WSD LAB PROJECT - GROUP 2 - REPORT

DOMAIN - TRANSPORTATION

Group members

Aarsh Goyal	19IM30001
Abhishek Prakash Mahalunge	19IM30003
Adarsh Modi	19IM30004
Ankita Radhakrishnan	19IM30005
Ayush Raj	19IM30006
Chauhan Param Jigneshbhai	19IM30007
Debjit Mandal	19IM30008

Aim of our project:

1. To detect drowsiness in drivers by detecting eyes and lips, and trying to see if they show signs of sleepiness such as closed or nearly closed eyes, yawning and so on.
2. To estimate the head pose of the driver, and to warn if they are looking away from the forward direction for too long.
3. To raise an alarm if a phone is detected by the user.
4. To implement the first 3 features in low light conditions (as is when most people feel drowsy).

Code

Available at the following Google drive link - https://drive.google.com/drive/folders/1k4AnHUjps8oRYKfwT7htuyfKUx9L_4ex?usp=sharing

The folder driver-safety contains:

1. main.py - our code with functions for face detection, checks drowsiness of eyes, yawning and head pose estimation
2. alarm1.wav - the alarm tone we have used throughout our code
3. classes.txt - contains information about the 80 classes of COCO objects that YOLO is capable of detecting, which we used in mobile phone detection
4. facial_landmarks_68markup.jpg - image which shows the 68 landmarks features as defined in dlib
5. fun.py - contains the code for the YOLO detection part which basically includes the Darknet 53 CNN model
6. lowlight_enhancement.ipynb - contains the Colab notebook used for image enhancement
7. yolov3-320.cfg, yovlov3-320.weights - supporting files for YOLO
8. shape_predictor_68_face_landmarks.dat - supporting file for using face detector from dlib
9. Zero-DCE-1-master - the code for the model, which we have modified a bit to fit our use
10. Our project report - WSD project report - Group 2.pdf (available at the link https://docs.google.com/document/d/16JOZADSwwwf3aY7QRI6OFmbiv9wMVbB93cc_XoGbjaQ/edit?usp=sharing)
11. Our final presentation, which summarizes all we have done - Driver Safety (Final).pdf (available at the link https://docs.google.com/document/d/1NWUZKcfsl3yF_rP2qzy2pLfN7XSqWZqFKiQebApti4o/edit?usp=sharing)
12. final_video.mp4 - demonstration video

Drowsiness Detection - presented on February 12

Main Idea

The main idea of this project is to detect drowsiness with the help of eye closure (distance between the eyelids).

```
cap = cv2.VideoCapture(0)
hog_face_detector = dlib.get_frontal_face_detector()
dlib_facelandmark=dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
while True:
        _, frame = cap.read()
        height,width = frame.shape[:2]
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

      faces = hog_face_detector(gray)
      for face in faces:
          face_landmarks = dlib_facelandmark(gray, face)
```

We will use the dlib library for detecting faces and predicting the facial landmarks of a person. Using the detector and predictor, after converting the frame (video capture) into grayscale, we detect the facial landmarks on the faces.

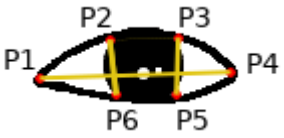
To the left is the default representation of the numbered facial landmarks.

Eyes:

Now that we have already detected the facial landmarks, we can concentrate on the eyes of the face in order to detect blinks .

Before that we can define a function that will help us to calculate facial landmark point coordinates (x,y) for the eyes, draw them on our frame (video capture) and return a numpy array with the corresponding points. In order to build a function that will help us to determine if there is closure between the lids or not, we could use the Eye Aspect Ratio value.

Let us enumerate the facial landmarks of a single eye with points P1 to P6 as shown below.



We will calculate the two vertical distances between points P2-P6 (A) and P3-P5 (B), and the horizontal (largest) distance between points P1-P4 (C).

So the Eye Aspect Ratio is defined as : (A+B)/(2*C)

```
def calculate_eye(eye):
    A = distance.euclidean(eye[1], eye[5])
    B = distance.euclidean(eye[2], eye[4])
    C = distance.euclidean(eye[0], eye[3])
    eye_aspect_ratio = (A+B)/(2.0*C)
    return eye_aspect_ratio
```

As shown in the facial landmarks image we can define two lists with numbers corresponding to the left and right eye.

left_eye = [36,37,38,39,40,41] right_eye=[42,43,44,45,46,47]

Now we are defining corresponding ranges to point to specific eyes, and append their coordinates to empty arrays.

```
        for n in range(36,42):
    x = face_landmarks.part(n).x
    y = face_landmarks.part(n).y
    leftEye.append((x,y))
            .
            .
            .
for n in range(42,48):
    x = face_landmarks.part(n).x
    y = face_landmarks.part(n).y
    rightEye.append((x,y))
            .
            .
```

The above explanation and calculation is referring to only one eye. In order to use both eyes Aspect Ratio values, we can find the average eye aspect ratio from the two eyes (left and right) and then round off the value to upto 2 decimals.

