INDEX

1. INTRODUCTION

Drowsiness detection is a safety technology that can prevent accidents that are caused by drivers who fell asleep while driving.
The objective of this intermediate Python project is to build a drowsiness detection system that will detect that a person's eyes are closed for a few seconds. This system will alert the driver when drowsiness is detected.

1.1 Evolution:

This project is helpful in preventing accidents happen due to the drowsiness of the driver. So, we will build a system using Python, OpenCV which will alert the driver when he feels sleepy.

1.2 Software and Hardware Requirements:

The Software and Hardware Requirements are stated as below:

- **Technical Hardware Requirement**

Name	Details
Processor	Intel Pentium 4 CPU
RAM	512 MB

Table 1.3.1 Hardware Requirements

Hardware Requirements are the necessary tools that must be available for project to be done properly. This System is built on Intel Pentium 4 CPU, having clock speed of 3.0GHz, with RAM size 512MB, display is of 15-inch color monitor, and internet keyboard.

- **Software Requirement**

Name	**Details**
Operating System	Windows OS/Linux
Platform	Pycharm
Language	Python

Table 1.3.2 Software Requirements

The documentation of this system is done using MS-Office.

2. Project Description

In this Python project, we will be using OpenCV for gathering the images from webcam and feed them into a Deep Learning model which will classify whether the person's eyes are 'Open' or 'Closed'. The approach we will be using for this Python project is as follows

Step 1 – Take image as input from a camera.
Step 2 – Detect the face in the image and recognize the face.
Step 3 – Detect the eyes from landmarks.
Step 4 – Find EAR of both eyes.
Step 5 – Check EAR to check whether the person is drowsy.

- **The Dataset**

The dataset used for this model is created by us. To create the dataset, we wrote a script that captures eyes from a camera and stores in our local disk. We separated them into their respective labels 'Open' or 'Closed'. The data was manually cleaned by removing the unwanted images which were not necessary for building the model. The data comprises around 7000 images of people's eyes under different lighting conditions. After training the model on our dataset, we have attached the final weights and model architecture file "models/cnnCat2.h5".
Now, you can use this model to classify if a person's eye is open or closed.

- **Prerequisites**

The requirement for this Python project is a webcam through which we will capture images. You need to have Python (3.6 version recommended) installed on your system, then using pip, you can install the necessary packages.
OpenCV – pip install OpenCV-python (face and eye detection).
MatplotLib - pip install matplotlib(To plot a graph)
pyttsx3 - pip install pyttsx3(for text to speach)
face_recognition - pip install face-recognition(To recognize a face)

3. Requirements (Hardware, Software)

Building a machine learning / deep learning workstation can be difficult and intimidating. There are so many choices out there. Would you go for NVidia developer box and spend $15,000? or could you build something better in a more cost-effective manner. Which hardware is right for your requirements? How much RAM do you need? The questions are endless and there might be no right or wrong answers.

- **Hardware Requirements:**
  - Processor – Intel I3 2 core processor, 2.2 GHz with Turboboost upto 3.1 GHz..
  - Motherboard – Intel DH61HO.
  - RAM – 4 GB DDR4 2133 MHz.
  - 2 TB Hard Disk (7200 RPM) + 512 GB SSD.

4. Technical Documentation

4.1 EAR

As mentioned, to a certain extent, the state of eyes indicates whether the driver is drowsy or not. Because there are significant differences about time of eyes closed between awake and drowsy. In a method of ellipse fitting was proposed to describe the shape of pupil. As shown in Fig 1, the method segments the pupil with traditional image process firstly. Then, an ellipse is fitted with the white pixels, which represent the shape of eyes. Lastly, the ratio of the major and minor axes of the ellipse was used to evaluate the eyes state.

We noticed its performance might be limited by the following facts:

(1) The pixel values are sensitive. Changeable environment is easy to make image segmentation to be worse.

(2) In practical application, the pixel values between pupils and glasses are very close, which lead to false ellipse fitting. In this paper, we design a new more stable parameter based on Dlib toolkit to evaluate the state of driver's eyes. It is more stable and precise than ellipse fitting method thanks to avoiding the traditional image process.

