Ain Shams University
Faculty of Computer & Information Sciences
Computer Science Department

Driver Drowsiness Detection

By:

Radwa Ayman Mohamed [CS]
Mahmoud Ashraf Farouk [CS]
Mohamed Abdelaty Mustafa [CS]
Mohamed Ali Mustafa [CS]
Mohamed Amr Farouk [CS]

Under Supervision of:

Dr. Salsabil Amin
Lecturer,
Basic Science Department,
Faculty of Computer and Information Sciences,
Ain Shams University.

TA. Dina Mohamed Sherif
Assistant Lecturer,
Basic Science Department,
Faculty of Computer and Information Sciences,
Ain Shams University.

July 2020

Acknowledgement

We would like to show our deep appreciation and gratitude for our supervisors, families, friends, and colleagues whose support encouraged us throughout this project.

Abstract

A lot of people fall victims to drowsy-related road accidents. Finding a solution to this problem is important and necessary as it threatens people's lives. Various automobile companies developed their own driver assistance systems to take actions in case the driver shows signs of fatigue, but not everyone can afford a high-end vehicle from these companies. Providing a system that is available to everyone to use despite their budget and circumstances is not impossible. A drowsy activity can be detected by mobile phones which are easy for a large amount of people to acquire. Driver Drowsiness Detection project aims to curb the danger of drowsy drivers on road and save as many lives as possible. Using mobile camera and Convolutional Neural Network models, the system observes the driver and classifies if they are drowsy or not. If drowsy state is detected for a certain amount of time, the system plays a loud disturbing alarm sound to alert the driver to stay awake or take some rest, so they do not put their lives and the lives of all people around them in danger.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	What the abbreviation stands for
AAA	American Automobile Association
ADAM	Adaptive Moment Estimation
ADAS	Advanced Driver-Assistance Systems
AI	Artificial Intelligence
API	Application Programming Interface
CNN	Convolutional Neural Network
CS	Computer Science
CSV	Comma-Separated Values
GUI	Graphical User Interface
IDE	Integrated Development Environment
MAE	Mean Absolute Error
MSE	Mean Squared Error
NHTSA	National Highway Traffic Safety Administration
NTHU	National Tsing Hua University
OpenCV	Open-Source Computer Vision
OS	Operating System
PC	Personal Computer
ReLU	Rectified Linear Unit
UI	User Interface
USB	Universal Serial Bus

1- Introduction

1.1 Motivation

High end vehicles have driver assistance systems to increase car and road safety. Meanwhile, economical vehicles do not have that kind of systems. This fact encourages developing a software that is available to everyone to use while driving to prevent accidents from occurring.
The project aims to harness computer and artificial intelligence to save lives, curb the danger of drowsy drivers on road and help them stay awake and alert.

1.2 Problem Definition

Car accidents are one of the major causes of injury and death. The AAA Foundation for Traffic Safety found in its survey of more than 3,500 drivers that drowsiness was identified in 8.8%–9.5% of all crashes examined [1]. The number of fatalities involving a drowsy driver was 697 or 1.9 percent of total fatalities in 2019 according to NHTSA [2].

Many factors lead to fatigue like sleep deprivation due to taking care of newborn baby, job stress, staying late with friends and medication. In a 24/7 society, with an emphasis on work, longer commutes, and exponential advancement of technology, many people do not get the sleep they need which leads to drive in drowsy state in a lot of cases. The effects of drowsy driving are the inability to focus, delayed reaction times, poor judgment, inability to judge distances and speeds, and, of course, falling asleep.

1.3 Objective

The main objective of the project is developing a driver drowsiness detection software to help driver stay awake. To achieve that, the system extracts the driver's face from the live video, tracks the eyes and detect whether they are aware or drowsy by detecting the eyes condition. If both eyes are closed for a certain amount of time, the system plays a loud sound to alert the driver to stay awake and prevent them from falling asleep and putting their lives and the others in danger.

1.4 Time Plan

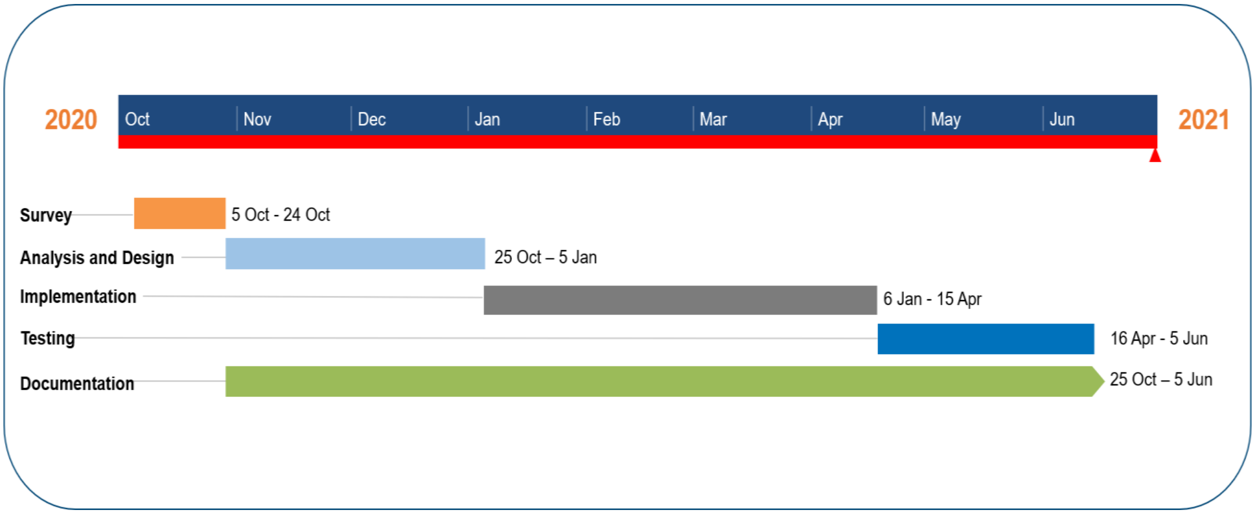

Figure 1. System Development Time Plan

System development includes 5 phases with each phase duration as shown in Figure 1.

- Phase 1 Survey [from October 5 to October 24, 2020]: searching and gathering information regarding the domain of the project.
- Phase 2 Analysis and Design [from October 25, 2020 to January 5, 2021]: designing the required diagrams and system architecture and preparing the datasets.
- Phase 3 Implementation [from January 6 to April 15, 2021]: building the software, coding, and training the CNN models.
- Phase 4 Testing [from April 16 to June 5, 2021]: testing system performance and making adjustments.
- Phase 5 Documentation [from October 25, 2020 to June 5, 2021]: writing and describing all the work done in details while developing the system.

1.5 Document Organization

The rest of the documentation is divided as follows.

Chapter 2, background, presents a detailed description of the field of the project, survey of the works done in the field, and description of existing similar systems.

Chapter 3, analysis and design, displays system architecture diagram along with detailed description of each module and the CNN models architecture in system overview. Use case diagram, class diagram, and sequence diagram are presented in system analysis and design.

Chapter 4, implementation and testing, describes in details all functions, techniques and algorithms implemented in the system. The chapter explains libraries used in project implementation, datasets and the preprocessing for each dataset, models building, training and performance, project classes, system integration, UI design and finally the testing.

Chapter 5, user manual, presents a complete user guide showing how to operate the project along with the required third-part program.

Finally, chapter 6, conclusion and future work, provides a complete summary of the whole project along with the results obtained, and ideas of what can be done in the future to improve the performance of the project.

2- Background

Drowsiness is a state of strong desire to sleep or feeling sleepy and tired. A variety of causes such as lifestyle factors like working long night shifts or raising small children, mental state as in depression, anxiety or stress, medical condition like diabetes, and sleeping disorder may lead to drowsiness.

Due to the current society structure, many people are obligated to participate in exhausting jobs, and some find themselves in a situation where they have to drive in fatigue state, which is risky and dangerous.

Figure 2. Effects of drowsy driving

With the rapid growing of technology, various techniques emerged to detect driver drowsiness. ADAS are sets of built-in technologies embedded in cars that assist drivers in driving using sensors and cameras. ADAS driver drowsiness detection technology obtains information such as driver's facial expression, vehicle position in lane, steering pattern, turn signal use and physiological parameters. The obtained information determines if the driver is drowsy or not. If drowsy activity is confirmed, the vehicle sounds off loud alert to warn the driver.

The problem of ADAS is the proprietary and limitation to high end automotive companies like BMW, Mercedes-Benz, and Tesla, which are not affordable to everyone.

Another approach to detect driver drowsiness is using deep learning and computer vision to develop models that can run on smart phone devices or small embedded systems, such models will benefit people who does not own high end vehicles. Computer vision is a field of AI that enables computers to understand and gather information from digital images and videos, then take actions based on that understanding. Deep learning is a subset of AI and type of machine learning methods that imitates human brain in data processing and decision making. Deep learning provides new tools to computer vision for detection and classification. Various works utilized these tools to detect driver drowsiness in different ways.

Rateb Jabbary et al. [3] developed Driver Drowsiness Detection Model Using CNN Techniques for Android Application. They trained the model on NTHU dataset [4] which contains 36 subjects from different ethnicities who have been recorded under day and night conditions. Their CNN model consists of 5 layers with Leaky ReLU as activation function for the first 4 layers and Softmax activation function for the last layer to get output class label probabilities, the overall accuracy achieved is 83.3%.

Quentin Massoz et al. [5] developed Multi-Timescale Drowsiness Characterization Based on a Video of a Driver's Face. They evaluated the model on 29 subjects via leave-one-subject-out cross-validation, develop a multi-timescale drowsiness characterization system composed of four binary drowsiness classifiers operating at four distinct timescales (5 s, 15 s, 30 s, and 60 s), and obtained accuracies of 70%, 85%, 89%, and 94% for the four classifiers operating at increasing timescales, respectively.