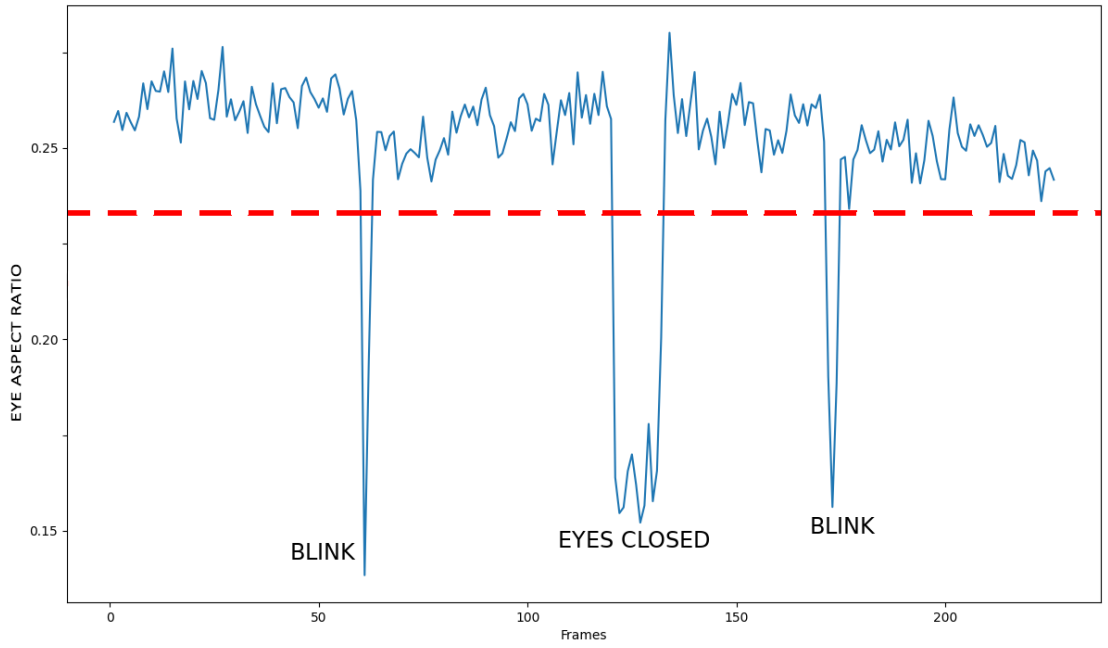
```
left_eye = calculate_eye(leftEye)
right_eye = calculate_eye(rightEye)

EYE = (left_eye+right_eye)/2
EYE = round(EYE,2)
```

Are the eyes drowsy or not ?

To decide if there is sufficient closure or not, we can set an Eye Aspect Ratio threshold. This means that if the current value is less than the threshold then it is a closure .

We can ”print” a plot showing the Aspect Ratio values for each frame. Below there is a simple example that I made to define my Eye Aspect Ratio threshold value.

In this plot (250 frames) we can see that most of the Eye Aspect Ratio values are less than 0.23 and there are 3 times that we see that difference. These 3 times correspond to 2 instant blinks and 1 time with eyes closed. So we will set our aspect ratio threshold value to 0.24 .




```
if EYE < 0.24:
  sleep_score=sleep_score+1
```

Actual drowsiness detection

By checking the eye aspect ratio values we can decide if there is drowsiness or not.
For every frame when the Eye Aspect Ratio becomes less than the threshold ratio, we increase the value of our variable(known as sleep_score)(initially set to 0)
by 1.

Whenever the sleep_score value increases more than 15 (our threshold time), it will show an Alert message on the screen, followed by the execution of an alarm sound until the sleep_score value returns back to 15 or less.

```
        if(sleep_score>15):
        cv2.putText(frame, "ALERT!", (10,50), font, 2, (0, 0, 255), 4)
        try:
          alarm.play()

        except:  # isplaying = False
          pass
if(sleep_score<15):
        alarm.stop()
```

Things we are working on:

1) To give an alert message even if the person yawns more than a particular number in a given time interval.

   How we are planning to do it – We are going to do it in the same way as we have done it for the eyes but in this case we have to just specify a particular threshold (maximum distance) and if the distance between the landmarks of mouth go beyond it then it will count it as yawning.

   And we have to find another threshold for the maximum number of yawns in a given time which can be considered normal. We are yet to decide these thresholds.

2) What we have done till now will work properly if the face has proper exposure to light but we want it to work properly even if there is a very little light.

   We are planning to do it by using other attributes of Opencv and by proper variations of brightness and contrast we will make sure that the detector perceives a face which can be analysed properly.

Date - 19th February

YAWNING

For detection of drowsiness through Yawning we have to check the amount of time a person was yawning in a given time interval. If it is coming more than a limit then it will give an alert.

How is it different from drowsiness detection through Eyes:

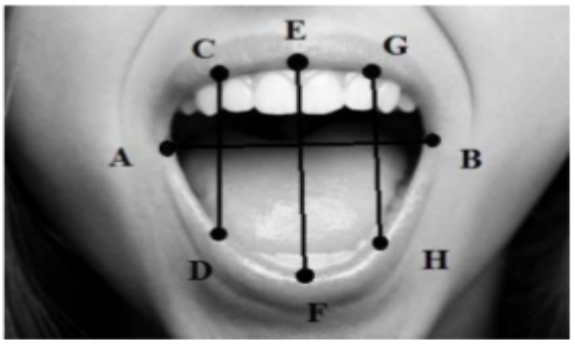
We were counting the number of continuous frames in which the eye aspect ratio was less than a particular number (0.24). But for Yawning we just cannot count the number of continuous frames in which our mouth was open. Instead, we have to look at the total number of frames in which our mouth's aspect ratio was more than a particular threshold in a given time interval.