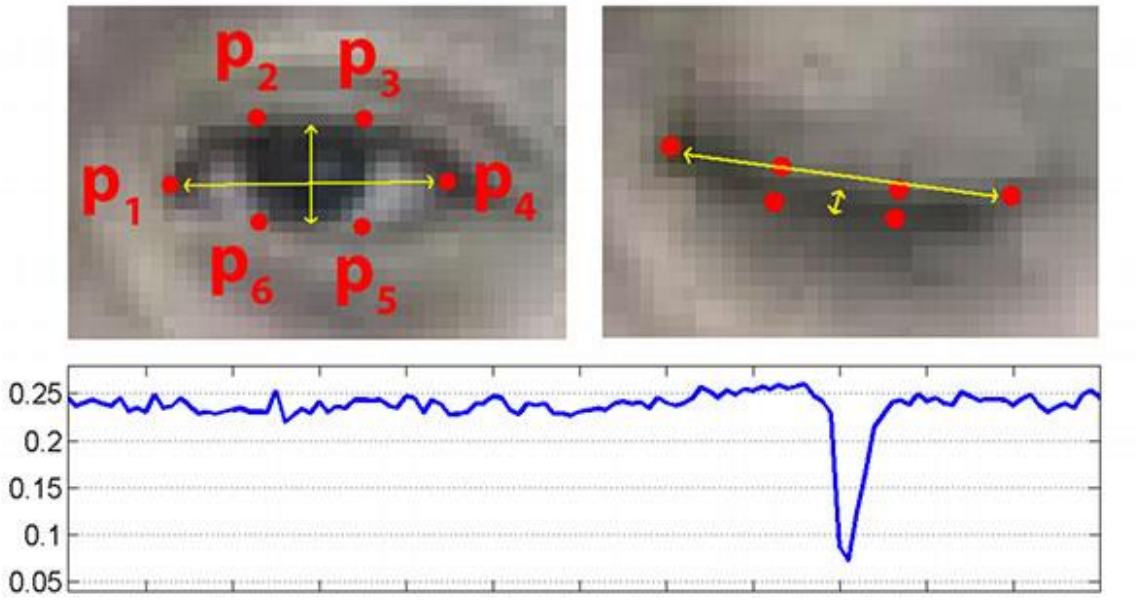


Fig. 1. Eyes landmarks. Upper: the distribution of eyes landmarks has significant differences. Bottom: the values of EAR at open and closed state.

4.2. OpenCV:

Deep Learning is a fast growing domain of Machine Learning and if you're working in the field of computer vision/image processing already (or getting up to speed), it's a crucial area to explore.
With OpenCV 3.3, we can utilize pre-trained networks with popular deep learning frameworks. The fact that they are pre-trained implies that we don't need to spend many hours training the network — rather we can complete a forward pass and utilize the output to make a decision within our application.

OpenCV does not (and does not intend to be) to be a tool for training networks — there are already great frameworks available for that purpose. Since a network (such as a CNN) can be used as a classifier, it makes logical sense that OpenCV has a Deep Learning module that we can leverage easily within the OpenCV ecosystem.

Popular network architectures compatible with OpenCV 3.3 include:

- GoogleLeNet (used in this blog post)
- AlexNet
- SqueezeNet
- VGGNet (and associated flavors)
- ResNet

4.3 Tkinter

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.
Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps −

- Import the Tkinter module.
- Create the GUI application main window.
- Add one or more of the above-mentioned widgets to the GUI application.
- Enter the main event loop to take action against each event triggered by the user.

Tkinter Widgets

Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called widgets.There are currently 15 types of widgets in Tkinter.

5. IMPLEMENTATION

Let's now understand how our algorithm works step by step.

Step 1 – Take Image as Input from a Camera

With a webcam, we will take images as input. So, to access the webcam, we made an infinite loop that will capture each frame. We use the method provided by OpenCV, cv2.VideoCapture(0) to access the camera and set the capture object (cap). cap.read() will read each frame and we store the image in a frame variable.

Step 2 – Detect the face in the image and recognize the face.

To detect the face in the image, we need to first convert the image into grayscale as the OpenCV algorithm for object detection takes gray images in the input. We don't need color information to detect the objects. We will be using haar cascade classifier to detect faces. This line is used to set our classifier face = cv2.CascadeClassifier('path to our haar cascade xml file'). Then we perform the detection using faces = face.detectMultiScale(gray). It returns an array of detections with x,y coordinates, and height, the width of the boundary box of the object. Now we can iterate over the faces and draw boundary boxes for each face.

```
gray=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
rects=det.detectMultiScale(gray,scaleFactor=1.1,minNeighbors=5,flags=cv2.CASCADE_SCALE_IMAGE)
```

