gray, rect)
    shape = face_utils.shape_to_np(shape)

    # extract the left and right eye coordinates, then use the
    # coordinates to compute the eye aspect ratio for both eyes
    leftEye = shape[lStart:lEnd]
    rightEye = shape[rStart:rEnd]
    leftEAR = eye_aspect_ratio(leftEye)
    rightEAR = eye_aspect_ratio(rightEye)

    # average the eye aspect ratio together for both eyes
    ear = (leftEAR + rightEAR) / 2.0

```

We loop over each of the detected faces in our implementation (specifically related to driver drowsiness), we assume there is only *one* face — the driver — but I left this for loop in here just in case you want to apply the technique to videos with *more than one* face.

For each of the detected faces, we apply dlib's facial landmark detector and convert the result to a NumPy array

We can then *visualize* each of the eye regions on our frame by using the `cv2.drawContours` function below — this is often helpful when we are trying to debug our script and want to ensure that the eyes are being correctly detected.

```

# compute the convex hull for the left and right eye, then
# visualize each of the eyes
leftEyeHull = cv2.convexHull(leftEye)
rightEyeHull = cv2.convexHull(rightEye)
cv2.drawContours(frame, [leftEyeHull], -1, (0, 255, 0), 1)
cv2.drawContours(frame, [rightEyeHull], -1, (0, 255, 0), 1)

```

Finally, we are now ready to check to see if the person in our video stream is starting to show symptoms of drowsiness:

```

# check to see if the eye aspect ratio is below the blink
# threshold, and if so, increment the blink frame counter
if ear < EYE_AR_THRESH:
    COUNTER += 1

    # if the eyes were closed for a sufficient number of

```



```

# then sound the alarm
if COUNTER >= EYE_AR_CONSEC_FRAMES:
    # if the alarm is not on, turn it on
    if not ALARM_ON:
        ALARM_ON = True
    # draw an alarm on the frame
    cv2.putText(frame, "DROWSINESS ALERT!", (10, 30),
                 cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
    playsound('alarm.wav')
# otherwise, the eye aspect ratio is not below the blink
# threshold, so reset the counter and alarm
else:
    COUNTER = 0
    ALARM_ON = False

```

we make a check to see if the eye aspect ratio is below the “blink/closed” eye threshold, `EYE_AR_THRESH`.

If it is, we increment `COUNTER`, the total number of *consecutive frames* where the person has had their eyes closed.

To handle playing the alarm sound, provided an `—alarm` path was supplied when the script was executed. We take special care to create a *separate thread* responsible for calling `sound_alarm` to ensure that our main program isn’t blocked until the sound finishes playing.

Next draw the text `DROWSINESS ALERT!` on our frame — again, this is often helpful for debugging, especially if you are not using the `playsound` library.

The final code block in our drowsiness detector handles displaying the output Frame to our screen.

```

# draw the computed eye aspect ratio on the frame to help
# with debugging and setting the correct eye aspect ratio
# thresholds and frame counters
cv2.putText(frame, "EAR: {:.2f}".format(ear), (300, 30),
             cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
# show the frame
cv2.imshow("Frame", frame)
key = cv2.waitKey(1) & 0xFF

# if the `q` key was pressed, break from the loop
if key == ord("q"):
    break
# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()

```

Summary:

In this we saw the demonstration of how to build a drowsiness detector and if you wish to implement the same you can pull the code with explanation from the given github repository link and implement it by yourself.

Github link: <https://github.com/Vimal0307/Fatigue-Detection>

The working proof of concept has been uploaded in YouTube and you can refer it to the link mentioned below:

POC video link: <https://www.youtube.com/watch?v=r-BMKeuvn18&feature=youtu.be>

So to conclude with, our fatigue detector hinged on two important computer vision techniques:

- Facial landmark detection
- Eye aspect ratio

You can refer to some of these topics through videos and research papers which will be in references section and can get more insights on computer vision.

References:

[1] Histogram of Oriented Gradient feature vector calculation which is used for object detection

https://www.youtube.com/watch?v=28xk5i1_7Zc&t=71s

[2] Real time eye blink detection using facial landmarks

<http://vision.fe.uni-lj.si/cvww2016/proceedings/papers/05.pdf>

[3] Learn Histogram of Oriented Gradients with Open CV

<https://www.learnopencv.com/histogram-of-oriented-gradients/>

[4] Dlib library documentation for Dlib related queries

<http://dlib.net/>

[5] Eye blink detection using OpenCV, Python and Dlib

<https://www.pyimagesearch.com/2017/04/24/eye-blink-detection-opencv-python-dlib/>