3- Analysis and Design

3.1 System Overview

3.1.1 System Architecture

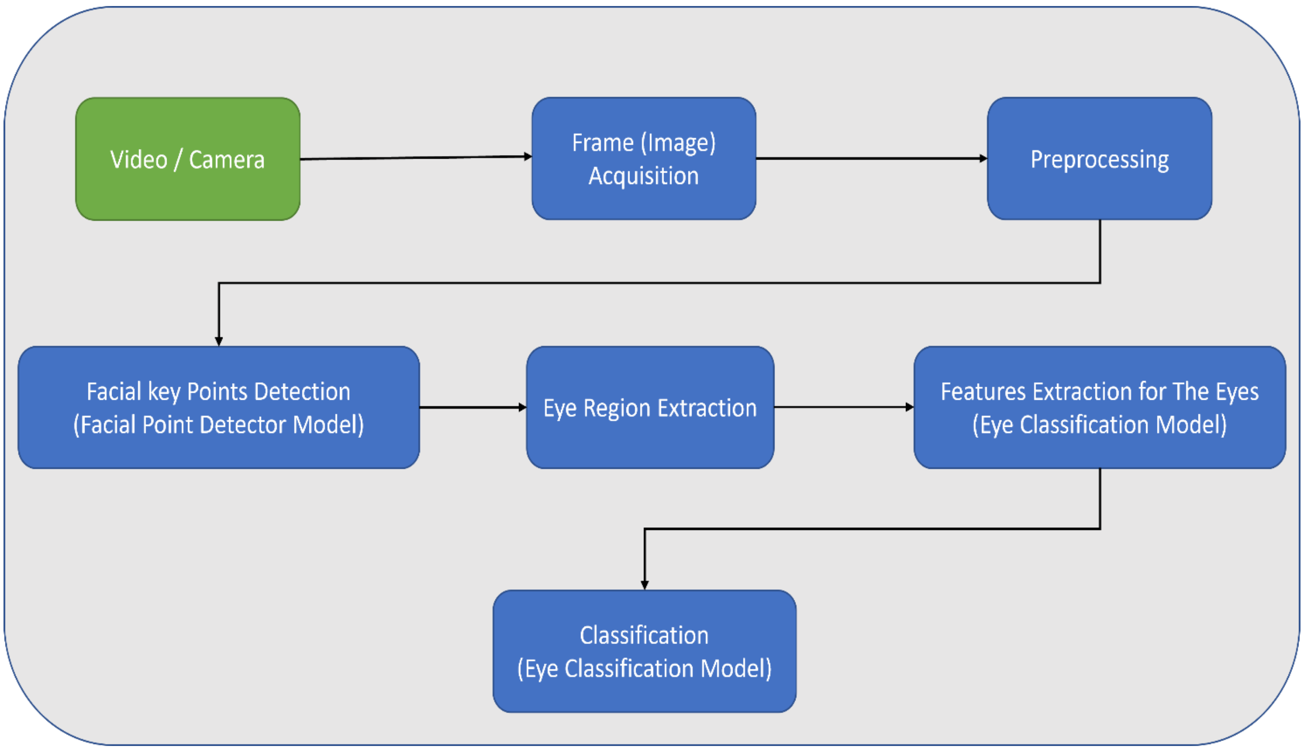

Figure 3. Driver Drowsiness Detection System Architecture

System architecture consists of 1 hardware entity and 6 main software modules, as shown in figure 3. The hardware entity is the camera which is used to record the driver and obtain real-time live video that will be fed to the system to analysis and classify. The main modules are divided as follows.

- Frame acquisition: In frame acquisition module, the video is processed frame by frame before applying preprocessing and passing the frame to CNN models.

- Preprocessing: A set of preprocessing techniques are applied to the frame before passing it to CNN models, such as increase brightness, increase contrast, and decrease noise.

- Facial key-points detection
  After frame acquisition and applying preprocessing, the driver face is extracted from the frame by OpenCV and is passed to facial key-points detection model, which detect the important features in the face like eyes, eyebrows, nose, and mouth.

  Facial key-points detection model is the first CNN model in the system and has been trained on facial key-points dataset [6]. The input to the model is 96x96 gray-scale image, the model outputs are 30 values [15 points (x, y) coordinates] indicating the important features in face. The model structure is constructed by 6 convolution and max pooling layers with convolution filters size = 3x3 and the number of filters in each layer are 16, 32, 64,128, 256, and 512, respectively. Following them are 5 dense and drop out layers with dropout rate = 0.1, and finally a dense output layer. All the layers have ReLU as an activation function except the last layer which has linear activation function. The model uses ADAM optimizer with learning rate 0.0005 and the loss function is mean squared error.

- Eye region extraction: Using the key points obtained from the facial points model, the eye region extraction module has the coordinates of the two eyes and can extract them from the frame.

- Features extraction: Since the eyes have been detected and obtained, they are fed to the eye model, which will extract their features and analysis them.
  Eye model is the second and last CNN model in the system and has been trained on eyes dataset. The input is 150x150x3 colored image, and the output is the classification class, closed or open. The model structure is constructed by 4 convolution and max pooling layers with convolution filters size = 3x3 and the number of filters in each layer are 32, 64,128, and 256, respectively. Following them are 3 dense and drop out layers with dropout rates 0.1, 0.2, and 0.1, respectively. The last layer is a dense layer with 2 neurons for the 2 classes. All the layers have ReLU as an activation function except the last layer which has Softmax activation function which gives probability of each class. The model uses ADAM optimizer with learning rate 0.00005 and the loss function is sparse categorical cross entropy.

- Classification: In the last module, the eye model classifies if the driver is drowsy or not based on the features of the eyes. If both eyes were found closed for a certain amount of time (5 frames), the loud alarm sound will be played, otherwise, nothing will happen.

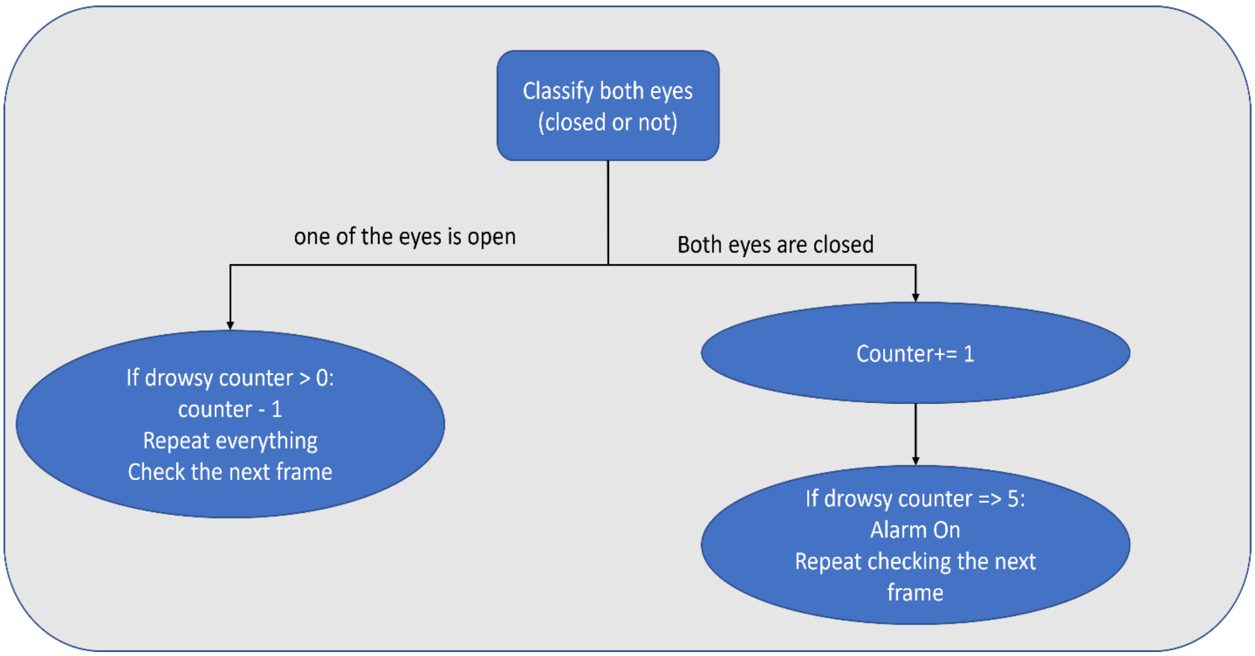

Figure 4. Classification Module

3.1.2 System Users

A. Intended Users:

The driver drowsiness detection system is built for end users who drive cars, buses, trucks, etc., who wish to remain safe on the road and avoid falling asleep while driving.

B. User Characteristics

To operate the system, the end user must have knowledge on how to use laptop, smart phone, and of course how to drive a car.