- Defining an interval is necessary because otherwise a person on a long drive may yawn many times and the total time he/she yawns may cross the threshold and it will give an unnecessary alert.

What we have to do:

To determine the yawning parameter, the aspect ratio of the mouth is calculated. It is calculated by the following formula, the code for which is also available below the image:

$$MAR = \frac{|CD| + |EF| + |GH|}{3 * |AB|}$$



```
def underlayer_calculate_lips(lips):
        A = distance.euclidean(lips[1], lips[7])
        B = distance.euclidean(lips[3], lips[5])
        C = distance.euclidean(lips[2], lips[6])
        D = distance.euclidean(lips[0], lips[4])
        lips_aspect_ratio = (A+B+C)/(3.0*D)
        return lips_aspect_ratio
```

How we have given an interval of 30 Minutes (30frames/sec * 60 sec * 30 minutes = 54000 frames):

We initialized a variable frame_counter to 0 and it will start increasing (as frame_counter= frame_counter+1) from the time when the person yawns for the first time. And in this time interval if the person was yawning for more than our threshold interval (300 frames) then it will give an alert. Otherwise after 30 minutes (frame_counter > 54000 frames) , frame_counter is again initialized to 0.

```
    if(frame_count>0):
            frame_count=frame_count+1

    if (frame_count > 54000):           #54000=30*60*30
            sleep_score2=0
            frame_count=0
```

How we have given a threshold for Yawning in the interval of 30 minutes:

Actually we don't yet have found any scientific proof for our threshold but just for conducting this we have taken a total of 300 frames .That means if the person yawns for a total of 10 seconds in 30 minutes it will show an alert. We have initialized a variable sleep_score2 that will one by one keep the total number of continuous frames in which our mouth was open beyond a certain mouth aspect ratio (we have taken 0.5). Once alarms are stopped (by the user) it will again be initialized to 0.

It will be zero again if half an hour passes after the first time we have yawned.

Creating a list of coordinates(by the help of which mouth aspect ratio will be calculated in each frame):

```
underlips = []
for n in range(60,68):
        x = face_landmarks.part(n).x
        y = face_landmarks.part(n).y
        underlips.append((x,y))
        next_point = n+1
        if n == 67:
                next_point = 60
        x2 = face_landmarks.part(next_point).x
        y2 = face_landmarks.part(next_point).y
        cv2.line(frame,(x,y),(x2,y2),(0,255,123),1)
```

```
under_lips = underlayer_calculate_lips(underlips)


LIPS = under_lips
LIPS = round(LIPS,2)
```

For both the upper and the lower lips, it can be done with the help of the following code:

```
underlips = []
for n in range(60,68):
        x = face_landmarks.part(n).x
        y = face_landmarks.part(n).y
        underlips.append((x,y))
        next_point = n+1
        if n == 67:
                next_point = 60
        x2 = face_landmarks.part(next_point).x
        y2 = face_landmarks.part(next_point).y
        cv2.line(frame,(x,y),(x2,y2),(0,255,123),1)

upperlips = []
for n in range(48,60):
        x = face_landmarks.part(n).x
        y = face_landmarks.part(n).y
        upperlips.append((x,y))
        next_point = n+1
        if n == 59:
                next_point = 48
        x2 = face_landmarks.part(next_point).x
        y2 = face_landmarks.part(next_point).y
        cv2.line(frame,(x,y),(x2,y2),(0,255,123),1)

upper_lips = upperlayer_calculate_lips(upperlips)

under_lips = underlayer_calculate_lips(underlips)

LIPS = (under_lips + upper_lips)
LIPS = round(LIPS, 2)
```

Applying the threshold condition for checking whether the person was yawning:

```
if LIPS > .50:
        sleep_score2=sleep_score2+1
        cv2.putText(frame,"Yawning",(10,height-80), font, 1,(255,0,0),1,cv2.LINE_AA)
        if (frame_count==0):
                frame_count=frame_count+1   #it will initiate the counting of total number of frames once we wi

else:
        cv2.putText(frame,"Not Yawning",(10,height-80), font, 1,(0,255,255),1,cv2.LINE_AA)
```

ALERT conditions:

```
if(sleep_score2)>50:         #For demonstration purpose it is 50
        #person is feeling sleepy so we beep the alarm
        cv2.putText(frame, "ALERT!", (10,50), font, 2, (0, 0, 255), 4)

        try:
            alarm.play()
        except:  # isplaying = False
            pass
```

Alarm will stop once the user presses a button (in our case it is ‘d’ character) and all the scores will be initiated to 0.

```
if (key==ord('d')):
        alarm.stop()
        sleep_score1=0
        sleep_score2=0


if (key == 27):
    break
```