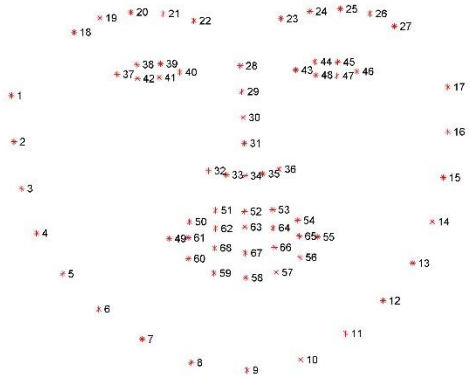
recognize the faces

```
match = face_recognition.compare_faces(encode_list, face, 0.5)
dist = face_recognition.face_distance(encode_list, face)
matin = numpy.argmin(dist)
```

Step 3 – Detect the eyes from landmarks.

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 on the labeled iBUG 300-W dataset.

```
def find_eye(shape):
  (lstart, lend) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
  (rstart, rend) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
  lefteye=shape[lstart:lend]
  righteye=shape[rstart:rend]
```



Step 4 – Find EAR of both eyes.

```
leftEAR=eye_aspect_ratio(lefteye)
right_EAR=eye_aspect_ratio(righteye)
ear=(leftEAR+right_EAR)/2
```

Step 5 – Check EAR to check whether the person is drowsy

we have obtained the facial landmarks based on Dlib toolkit. As shown in Figure 6, for each eye, there are six points distributed around to locate the position of eye. The distribution of eyes landmarks has significant differences between open and closed state. In Eye Aspect Ratio was application to record the blink frequency. EAR can be computed according to the position of eyes landmarks by: EAR = kP2 − P6k + kP3 − P5k 2 kP1 − P4k where Pi, i = 1, 2, . . . , 6 is the coordinate of eyes landmarks.when the eyes of driver are open, the EAR is over 0.2. In contrast, the EAR is less than 0.2.

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

6. Result

Let's start our project and see the working of our project. To start the project, you need to open a command prompt, go to the directory where our main file "drowsiness detection.py" exists. Run the script with this command.

Output Screenshot:

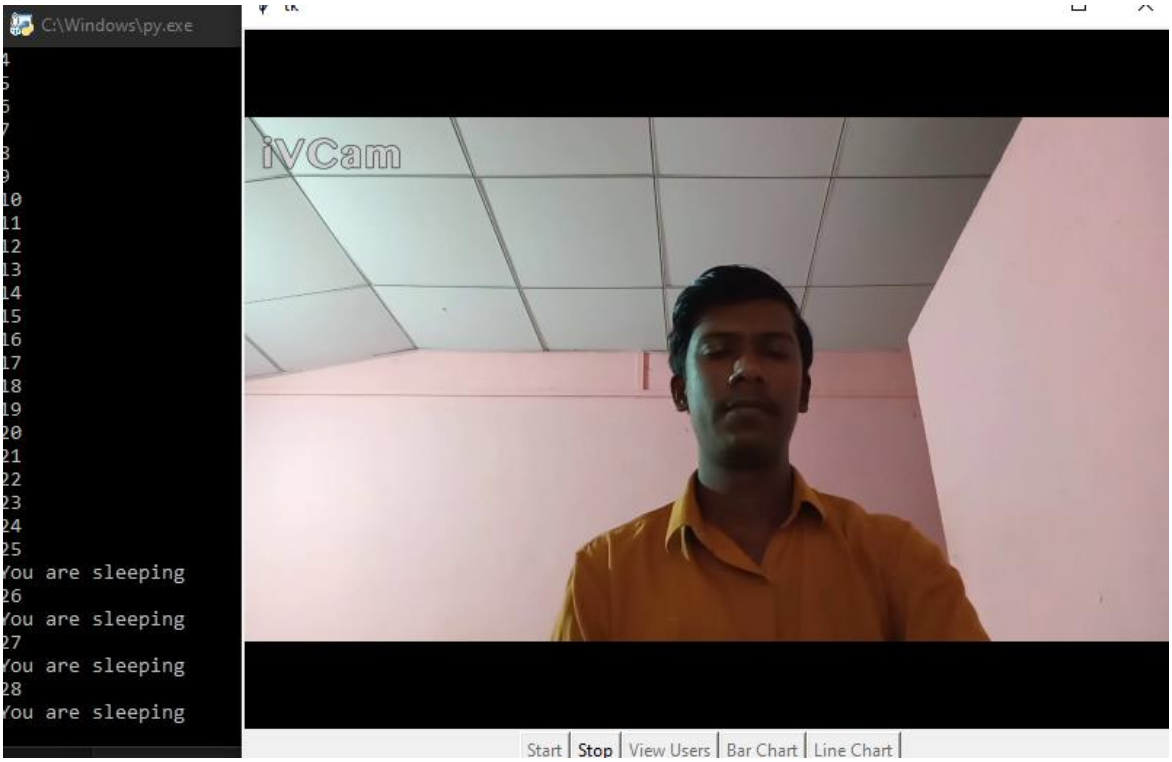


Fig 1:-Main View(find the user is drowsing and if drowsing then play the alarm)