3.2 System Analysis & Design

3.2.1 Use Case Diagram

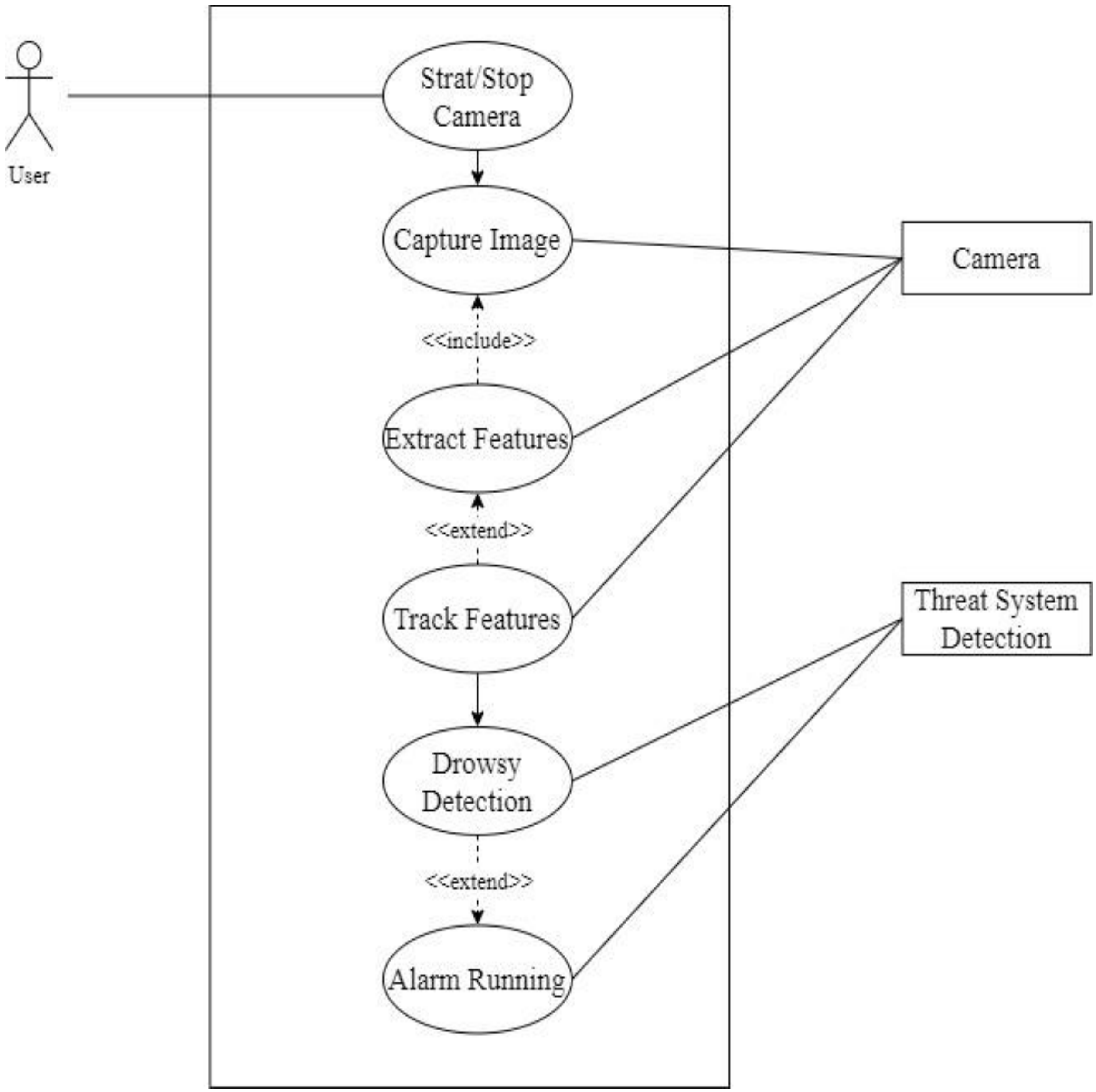

Figure 5. Use case diagram

3.2.2 Class Diagram

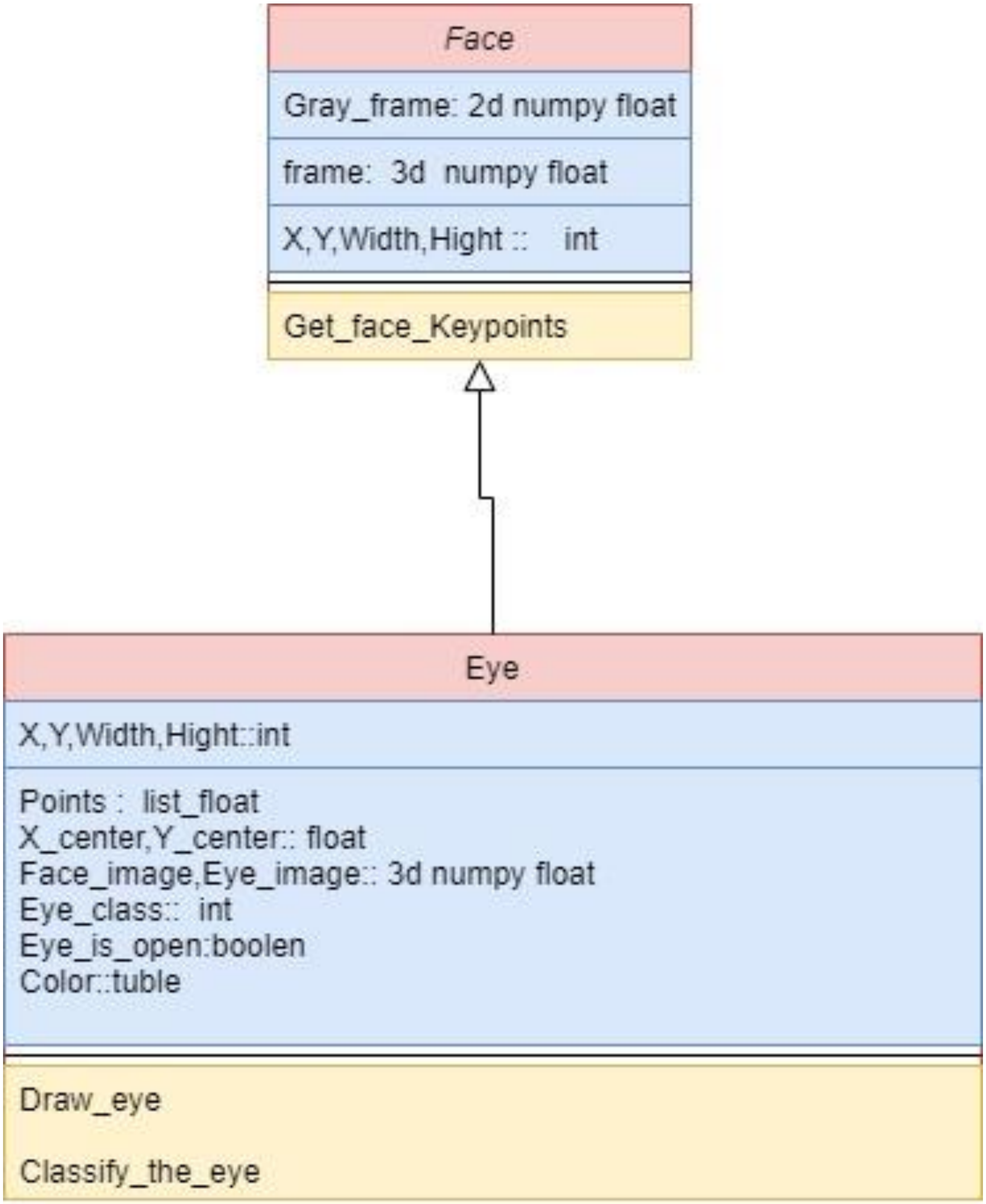

Figure 6. Class diagram

3.2.3 Sequence Diagram

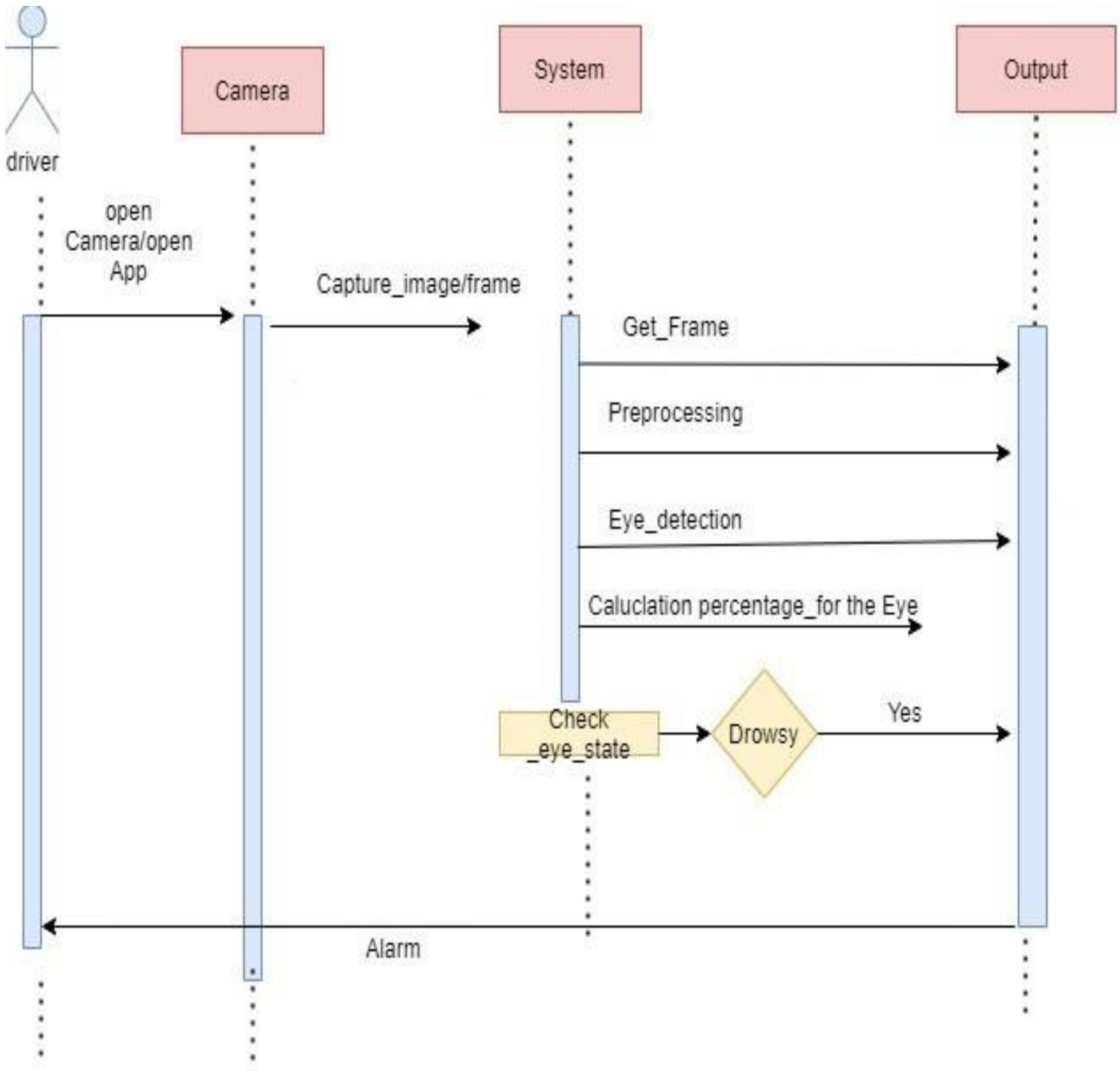

Figure 7. Sequence diagram

4- Implementation and Testing

4.1 IDE

PyCharm

4.2 Programming language

Python 3.6

4.3 Libraries and Packages

Python provides a variety of libraries for mathematics, computer vision, machine learning, deep learning and plotting diagrams. The following are the libraries used in project implementation.

- NumPy
  NumPy [7] is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

- OpenCV
  OpenCV [8] is a library of programming functions mainly aimed at real-time computer vision. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, and track moving objects, etc.

- TensorFlow
  TensorFlow [9] is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

- Matplotlib
  Matplotlib [10] is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter.

- OS
  The OS [11] module in Python provides functions for interacting with the operating system. OS comes under Python's standard utility modules.

- Time
  Time function from time library is used to calculate the models' training time.

- Pandas
  Pandas [12] is a software library written for the Python programming language for data manipulation and analysis. It offers data structures and operations for manipulating numerical tables and time series.

- Sklearn
  Model selection's train_test_split function from Sklearn splits arrays or matrices into random train and test subsets.

- Pygame
  Mixer from Pygame library is used to play the alarm sound in the system when drowsiness activity is detected.

- Tkinter
  Tkinter is a Python binding to the Tk GUI toolkit. It is the standard Python interface to the Tk GUI toolkit.

4.4 Datasets

4.4.1 Facial Key-Points Dataset

The dataset is a CSV file consists of 7049 gray-scale images. Each row contains the (x, y) coordinates for 15 key-points, and image data as row-ordered list of pixels in the last (31st) column. Each predicted key-point is specified by an (x, y) real-valued pair in the space of pixel indices. The 15 key-points represent the following elements of the face:
left_eye_center, right_eye_center, left_eye_inner_corner, left_eye_outer_corner, right_eye_inner_corner, right_eye_outer_corner, left_eyebrow_inner_end, left_eyebrow_outer_end, right_eyebrow_inner_end, right_eyebrow_outer_end, nose_tip, mouth_left_corner, mouth_right_corner, mouth_center_top_lip, mouth_center_bottom_lip.
Left and right refers to the point of view of the subject.
The input image consists of a list of pixels (ordered by row), as integers in (0,255) range. The images size is 96x96 pixels.