Date - 5th March

Detecting facial features in low light conditions

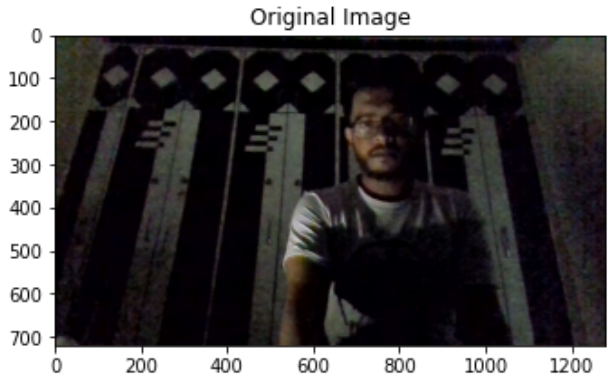
We found two approaches to brighten the video if there is a low light condition in order to detect the facial features. The first approach is a Convolutional Neural Network based approach which brightens the image. Through this approach, we were only able to brighten still images, but not videos.

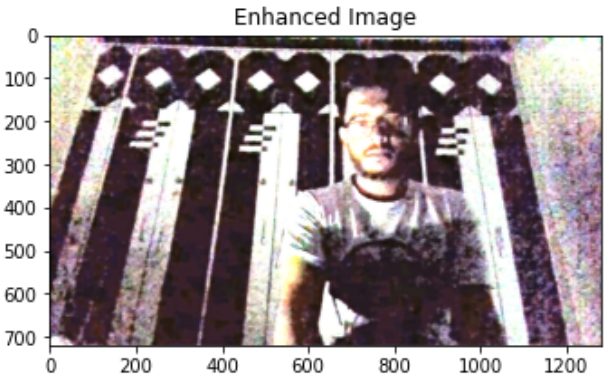
The second approach is to convert the video to grayscale and normalize the pixel contrast within a certain range (here, it is from 120 to 255). By doing so, the contrast of the image will be normalised, such that the facial features of the face are visible.

Once the facial features are detectable by the camera, the code runs in the normal manner and detects the facial features and measures their aspect ratios. It triggers the alarm when the aspect ratios cross the defined thresholds for a definite amount of time.

```
cv2.normalize(gray, gray, 120, 255, cv2.NORM_MINMAX)
frame = gray
```

CNN Model output for illuminating an image:





If the person is not looking directly at the screen (for occluded angles of the head)

We also tried to include the case where the head was at an occluded (not 90) angle.

- Using the help of a few other facial models, such as a caffe model and a tensor flow model.
- The caffe model is used along with its default text file so as to detect the face. This is also called the dnn model, and it basically detects that part of the image which can correspond to a face, and creates a rectangular/square marking around it

This is done with the help of the following code:

```
 blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)), 1.0, (300, 300), (104.0, 117.0, 123.0))
    model.setInput(blob)
    face = []
    ans = model.forward()
#to pass from one array to another

    for i in range(ans.shape[2]):
#taking length and width

        confidence = ans[0, 0, i, 2]
        if confidence > 0.5:
            box = ans[0, 0, i, 3:7] * np.array([w, h, w, h])
#w and h are the length and width of the frame respectively

            (x, y, x1, y1) = box.astype("int")
            face.append([x, y, x1, y1])  # stores the coordinates of the rectangle that surrounds the face
            # cv2.rectangle(frame, (x, y), (x1, y1), (0, 0, 255), 2)
```

cv2.dnn.blobFromImage(image, scaling-factor (here it is 1, otherwise it helps in normalizing the image), spatial size expected by the model which is 300 * 300, mean subtraction values from RGB channels) (typically is stored as a numpy array).

For mean subtraction values, 104.0, 117.0, 123.0 are the values which are given for this particular model. So, here the values are subtracted from each input channel of each pixel of the image. It then searches among

those pixels which have the same preprocessing so as to figure out which can possibly account for a face.

- landmarks = tensorflow.saved_model.load('landmark_model')

This loads the existing file into the code for use for identifying the landmarks, which turn out to be the same as the shape predictor landmarks

- We then try to maintain the shape as a square, by adjusting the coordinates if it is not already a square.

That is done in the following code:

```
offset = int(abs((faces[3] - faces[1]) * 0.1))
faces[1] += offset
faces[3] += offset
#this is applied to the vertical distance

# box indices = 0 denotes left x, 1 top y, 2 right x and 3 bottom y
w = faces[2] - faces[0]
h = faces[3] - faces[1]
# getting the width and height of the box to make sure its a square
diff = int(abs(w - h) / 2)
# getting a value to add or subtract by so as to convert it into a square

if w < h:
   faces[0] -= diff
   faces[2] += diff
   if ((h - w) % 2 == 1):
      faces[2] += 1
elif w > h:
   faces[1] -= diff
   faces[3] += diff
   if ((w - h) % 2 == 1):
      faces[3] += 1
```