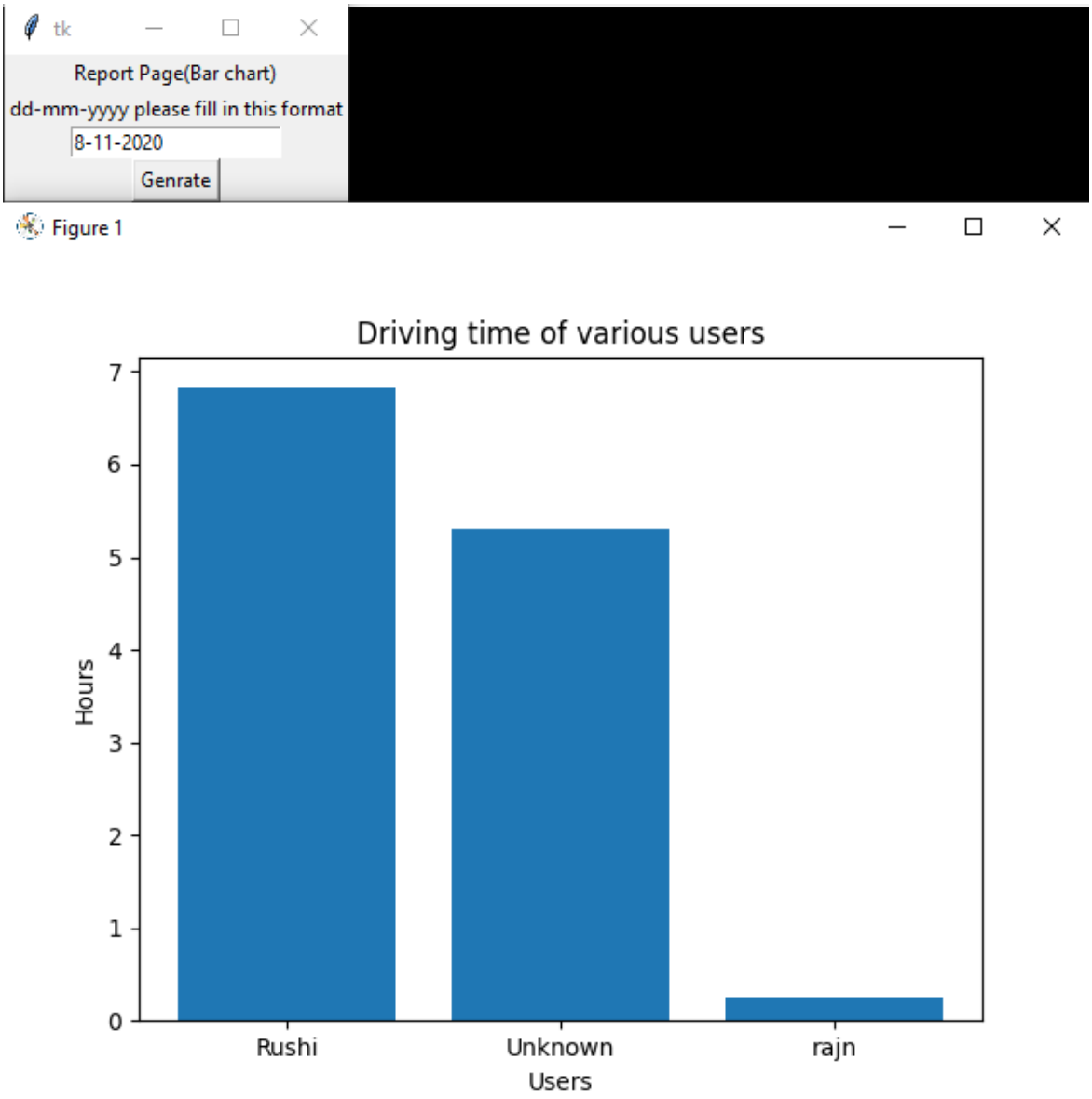


Fig:-2 Display a bar chart

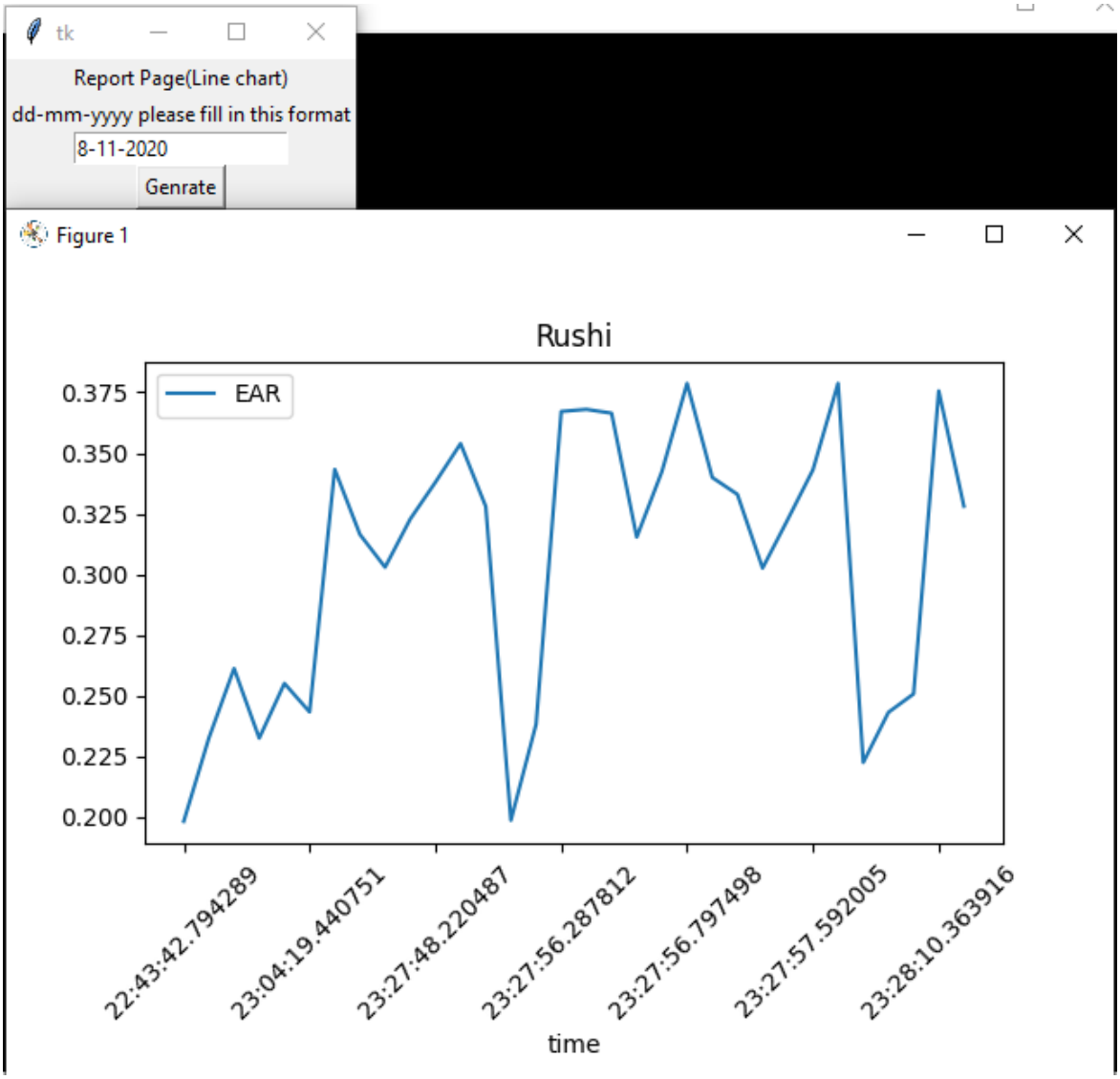


Fig 3 :- line chart for EAR for each user

7. CONCLUSION

In this Python project, we have built a drowsy driver alert system that you can implement in numerous ways. We used OpenCV to detect faces and eyes using a haar cascade classifier and then we used an EAR technique to predict the status.

8. References

https://www.pyimagesearch.com/2017/08/21/deep-learning-with-opencv/

https://arxiv.org/ftp/arxiv/papers/1710/1710.06854.pdf

https://data-flair.training/blogs/python-project-driver-drowsiness-detection-system/