	A	B	C	D	E	F	G	H	I	J
1	left_eye_c	left_eye_c	right_eye_	right_eye_	left_eye_ii	left_eye_ii	left_eye_c	left_eye_c	right_eye_	right_eye_
2	66.03356	39.00227	30.22701	36.42168	59.58208	39.64742	73.13035	39.97	36.35657	37.3894
3	1 43 41 39 43 39 38 42 45 49 55 51 50 52 48 45 44 52 56 92 128 134 132 141 146 145 143 134 137 146 150 146									
4	64.33294	34.97008	29.94928	33.44871	58.85617	35.27435	70.72272	36.18717	36.03472	34.36153
5	65.05705	34.90964	30.90379	34.90964	59.412	36.32097	70.98442	36.32097	37.67811	36.32097
6	10 101 96 92 91 86 88 91 98 103 107 112 110 115 116 74 48 71 115 123 63 36 42 56 67 68 96 151 172 162 158									
7	65.22574	37.26177	32.0231	37.26177	60.00334	39.12718	72.31471	38.38097	37.61864	38.75411
8	2 161 166 168 176 186 161 99 57 21 0 1 1 1 1 1 1 1 1 1 1 1 2 1 0 79 78 77 65 41 19 5 0 1 1 1 1 1 0 9 24 36 77 136									
9	66.7253	39.62126	32.24481	38.04203	58.56589	39.62126	72.51593	39.88447	36.98238	39.09485
10	09 109 107 104 47 17 24 19 26 33 28 24 28 32 45 51 51 57 85 102 115 136 103 131 135 135 141 137 132 165 14									
11	69.68075	39.96875	29.18355	37.56336	62.8643	40.16927	76.89824	41.17189	36.40105	39.36763
12	64.13187	34.29004	29.57895	33.13804	57.79715	35.15404	69.02658	34.29004	34.76166	33.71404
13	67.46889	39.41345	29.35596	39.62172	59.55495	40.45477	75.59161	40.03824	37.47821	40.45477
14	5 93 88 94 139 112 129 119 102 121 126 111 110 94 67 132 201 213 221 226 223 223 227 229 232 231 226 212									
15	65.80288	34.7552	27.47584	36.1856	58.65216	37.32928	72.95296	35.89952	36.3424	37.0432
16	109 107 108 107 102 93 93 98 103 105 102 99 98 102 96 55 28 27 22 16 13 15 21 27 27 40 40 38 46 54 54 47 52									
17	64.12123	36.74031	29.46892	38.39015	58.62092	37.84062	71.272	37.29034	36.34462	39.49046

Figure 8. Facial key-points dataset sample

Facial key-points dataset preparing code.

dataGet() function.

Input: none.
Return: images, training and testing splits.
Description: dataGet() function reads the csv file, deals with the missing values in the dataset by forward fill approach using fillna() function, apply preprocessing on the images, extract the key-points (all columns except the last one) as targets, splits the data into training and testing (the data is split into 95% training and 0.05% testing).

```
 def dataGet():
    df = pd.read_csv('F:\senior year CS department\graduation project\second
demo\dataset/training.zip',
                     compression='zip',
                     header=0,
                     sep=',',
                     quotechar='"')


    df.fillna(method='ffill', inplace=True)

    X = preprocessing_Images(df.Image)

    targets = list(df.columns)
    targets.remove("Image")

    y = df[targets]
    # Split the data into 95 : 05 ratio
    x_train, x_test, y_train, y_test = model_selection.train_test_split(X, y,
test_size=0.05, random_state=42)

    return   df.Image , x_train, x_test, y_train, y_test
```

preprocessing_Images() function

Input: images in last column in dataset.
Return: images after preprocessing.
Description: preprocessing_Images() function reshapes the images into 96x96 array, normalizes them to be in range (0, 1), puts them in 96x96x1 shape to be fed to the CNN model.

```
def preprocessing_Images(data):
    data = data.apply(lambda x: np.fromstring(x, dtype=int, sep='
').reshape(96, 96))

    # Normalize the image
    data = data / 255

    # empty array to feed the model of shape(96,96,1)
    temp = np.empty((len(data), 96, 96, 1))

    # expanding dimensions to (96,96,1)
    for i in range(len(data)):
        temp[i,] = np.expand_dims(data[i], axis=2)
    return temp
```

4.4.2 Eyes Dataset

The eyes dataset contains colored images of eyes, divided into closed eyes and open eyes. Each image has a single eye. The images are different in size and lighting conditions. Lighting conditions are good, normal, and dark. The images have eyes with and without glasses. The dataset is divided into training and testing folders. Training folder contains 9964 images, 7874 open eye images and 2090 closed eye images. Testing folder contains 486 images, 277 open eye images and 209 closed eye images. The dataset is collected from various sources on the internet, in addition to pictures of acquaintances and friends.

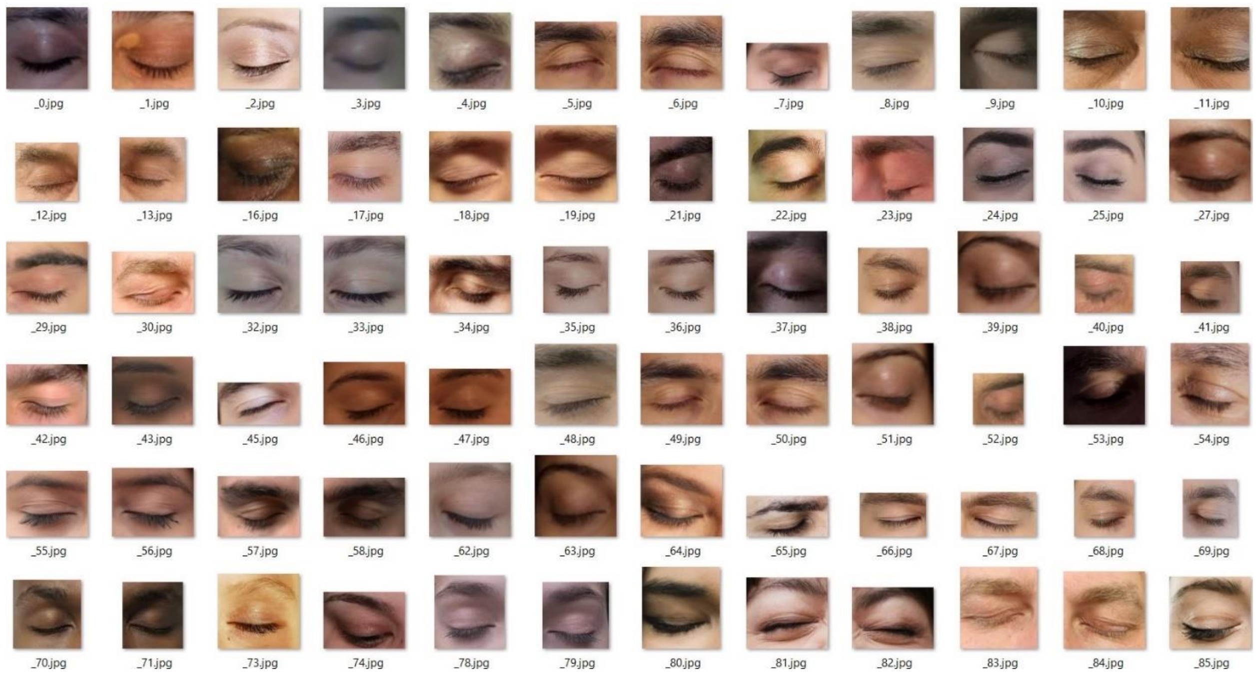

Figure 9. Eyes dataset: closed eyes sample

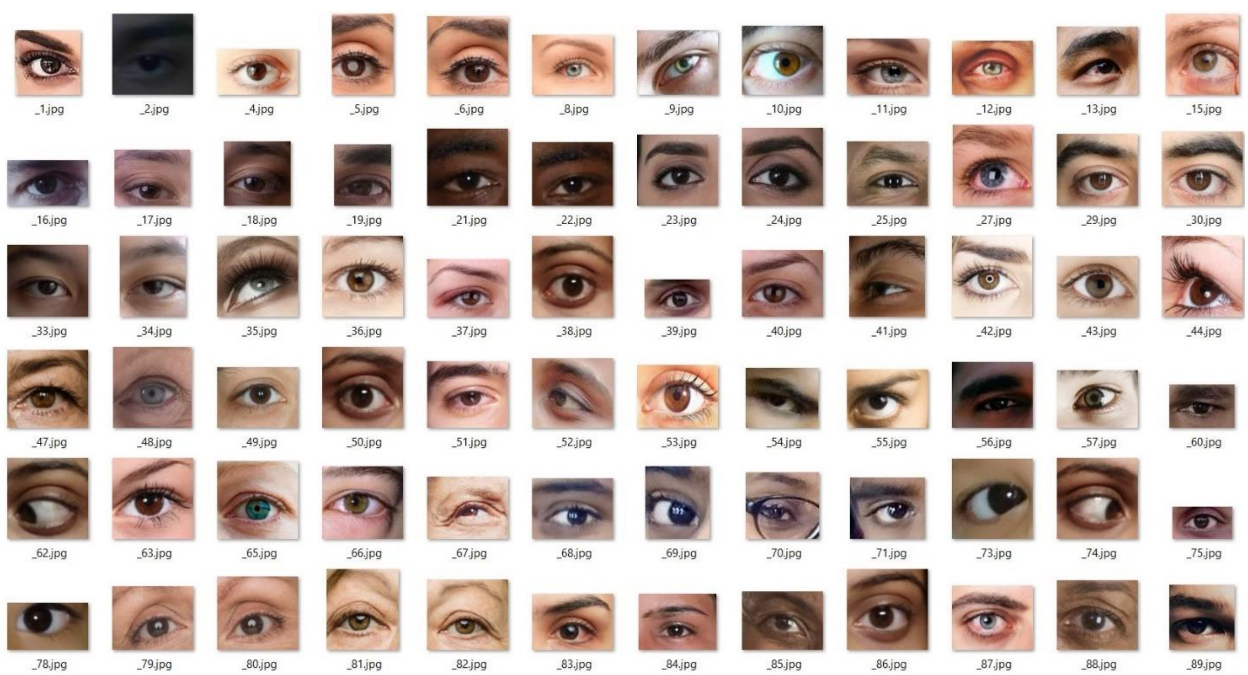

Figure 10. Eyes dataset: open eyes sample

Eyes dataset preparing code.

load_images_from_folder() function.