- The tensorflow model has the same markings as that of our initial face predictor. However, its advantage is that it is possible for us to make the points move along with the movements of the face (of which we included some 45 degrees approx to either side and also upward movement in our code).

```
faceinframe = cv2.cvtColor(cv2.resize(frame[faces[1]:faces[3],
faces[0]:faces[2]], (128, 128)), cv2.COLOR_BGR2RGB)
```
#extracting from the rectangle the face,converting it to the size 128 x 128 as desired by the tensorflow model

```
predictions =
landmarks.signatures["predict"](tensorflow.constant([faceinframe],
dtype=tensorflow.uint8))
```
#using the predictor so as to get the face coordinates, not unlike the way the dlib face predictor did it

```
marks = np.array(predictions['output']).flatten()[:136]
```
#converting the 3D array of RGB into a 1D array using flatten() and then taking those coordinates upto 136

```
marks = np.reshape(marks, (-1, 2))
marks *= (faces[2] - faces[0])
marks[:, 0] += faces[0]
marks[:, 1] += faces[1]
marks = marks.astype(np.uint)
```
#here we basically compute the coordinates of the points as we did using the shape face predictor

```
for i in marks:
  cv2.circle(frame, (i[0], i[1]), 2, (0, 0, 255), -1, cv2.LINE_AA)
```
#drawing those points on the person's face so as to see upto what extent of movement we can use this code

The marks array stores the coordinates, so we would like to try using it as the input in our existing code and see if it works.

Topics we will be working on this week

1. Making the code detect facial features if the camera is at a height (as would typically be).
2. Working on improving the facial features detection in low light conditions.

Date - 12th March

Head Pose Detection

If the driver is not looking forward towards the road and is distracted for more than a particular time interval then the chances of road accidents increases which is another concern of safety of the driver and the passengers.
Therefore to solve this problem the feature of Head Pose Detection is introduced.
Here we are calculating the deviation that the driver's head makes relative to the alignment of the camera. We are recording the head position by tracking the X-coordinate of our nose tip.

The nose tip is determined by Landmark 34 of our model.

```
X=face_landmarks.part(34).x
```

We take initial 10 Frames and record the X- coordinates of nose tip in each frame and append it to a list .Initial_nose_tip_X=[]
And after that we take the average of all the 10 coordinates and fix it as our reference point for future comparison.

```
average_nose_X
```

We have printed them here :

```
[330, 328, 329, 329, 329, 330, 330, 329, 330, 330]
329
```

```
266
252
243
242
242
242
240
241
238
266
252
243
242
242
242
240
241
238
238
240
```

```
442
448
444
445
459
446
455
444
455
442
448
444
445
459
446
455
444
455
454
448
```

Now when the person turns right, X-coordinates start decreasing and as soon as the X-coordinated falls below the threshold we continuously increase the counter.
By analysing the coordinates we have taken a deviation of 75 units as our threshold on the right side.

Now when the person turns left, X-coordinates start increasing and as soon as the X-coordinated rises above the threshold we continuously increase the counter.

By analyzing the coordinates ,we decided that a deviation of 100 units can be taken as the threshold on the left side.

We have taken the threshold higher on the left side in comparison to that on the right side because In India most of the cars have driver seats on the right sides and thus they will turn toward left more.

Conditions implemented in following way:

If for a continued 40 frames, the person is not looking in the proper direction we start giving alert. (40 frames is just an assumption for demonstration).

```
if(X>(average_nose_X + 120)):
        count_X=count_X+1
if(X<(average_nose_X - 85)):
        count_X=count_X+1
if(X<(average_nose_X+120) and X>(average_nose_X - 85)):
        count_X=count_X-1
```

Date : 26/03/21

Mobile Detection

Sometimes we can see that the major reason for most of the accidents are due to drivers indulging in using mobile phones.
So we have tried to detect mobile phones whenever it comes under the frame and we will beep the alarm as alert.

We explored github regarding this and found various codes related to it and what we found most suitable has used Yolo.

YOLO: (**Paper referred to - YOLOv3: An Incremental Improvement Joseph Redmon and Ali Farhadi, University of Washington)**

YOLO is a clever convolutional neural network (CNN) for doing object detection in real-time. The algorithm applies a single neural network to the full image, and then divides the image into regions and predicts bounding boxes and probabilities for each region.

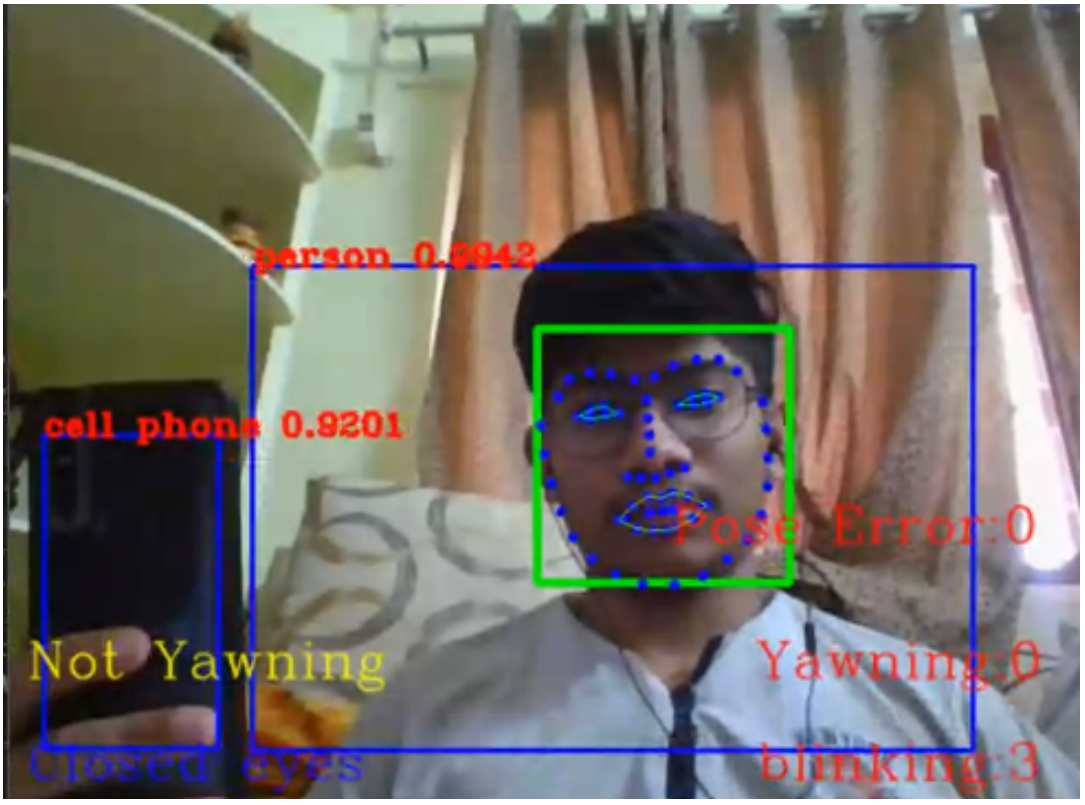
It is a pre-trained model.

We downloaded it this way.

```
def weights_download(out='models/yolov3.weights'):
    _ = wget.download('https://pjreddie.com/media/files/yolov3.weights',
out='models/yolov3.weights')
```

We have used a neural network Darknet

Darknet prints out the objects it detected, its confidence, and how long it took to find them. (this is used for training and testing).
YOLO will display the current FPS and predicted classes as well as the image with bounding boxes drawn on top of it.



- this helps to extract features from the object
- consists of 53 convolutional layers in total
- yolo basically uses a blob - that is it tries to take coordinates of 4 points all around the object (called the bounding box).

- Trained on a COCO datatset - (COCO stands for common objects in context - it includes a list of 80 objects ranging from people to knives to teddy bears).
- It can be used with image recognition and classification, and also for object detection.
- So here, we are using it to detect the objects, draw bounding boxes around it and also for classification - where we used it to identify what kind of object is being recognized
- Volume: 330K images (200K+ annotated); more than 2M instances in 80 object categories, with 5 captions per image, and 250,000 people with key points.

- it can detect any number of objects in the frame, but out of the 80 objects which are given in the list, only the phone and some food items are likely to be found with the driver, at the most. However, we sound the alarm only when the phone is detected. If it is possible to edit it such that we try to detect only mobile phones, we will be trying that over the week.

- So as to not alarm the driver, we would replace the alarm with a mild warning to pay attention.
- The size of the images in this model is 320*320.
- The following code shows how each frame is changed to fit according to the model's prerequisites.

```
img = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
img = cv2.resize(img, (320, 320))
img = img.astype(np.float32)
img = np.expand_dims(img, 0)
img = img / 255
```

$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$

- changing the color channel from BGR to RBG
- changing the size to the one required by the model
- converting the image to a numpy array for classification purposes
- adding a new axis at the beginning (index 0) using np.expand_dims()
- we are also normalizing the image to pixels between 0 and 1 (from between 0 to 255)

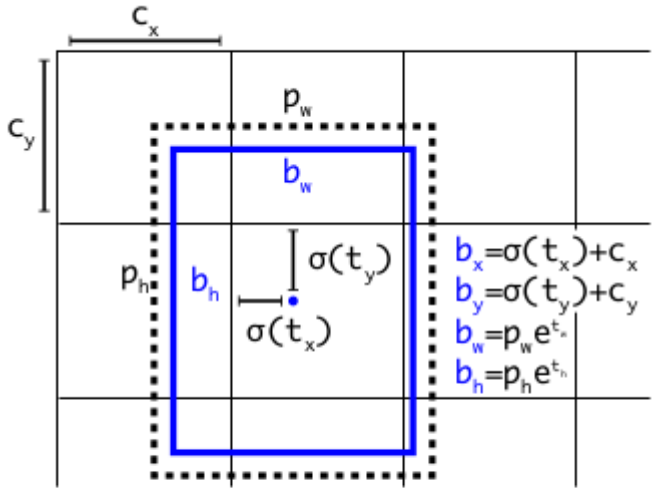


Figure 2. **Bounding boxes with dimension priors and location prediction.** We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function. This figure blatantly self-plagiarized from [15].