Input: the folder path containing images of people.
Return: list of eyes images and the total number of the extracted eyes.
Description: load_images_from_folder() function read each image in folder by OpenCV function imread(), convert image to gray-scale, detect eyes in the images by OpenCV's CascadeClassifier, if an eye or more is detected, it will be extracted from the image, resized to 150x150, and appended to eyes images list.

```
def load_images_from_folder(folder):
    images = []
    i=0
    no_of_eyes = 0
    for filename in os.listdir(folder):
        img = cv2.imread(os.path.join(folder,filename))

        if img is not None:
            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

            eyeCascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_lefteye_2splits.xml')
            eyes = eyeCascade.detectMultiScale(gray)
            img_with_detections = np.copy(img)
            i+=1
            print(i,"-"," has",len(eyes) , " eye in it")
            no_of_eyes += len(eyes)
            if (len(eyes) > 0):
                for (x, y, w, h) in eyes:
                    roi_color = img[y:y + h, x:x + w]
                    immg = cv2.resize(roi_color, (150, 150))
                    images.append(immg)
            else: continue
    return images , no_of_eyes
```

increaseContrast() function.

Input: colored image, clip limit.
Return: RGB image after increasing contrast.
Description: increaseContrast() function uses OpenCV function to increase contrast for a given RGB image.

```
def increaseContrast(img , clipLimit=4):
    # https://learnopencv.com/color-spaces-in-opencv-cpp-python/
    # -----Converting image to LAB Color model-------------------------------
----
    lab = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)

    # -----Splitting the LAB image to different channels---------------------
----
    l, a, b = cv2.split(lab)

    # -----Applying CLAHE to L-channel---------------------------------------
----
    clahe = cv2.createCLAHE(clipLimit, tileGridSize=(8, 8))
    cl = clahe.apply(l)

    # -----Merge the CLAHE enhanced L-channel with the a and b channel-------
----
    limg = cv2.merge((cl, a, b))

    # -----Converting image from LAB Color model to RGB model----------------
----
    contrasted = cv2.cvtColor(limg, cv2.COLOR_LAB2BGR)
    img = contrasted

    return img
```

augment_the_images() function.

Input: eyes images folder path.
Return: list of augmented images.
Description: for each file in the given folder, OpenCV reads the image, increaseContrast() function is applied on the image, image is resized to 150x150, finally the image is appended to images list.

```
def augment_the_images(folder):

    images = []
    i=0
    for filename in os.listdir(folder):
        img = cv2.imread(os.path.join(folder,filename))


        contrusted_img = increaseContrast(img,1.5)
        contrusted_img = cv2.resize(contrusted_img, (150, 150))

        images.append(contrusted_img)

        i+=1
        print(i)
    return images
```

4.5 Models Implementation

4.5.1 Facial Key-Points Model

Loading training and testing data which will feed the model by dataGet() function.

```
hh,x_train, x_test, y_train, y_test = dataGet()
```

Model structure is constructed by TensorFlow's sequential function, the input is 96x96x1 gray-scale images, and the model's outputs are 15 key-points features coordinates extracted from the images.

```
model = Sequential([
    tf.keras.layers.Conv2D(16, 3, padding='same',
activation='relu',input_shape=(96,96,1)),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(32, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(64, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(128, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(256, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(512, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.1),

    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.1),

    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.1),

    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.1),

    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.1),

    tf.keras.layers.Dense(30)
])
```

Compiling the model with Adam optimizer and MSE loss function.

```
model.compile(optimizer=Adam(lr=0.001),
                loss='mean_squared_error',
                metrics='mae')
```

Training the model on the training data (80% for training and 20% for validation), number of epochs is 15, batch size is 64, and calculating the time taken in training.

```
start_time = time.time()

history = model.fit(x_train, y_train, epochs=15,
batch_size=64,validation_split=0.2 )



end_time = time.time()

time_taken = end_time - start_time
```

Plotting training and validation mean absolute error against epochs.

```
mae = history.history['mae']
val_mae = history.history['val_mae']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(mae))

plt.plot(epochs, mae, 'r', label='Training mean absolute error')
plt.plot(epochs, val_mae, 'b', label='Validation mean absolute error')
plt.title('Training and validation accuracy and the taken time in training
is: ' + str(int(time_taken)) + "seconds for " +str(epochs) + " epochs" )
plt.legend(loc=0)
plt.show()
```

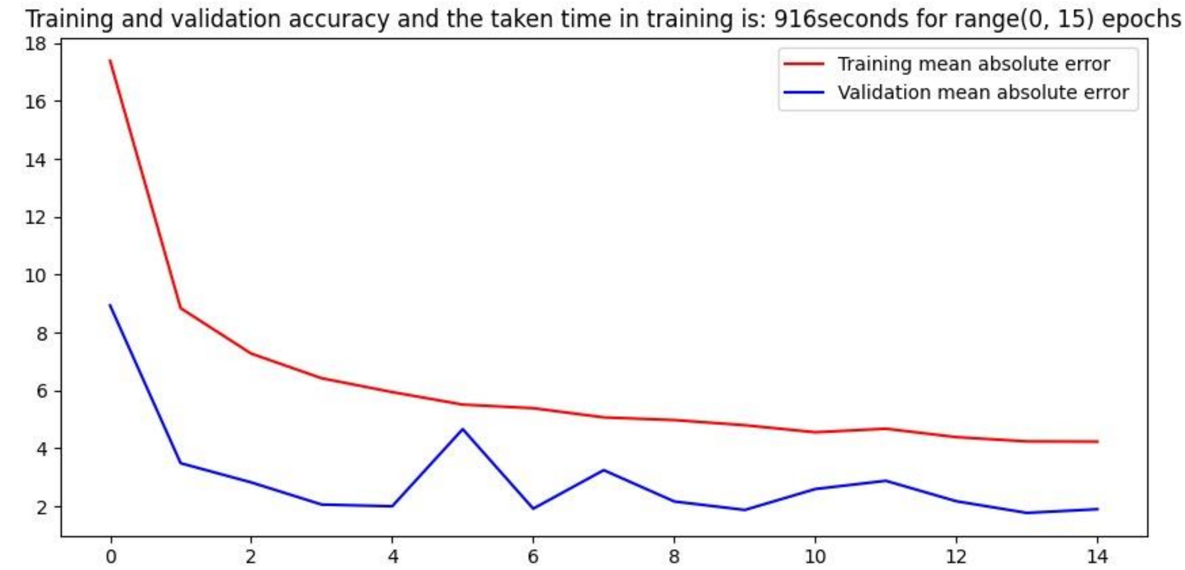

Figure 11. Facial key-points model MAE plot

Plotting training and validation mean squared error against epochs.

```
plt.plot(epochs, loss, 'r', label='Training mean square error')
plt.plot(epochs, val_loss, 'b', label='Validation mean square error')
plt.title('Training and validation accuracy and the taken time in training
is: ' + str(int(time_taken)) + "seconds for " +str(epochs) + " epochs" )
plt.legend(loc=0)
plt.show()
```

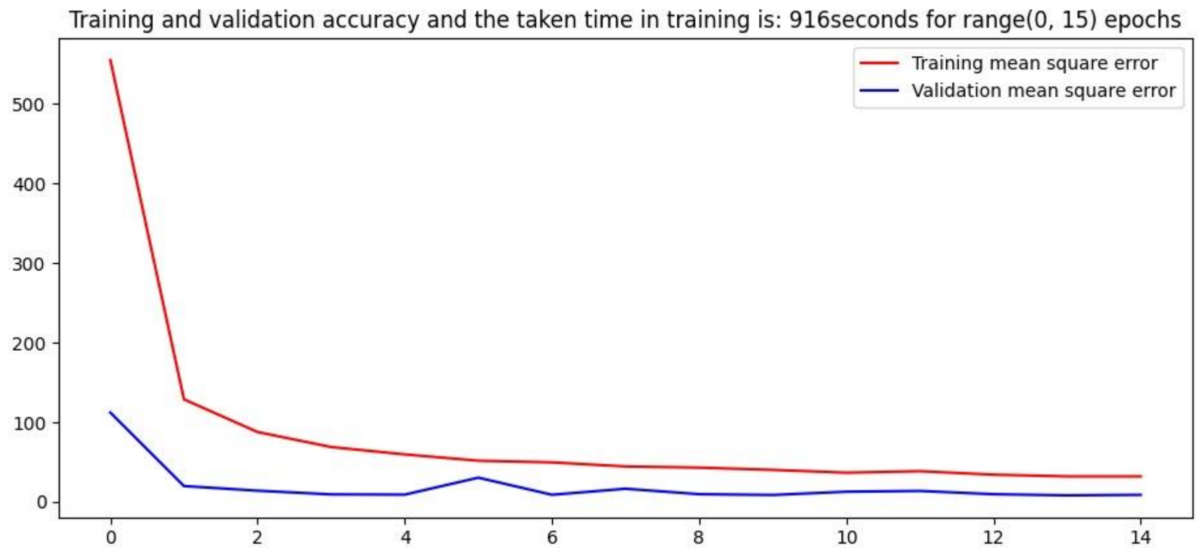

Figure 12. Facial key-points model MSE plot

Saving the model so it can be used again without retraining.

```
export_path_sm = "./models/{}.h5".format("face_key_point_999")
print(export_path_sm)
tf.saved_model.save(model, export_path_sm)
```

Model evaluation on testing data.

```
model.evaluate(x_test,y_test)
print(model.evaluate(x_test,y_test))
```

```
12/12 [==============================] - 1s 65ms/step - loss: 9.2502 - mae: 1.9637
12/12 [==============================] - 1s 65ms/step - loss: 9.2502 - mae: 1.9637
[9.250161170959473, 1.9637163877487183]
```

Figure 13. Facial key-points model evaluation output

The model obtained MSE = 9.25 and MAE = 1.96 on the testing data.

4.5.2 Eye Model

Through trial and error, various eye models architectures have been trained, model -13.py had the best performance, so that model is the one saved and used in the system.

Loading training and validation data from eyes dataset.

```
base_dir = './data/'

train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'test')

# Directory with our training cat/dog pictures
train_open_dir = os.path.join(train_dir, 'Open')
train_closed_dir = os.path.join(train_dir, 'Closed')

# Directory with our validation cat/dog pictures
validation_open_dir = os.path.join(validation_dir, 'Open')
validation_close_dir = os.path.join(validation_dir, 'Closed')

train_open_fnames = os.listdir(train_open_dir)
train_closed_fnames = os.listdir(train_closed_dir)
```

Generating training and validation batches of tensor image data by TensorFlow's ImageDataGenerator() function. The batch size is 32, and class mode is binary because the model classifies 2 classes (closed, open).

```
train_datagen = ImageDataGenerator()
test_datagen  = ImageDataGenerator()

# --------------------
# Flow training images in batches of 20 using train_datagen generator
# --------------------
train_generator = train_datagen.flow_from_directory(train_dir,
                                                    batch_size=32,
                                                    class_mode='binary',
                                                    target_size=(150, 150))

# --------------------
# Flow validation images in batches of 20 using test_datagen generator
# --------------------
validation_generator =  test_datagen.flow_from_directory(validation_dir,
                                                         batch_size=32,
                                                         class_mode =
'binary',
                                                         target_size = (150,
150))
```

Model structure is constructed by TensorFlow's sequential function, the input is 150x150x3 colored eye images, and the model's outputs the probabilities for the 2 classes by Softmax activation function.

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, 3, padding='same',
activation='relu',input_shape=(150,150,3)),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(64, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(128, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(256, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),


    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.1),

    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.2),

    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.1),

    tf.keras.layers.Dense(2 , activation='softmax')
])
```

Model compiling using Adam optimizer and sparse_categorical_crossentropy loss function.

```
model.compile(optimizer=Adam(lr=0.00005),
                loss=sparse_categorical_crossentropy,
                metrics=['acc'])
```

Model training on training and validation data, with number of epochs 50. After training the model is saved and the time taken for training is calculated.

```
start_time = time.time()

history = model.fit_generator(
    train_generator,
    validation_data = validation_generator,
    epochs=50,
    verbose=1
)

end_time = time.time()

export_path_sm = "./models/{}.h5".format("driver_drowsiness_softmax_13")
print(export_path_sm)
tf.saved_model.save(model, export_path_sm)

time_taken = end_time - start_time
```

Plotting the model's training and validation accuracy and loss against epochs.

```
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.plot(epochs, loss, 'r', label='Training accuracy')
plt.plot(epochs, val_loss, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy and the taken time in training
is: ' + str(int(time_taken)) + "seconds for " +epochs + " epochs" )
plt.legend(loc=0)
plt.figure()

plt.show()
```

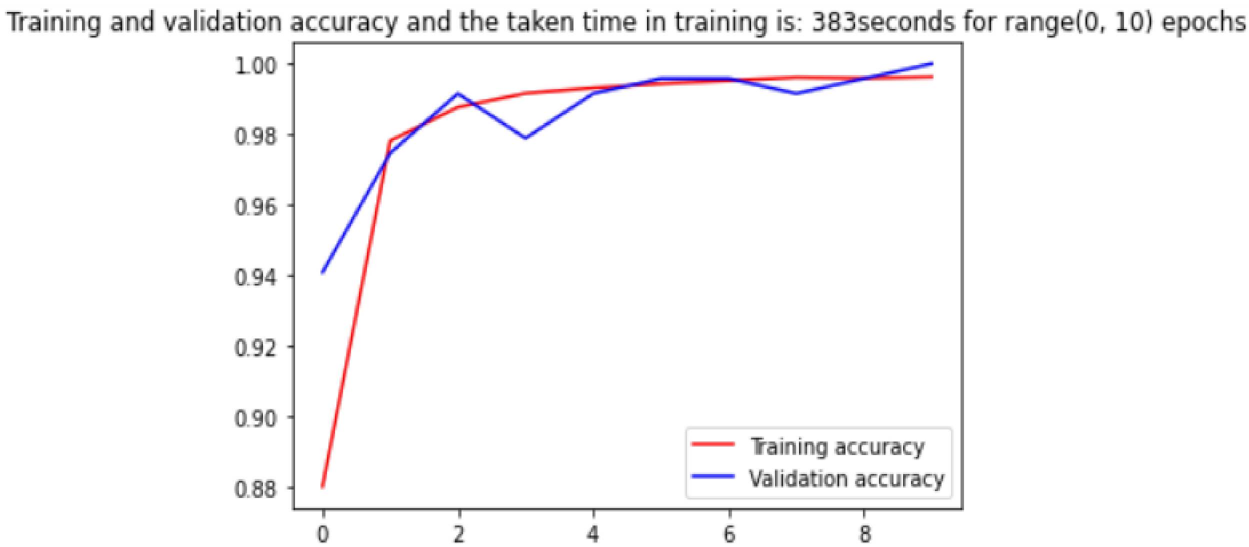

Figure 14. Eye model accuracy plot

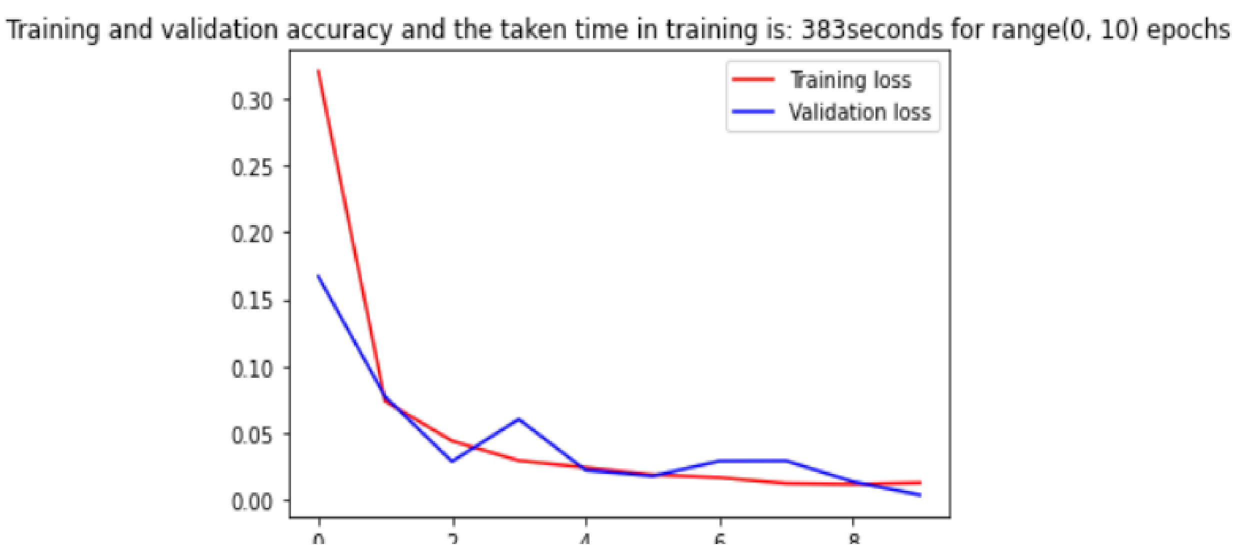

Figure 15. Eye model loss plot

4.6 Project Classes

First step to operate the system is loading the trained and saved models.

Load facial key-points model.

```
export_path_sm = "./models/facial_points.h5"
keypoints_model = tf.keras.models.load_model(export_path_sm)
```

Load eye model.

```
export_path_sm = "./models/driver_drowsiness_softmax_13.h5"
eye_model = tf.keras.models.load_model(export_path_sm)
```

draw_rectangle_with_text() function.

Input: image to draw rectangle on, x and y are the center coordinates, w is the width, h is the height, text color and the text to be written above the rectangle.
Return: image with rectangle drawn on it.
Description: draw_rectangle_with_text() function uses OpenCV's rectangle function to draw rectangle on a given image with the specified center, width, height for the rectangle. Then the desired text is printed on the image by OpenCV's putText function.

```
def draw_rectangle_with_text(img, x ,y , w , h ,color , text):
    cv2.rectangle(img, (x - w, y - h), (x + w, y + h), color, 2)
    # write over the rectangle the percentage of being open
    cv2.putText(img, text, (x - w - 6, y - h - 6), cv2.FONT_HERSHEY_SIMPLEX,
0.4, color, 2)
    return img
```

getpoints() function.

Input: 96x96 gray-scale image.
Return: 15 facial key-points coordinates.
Description: getpoints() function normalizes a given image then feeds it to the facial key-points models to predict the important features in the face in the image.

```
def getpoints(img):
    temp = np.expand_dims(img, axis=2)
    temp=temp/(255.0)
    temp = np.expand_dims(temp, axis=0)
    points = keypoints_model.predict(temp)
    return points
```

4.6.1 FaceAndPoints Class

FaceAndPoints constructor.

Input: frame, x and y coordinates, width, height, and gray face image
Description: the constructor initializes the attributes that will be used in the class.

```
def __init__(self, frame, x, y, w, h, gray_face):
    self.frame = frame
    self.x = x
    self.y = y
    self.width = w
    self.height = h
    self.gray_face = gray_face
```

get_face_keypoints() function.

Input: none.
Return: face image and key-points in the face array.
Description: get_face_keypoints() function resizes the input gray face image to 96x96, call getpoints() function to extract key-points in face, extracts the face from the input frame, and resizes the face to 400x400 to be more visible.

```
def get_face_keypoints(self):
    face_for_keypoints = cv2.resize(self.gray_face, (96, 96))
    keypoints = getpoints(face_for_keypoints)

    face = self.frame[self.y: self.y + self.height, self.x: self.x +
self.width]
    face = cv2.resize(face, (400, 400))

    return face, keypoints
```

4.6.2 Eye Class

Eye constructor.

Input: x and y indexes, width, height, points, and face image.
Description: the constructor initializes the attributes that will be used in the class.

```
def __init__(self, x_index, y_index, w, h, points, face_image):
    self.x_index = x_index
    self.y_index = y_index
    self.points = points
    self.x_center = int(int(int(self.points[0][self.x_index]))*400/96)
    self.y_center = int(int(int(self.points[0][self.y_index]))*400/96)
    self.width = w
    self.height = h
    self.eye_is_open = False
    self.face_image = face_image

    # extract the eye from the image
    self.eye_img = self.face_image[self.y_center - h - 10: self.y_center + h,
self.x_center - w-5: self.x_center + w+5]
    self.eye_class = 0
    self.color = (0,0,0)
```

classify_the_eye() function.

Input: None.
Return: None.
Description: classify_the_eye() function resizes the eye image to 150x150 then transform it to np array to be in form 150x150x3 to be fed to eye model, and finally classifies the eye and assign the classification probability to eye_class attribute.

```
def classify_the_eye(self):
    # here we resize the  eye image, because the model accept 150 x 150
images only
    resized_eye_img = cv2.resize(self.eye_img, (150, 150))
    # here we prepare the eye image for the model to be np.array(150 x 150 x
3)
    resized_eye_for_predection = np.array([resized_eye_img])
    self.eye_class = eye_model.predict(resized_eye_for_predection)[0][1]
```

draw_eye() function.

Input: None.
Return: face image with green rectangle around the eye with if it is open and red if it is closed.
Description: draw_eye() function checks the eye's class, if it is open, the rectangle color will be green, otherwise it will be red. Then the draw_rectangle_with_text() function will be called twice to draw rectangle around the eye, and draw small dot in the center of the eye.