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

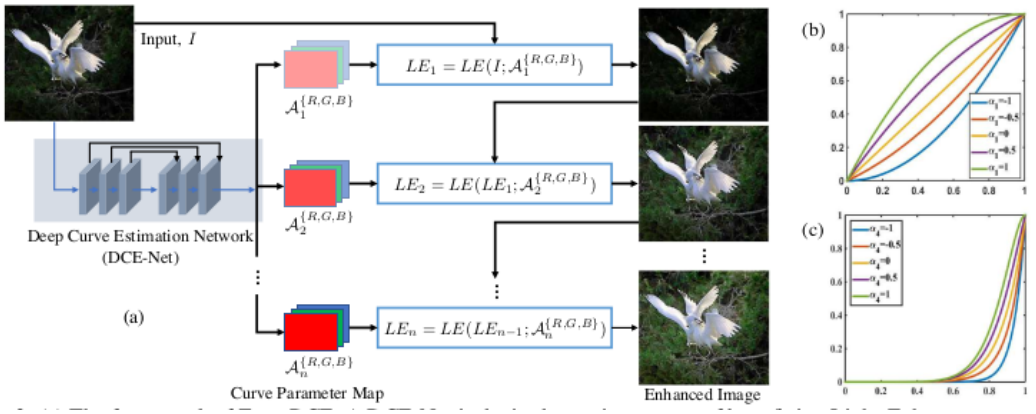
Table 1. **Darknet-53.**

Zero-DCE

(Paper referred to - Zero-Reference Deep Curve Estimation for Low-Light Image Enhancement Chunle Guo Chongyi Li Jichang Guo Chen Change Loy Junhui Hou Sam Kwong Runmin Cong BIIT Lab, Tianjin University, City University of Hong Kong, Nanyang Technological University, Beijing Jiaotong University)

- what zero means - it means that it does not use any reference, that it is does not use any paired or other unpaired data
- so it basically takes a low light image as input and produces curves as the output, in such a manner that it does not affect the values of the neighbouring pixels.
- curves are presumably that based on the pixel values of the images
- it is used to estimate a set of best-fitting light-enhancement curves for an input image
- it maps all the pixels of the image (which are in RGB format) and then applies various curves to obtain the final enhanced image
- There are various features which make an image of low quality in low light such as spatial consistency loss, exposure control loss, color constancy loss, illumination smoothness loss, and so on.
- Thus, this model basically tries to take care of these losses and thus with the help of curves produces the final enhanced image.

- One logic that has been applied to this model is that we try to find the pixel of maximum intensity in each channel - and we are able to crudely define an illumination structure for the same



Thus, we can see that the image is treated 3 times, and it gets enhanced at each time

- 3 things that matter to a light estimation (LE) curve:
- pixel value should be between 0 and 1 (helps to avoid information loss)
- curve should be monotonous
- it should be a simple curve

$LE(I(x); \alpha) = I(x) + \alpha I(x)(1 - I(x))$
where x denotes pixel coordinates, $LE(I(x); \alpha)$ is the enhanced version of the given input $I(x)$, $\alpha \in [-1, 1]$ is the trainable curve parameter, which adjusts the magnitude of LE-curve and also controls the exposure level.

How we can apply this model (which is meant for images) to videos:

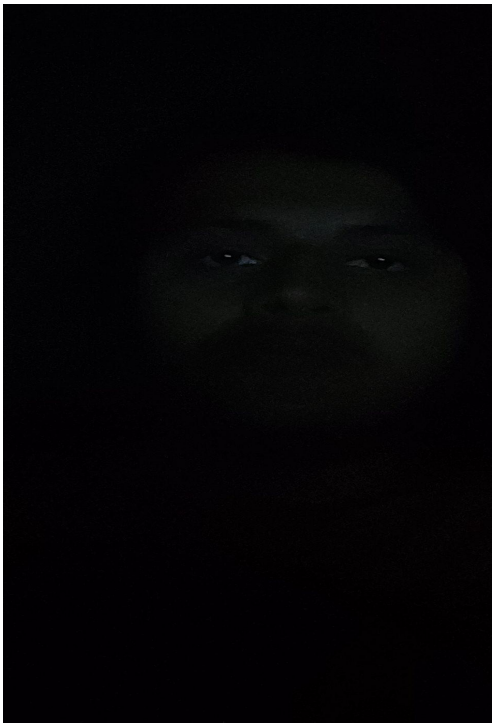
1. Loop over all frames in the video file
2. For each frame, pass the frame through the CNN
3. Obtain the predictions from the CNN
4. Maintain a list of the last *K* predictions
5. Compute the average of the last *K* predictions and choose the label with the largest corresponding probability
6. Label the frame and write the output frame to disk
7. This is called a rolling point average method

Our final touches to the project - by April 9

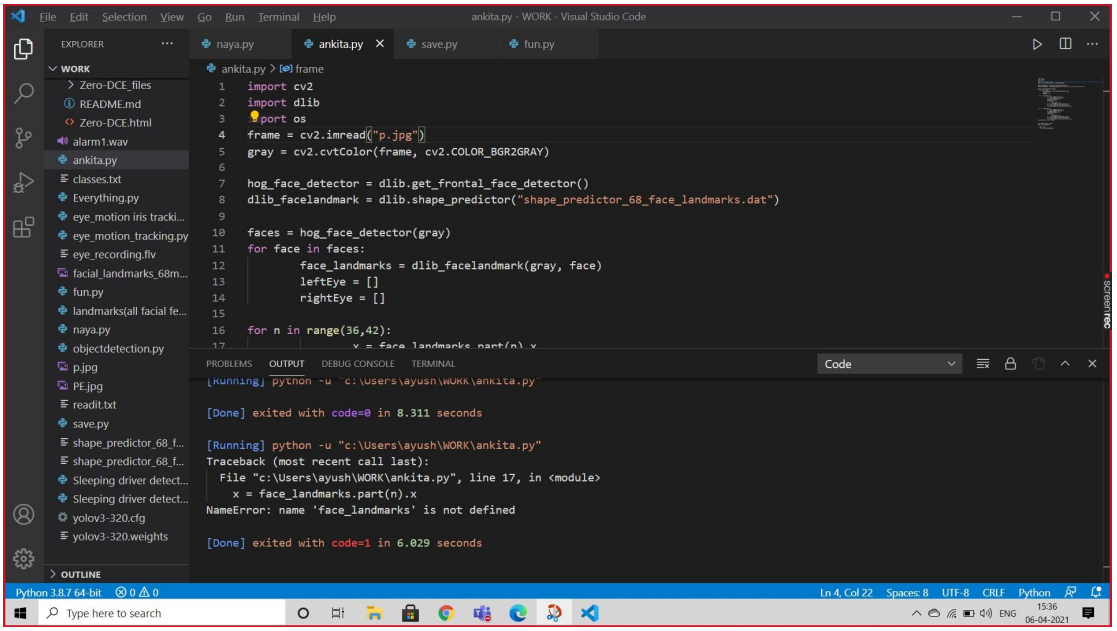
First, we tried taking images of ourselves in low light conditions, and then tried passing them through the model. We found out that while our usual code ran in the case of certain images, it failed in one with extremely low light conditions.

The following images show exactly how enhancement can take place.

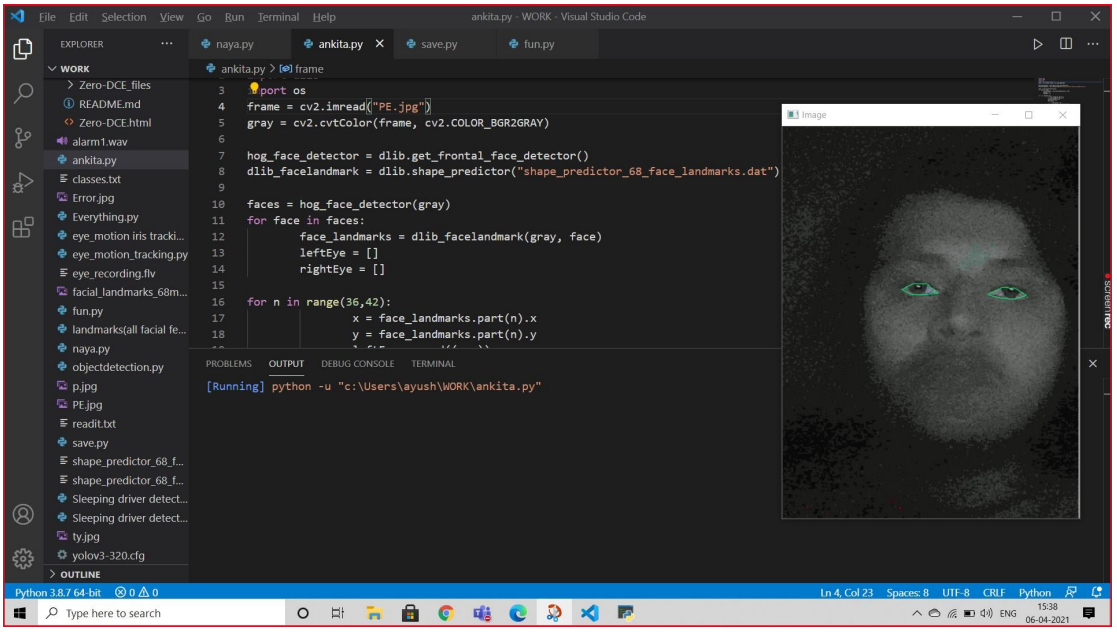
Our model detected his facial features in both cases (although lesser in the first case). However, it failed in the next case completely.
Since the original image had a significantly high quality and a size of 1MB, it was first compressed by about 97% to get a small size image, which is displayed on the right after being passed through the Zero-DCE model for enhancement.



When we passed the first raw image to our eye detection code, we got the following error:



This implied that the code did not detect any facial landmark features at all in the input image.
However, when the enhanced image was used, it detected the eyes properly.



Thus, while the Zero-DCE model is not necessary in all condiitons, it definitely helps to increase accuracy and even bring about detection.
How we implemented it in our quest:

1. We needed a GPU driver (nvidia) for running the model code. Since this was not available, we did the code for enhancement on Google Colab.
2. Since colab does not have an inbuilt video or audio device, we first recorded a video with our webcam in low light conditions, and gave that as input to the code.
3. We saved each frame of the video as an image in our parent folder, and passed that to the Zero-DCE model.Then, the output was again stored in the parent folder, and this was read as an input into Colab.
4. We then used each enhanced frame in the rest of the code as normal. On Colab, we only printed the values of the various parameters and did not display the frames.
5. For the sake of demonstration, we made a video from the enhanced frames. Due to the difference in fps between the input video and the enhanced frames, we exported this at a speed of 10fps so as to sync with the original video, and then recorded the usual code with that as input.