```
def draw_eye(self):
    if self.eye_class > 0.5:
        # if self.eye_class <= 0.65 : self.eye_class = (self.eye_class +
0.35)
        print('this eye: ', self.eye_class)
        self.color = (0, 255, 0)  # green color for open eyes
        self.eye_is_open = True  # set the boolean to true


    else:
        print('eye: ', self.eye_class)
        self.color = (0, 0, 255)
        self.eye_is_open = False

    text_on_the_eye = 'this eye is ' + str(int(float(self.eye_class) * 100))
+ "% open"
    self.face_image = draw_rectangle_with_text(self.face_image,
self.x_center, self.y_center,
                                               self.width, self.height,
self.color, text=text_on_the_eye)



    self.face_image = draw_rectangle_with_text(self.face_image,
self.x_center, self.y_center,
                                               1, 1, self.color, text="")


    return self.face_image
```

4.7 Integration

In usethemodel.py file, the models, classes, and functions are integrated to form the Driver Drowsiness Detection system. The functions are explained as follows.

get_larges_face() function.

Input: gray-scale frame.
Return: x, y coordinates, width, height, and gray-scale face image.
Description: get_larges_face() function uses OpenCV's CascadeClassifier function to detect faces in the input frame. If faces are detected, the function searches for the largest face among them, crops it from the frame, and returns its center coordinates, its width and height, and the cropped image itself. If no faces are detected, the function returns the input frame.

```
def get_larges_face(gray_frame):
    facecascade = cv2.CascadeClassifier(cv2.data.haarcascades +
"haarcascade_frontalface_default.xml")
    faces = facecascade.detectMultiScale(gray_frame)
    # to make sure that there is a face in the image
    if len(faces) > 0:
        # to get the biggest face in the image
        max_w = 0
        max_h = 0
        index = -1
        for i in range(len(faces)):
            (x, y, w, h) = faces[i]
            # we compare between the area of the faces if there is more than
one face
            if ((w * h) > (max_w * max_h)):
                max_w = w
                max_h = h
                index = i
        # here we get the width and the hight of the face to crop the face
from the image
        (x, y, width, height) = faces[index]

        gray_face = gray_frame[y: y + height, x: x + width]
        return x, y, width, height, gray_face
    # if the number of faces is zero "no face exist return all zeros"
    else:
        return 0, 0, 0, 0, gray_frame
```

eyesanddrowsyness() function.

Input: left eye object, right eye object, face image, and drowsy counter.
Return: face image with rectangle around the face and drowsy counter.
Description: eyesanddrowsyness() function has 3 cases. The first case is when both eyes are open, the drowsy counter is decremented by one, and green rectangle is drawn around the face. The second case is when one of the eyes is open, the drowsy counter is decremented by one, and blue rectangle is drawn around the face. The third case is when both eyes are closed, the drowsy counter is incremented by one, and red rectangle is drawn around the face.

```
def eyesanddrowsyness(left_eye, right_eye, img, drowsycounter):

    if left_eye.eye_is_open and right_eye.eye_is_open:
        color = (0, 255, 0) # green
        thickness = int((drowsycounter+2)/2)  # decrease the thickness, and
the "+2" is just to make sure that it will one or more
        drowsycounter = (drowsycounter>0)*(drowsycounter - 1) # decrement the
drowsycounter by one

        # draw and write on the img , the image is 400 x 400 , so i used 15,
15, to leave some space from up and left
        # and made the width and height 370 so that will leave the same space
from right and down "15+370=385"
        img = draw_rectangle_with_text(img, 15, 15, 370, 370, color,
text='both are open', thickness=thickness)


    elif left_eye.eye_is_open or right_eye.eye_is_open:
        color = (255, 0,0) #blue ... "yes, in opencv it is (B,G,R) not
(R,G,B)" so this is blue not red
        thickness = int((drowsycounter+2))
        drowsycounter =  (drowsycounter>0)*(drowsycounter - 1) # decrement
the drowsycounter by one

        # draw and write on the img , the image is 400 x 400 , so i used 15,
15, to leave some space from up and left
        # and made the width and height 370 so that will leave the same space
from right



and down "15+370=385"
        img = draw_rectangle_with_text(img, 15, 15, 370, 370, color,
text="one is open", thickness=thickness)

    else:
        color = (0, 0, 255) # red ... "yes, in opencv it is (B,G,R) not
(R,G,B)" so this is red not blue
        thickness = int((drowsycounter + 2) / 1.5) # increase the
```

```
thickness,by 2
        drowsycounter = (drowsycounter +1*(drowsycounter<=20)) # increase the
thickness,by 1 + make sure it will not exceed 20

        # draw and write on the img , the image is 400 x 400 , so i used 15,
15, to leave some space from up and left
        # and made the width and height 370 so that will leave the same space
from right and down "15+370=385"
        img = draw_rectangle_with_text(img, 15, 15, 370, 370, color,
text="both are closed", thickness=thickness)

    print('drowsy counter: ',drowsycounter)
    return img, drowsycounter
```

project_func() function.

Input: current frame in the video, drowsy counter.
Return: final frame, drowsy counter.
Description: project_func() function can be considered as the main function of the project. In converts the input frame to gray-scale image, calls get_larges_face() function and checks if width and height are equal 0, then there no face in the current frame, and returns. Otherwise, face object is created, the driver's face and the facial key-points are extracted. Left eye object is created, the eye is classified, a rectangle is drawn around the eye, then the same steps are repeated for the right eye. Finally, eyesanddrowsyness() function is called to draw rectangle around the face according to both eyes classification.

```
def project_func(frame, drowsycounter):
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # get the larges (nearest) face in the frame if there are more than one
    # if  one return it
    x, y, w, h, gray_face = get_larges_face(gray_frame)

    # check if width and height are zere means no faces so retern the fram
dirctly with no change
    if w == 0 and h == 0 :
        return frame, drowsycounter

    # face object
    face_OB = PC.FaceAndPoints(frame, x, y, w, h, gray_face)

    # get the extracted face and the facial keypoints
    the_driver_face, the_driver_facial_keypoints =
face_OB.get_face_keypoints()

    # Eye object for the left eye
    # PC.Eye(X_Point, Y_point, Width, Height, points, driver face image)
    left_eye_OB = PC.Eye(0, 1, 50, 50, the_driver_facial_keypoints,
the_driver_face)

    # predect if the eye is clossed of open
    left_eye_OB.classify_the_eye()

    # draw rectangle around the left eye and put the percentage on it
    frame_with_lefteye_draw = left_eye_OB.draw_eye()

    # Eye object for the left eye
    # PC.Eye(X_Point, Y_point, Width, Height, points, driver face image)
    right_eye_OB = PC.Eye(2, 3, 50, 50, the_driver_facial_keypoints,
frame_with_lefteye_draw)

    # predect if the eye is clossed of open
    right_eye_OB.classify_the_eye()
```

```
    # draw rectangle around the right eye and put the percentage on it
    frame_with_eyes = right_eye_OB.draw_eye()

    # draw rectangle around the face with different colors and different
thikness accourding to the drowsycounter
    # and the closed and open eyes and increase or decrease the drowsycounter
according to the case
    # then return the img with the drawed rectangle and the drowsy detection
    final_frame, drowsycounter= eyesanddrowsyness(left_eye_OB, right_eye_OB,
frame_with_eyes,drowsycounter)

    return final_frame, drowsycounter
```

adjust_brighness() function.

Input: current frame, gamma value.
Return: lookup table.
Description: the function builds a lookup table mapping the pixel values [0, 255] to their adjusted gamma values, then applies gamma correction using the lookup table to adjust a given frame brightness.

```
def adjust_brighness(image, gamma=1.0):
    # build a lookup table mapping the pixel values [0, 255] to
    # their adjusted gamma values
    invGamma = 1.0 / gamma
    table = np.array([((i / 255.0) ** invGamma) * 255 for i in np.arange(0,
256)]).astype("uint8")
    # apply gamma correction using the lookup table
    return cv2.LUT(image, table)
```

increaseContrast() function.

Input: current frame.
Return: frame after increasing contrast.
Description: the function uses OpenCV's functions to increase contrast in a given frame.

```
def increaseContrast(img):

    # https://learnopencv.com/color-spaces-in-opencv-cpp-python/
    # -----Converting image to LAB Color model--------------------------------
----
    lab = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)

    # -----Splitting the LAB image to different channels---------------------
----
    l, a, b = cv2.split(lab)

    # -----Applying CLAHE to L-channel---------------------------------------
----
    clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8, 8))
    cl = clahe.apply(l)

    # -----Merge the CLAHE enhanced L-channel with the a and b channel-------
----
    limg = cv2.merge((cl, a, b))

    # -----Converting image from LAB Color model to RGB model----------------
----
    contrasted = cv2.cvtColor(limg, cv2.COLOR_LAB2BGR)
    img = contrasted

    # r, g, b = cv2.split(img)
    # clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    # cl1 = clahe.apply(r)
    # clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    # cl2 = clahe.apply(g)
    # clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    # cl3 = clahe.apply(b)
    #
    # limg = cv2.merge((cl1, cl2, cl3))

    return img
```

decreaseNoice() function.

Input: current frame.
Return: frame after decreasing the noise.
Description: the function uses OpenCV's functions to decrease noise in a given frame by applying median blurring.

```
def decreaseNoice(img):

    # ----- apply median blurring -------------------------------------------
    median_blured = cv2.medianBlur(img, 3)

    # img = gausian_blured
    img = median_blured


    return img
```

preprocess_the_frame() function.

Input: current frame.
Return: frame after applying preprocessing.
Description: preprocess_the_frame() function calls adjust_brighness(), increaseContrast(), and decreaseNoice functions to apply these preprocessing techniques on a given frame.

```
def preprocess_the_frame(img):
    img = adjust_brighness(img,7)
    img = increaseContrast(img)
    img = decreaseNoice(img)
    return img
```

4.8 UI Design

The main function of the gui.py file initializes the Tkinter root window, which is the main application window in the project, calls App class to run system functions, and runs mainloop() function.

```python
def main(args):
    root = tk.Tk()
    app = App(root, "OpenCV Image Viewer")
    root.mainloop()

if __name__ == '__main__':
    sys.exit(main(sys.argv))
```

The App class functions are explained as follows.

App Constructor

Input: parent window, window title
Description: the app constructor initializes the GUI attributes.

```python
def __init__(self, parent, title):
    tk.Frame.__init__(self, parent)
    self.is_running = False
    self.thread = None
    self.queue = Queue()
    self.photo = ImageTk.PhotoImage(Image.new("RGB", (800, 600), "white"))
    parent.wm_withdraw()
    parent.wm_title(title)
    self.create_ui()
    self.grid(sticky=tk.NSEW)
    self.bind('<<MessageGenerated>>', self.on_next_frame)
    parent.wm_protocol("WM_DELETE_WINDOW", self.on_destroy)
    parent.grid_rowconfigure(0, weight = 1)
    parent.grid_columnconfigure(0, weight = 1)
    parent.wm_deiconify()
```

create_ui() function.

Input: none.
Return: none.
Description: the function creates 3 components in the main window. First component is the frame which will display the captured video. Second component is the stop button to close the camera. Third component is the start button which will open the camera.

```
def create_ui(self):
    self.button_frame = ttk.Frame(self)

    self.stop_button = ttk.Button(self.button_frame, text="Stop",
command=self.stop)
    self.stop_button.pack(side=tk.RIGHT)

    self.start_button = ttk.Button(self.button_frame, text="Start",
command=self.start)
    self.start_button.pack(side=tk.RIGHT)

    self.view = ttk.Label(self, image=self.photo)
    self.view.pack(side=tk.TOP, fill=tk.BOTH, expand=True)
    self.button_frame.pack(side=tk.BOTTOM, fill=tk.X, expand=True)
```

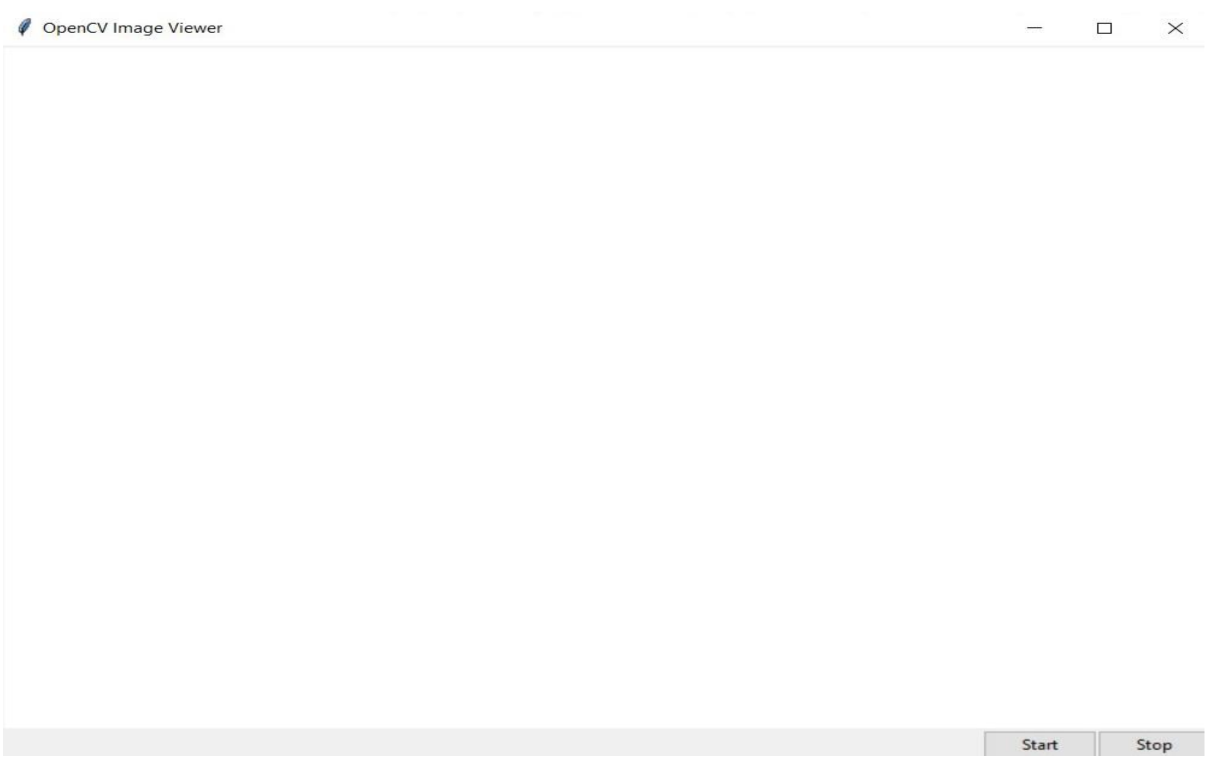

Figure 16. User interface window

videoLoop() function.

Description: The camera is opened by VideoCapture() function with takes input 0 for laptop camera, and input 1 for mobile or external camera. The drowsy counter is initialized by 0. The mixer is initialized to play the alarm sound in case of drowsiness. The while loop is for processing each frame as long as the camera is running. Current frame is extracted from the live video, flipped horizontally, and passed to preprocess_the_frame() function, then project_func(). The frame is resized to 600x600 to be visible and clear in the run. Finally, the drowsy counter is checked, if it is greater than or equal to 5, the alarm sound will play.

```
def videoLoop(self, mirror=False):
    No=1
    cap = cv2.VideoCapture(No)
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, 800)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 600)
    drowsycounter = 0  # initialization of the drowsyness counting
    mixer.init()
    sound = mixer.Sound('alarm.mp3')  # this is the alarm sound
    while self.is_running:
        ret, to_draw = cap.read()

        # make it flipped like mirror
        to_draw = cv2.flip(to_draw, 1)

        # conition = 0 for high light
        # conition = 1 for good light
        # conition = 2 for bad light
        to_draw = pjbody.preprocess_the_frame(to_draw, condition=0)

        # get the latest frame with all the drawing on it after applying the
models
        to_draw, drowsycounter = pjbody.project_func(to_draw, drowsycounter)

        # resize to a fixed size
        to_draw = cv2.resize(to_draw, (600, 600))


        # if his both eyes was closed for 7 frames so alarm play
        if drowsycounter >= 5:
            sound.play()

        if drowsycounter < 5:
            sound.stop()



        if mirror is True:
            to_draw = to_draw[:,::-1]
        image = cv2.cvtColor(to_draw, cv2.COLOR_BGR2RGB)
        self.queue.put(image)
        self.event_generate('<<MessageGenerated>>')
```

4.9 Testing

4.9.1 Testing Levels.

Table 1. Testing approaches comparison

Criteria	Black box testing	White box testing
Definition	Black box testing is a software testing method in which the internal structure, design, and implementation of the module being tested is not known to the tester	White box testing is a software testing method in which the internal structure, design, and implementation of the module being tested is known to the tester
Applicable levels	Applicable to higher levels of testing	Applicable to lower levels of testing
Responsibility	Software testers	Software developers
Programming knowledge	Not required	Required
Implementation knowledge	Not required	Required
Test cases basis	Requirement Specifications	Detailed Design

Unit testing

Each module in the system is tested separately to ensure that it is working correctly. Unit testing is based on white box testing approach. The face model and eye model are tested independently.

Integration testing

Different software modules are combined and tested as a group to make sure that integrated system is ready for system testing. Integration testing is based on black box testing approach. The face model and eye model are integrated and tested together.

System testing

System testing is performed on a complete, integrated system. It allows checking system's performance as per the requirements. It tests the overall interaction of components.

Acceptance testing

The last testing level, Acceptance testing (or User Acceptance Testing), is conducted to determine whether the system is ready for release. During this testing phase, the end user tests the system to find out whether the application meets their business needs. Once this process is completed and the software is accepted, the program will be delivered to production.

4.9.2 Testing Results.

Facial key-points model.

Table 2. Facial key points model results

Case	Number of trials	Expected output	Correct output	Percentage
Person in good light	30	30	25	83.5%
Person in poor light	30	30	20	66.6%
Driver in car with good light	30	30	24	80%
Driver in car with poor light	30	30	20	66.6%

The overall accuracy is 74.16%

Table 3. Eye model results

Case	Number of trials	Expected output	Correct output	Percentage
Person in good light	30	30	24	80%
Person in poor light	30	30	19	63.5%
Driver in car with good light	30	30	23	76.6%
Driver in car with poor light	30	30	18	60%

The overall accuracy is 70%

5- User Manual

The following steps describe in details how to operate the Driver Drowsiness Detection project.

Step 1: To run the project using laptop camera, skip directly to step 3. To run the project using mobile camera, follow step 2 instructions.

Step 2: DroidCam installation.
DroidCam application shall be installed on the mobile phone, and DroidCam PC client shall be installed on the PC.

Step 2.1: Install DroidCam from Google Play or App Store.

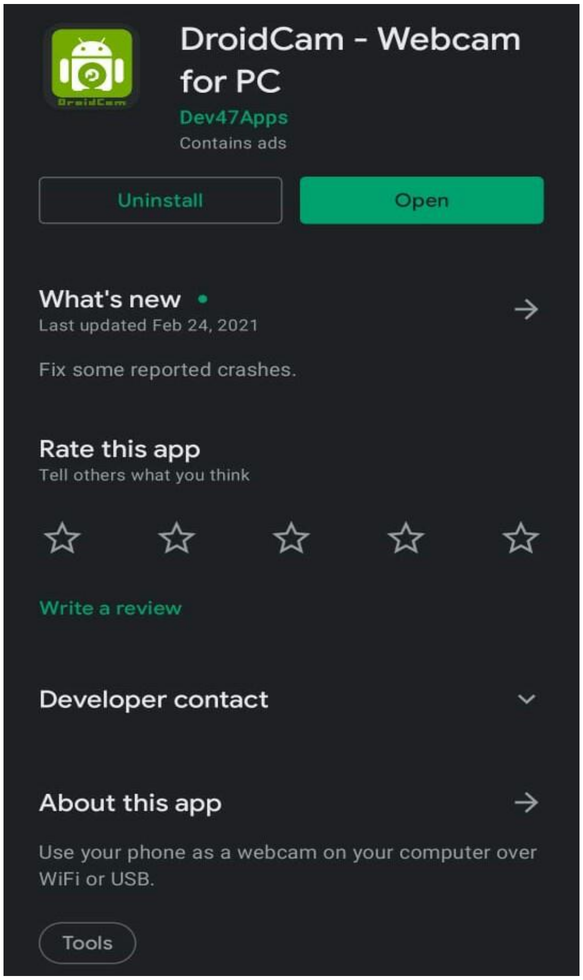

Figure 17. DroidCam application on google play

Step 2.2: Install DroidCam PC client from Dev47Apps website by the following link. https://www.dev47apps.com/ . follow the installation steps.

Step 2.3: Open DroidCam on mobile and PC and connect them through USB. Refer to the following link for connection steps.
https://www.dev47apps.com/droidcam/connect/

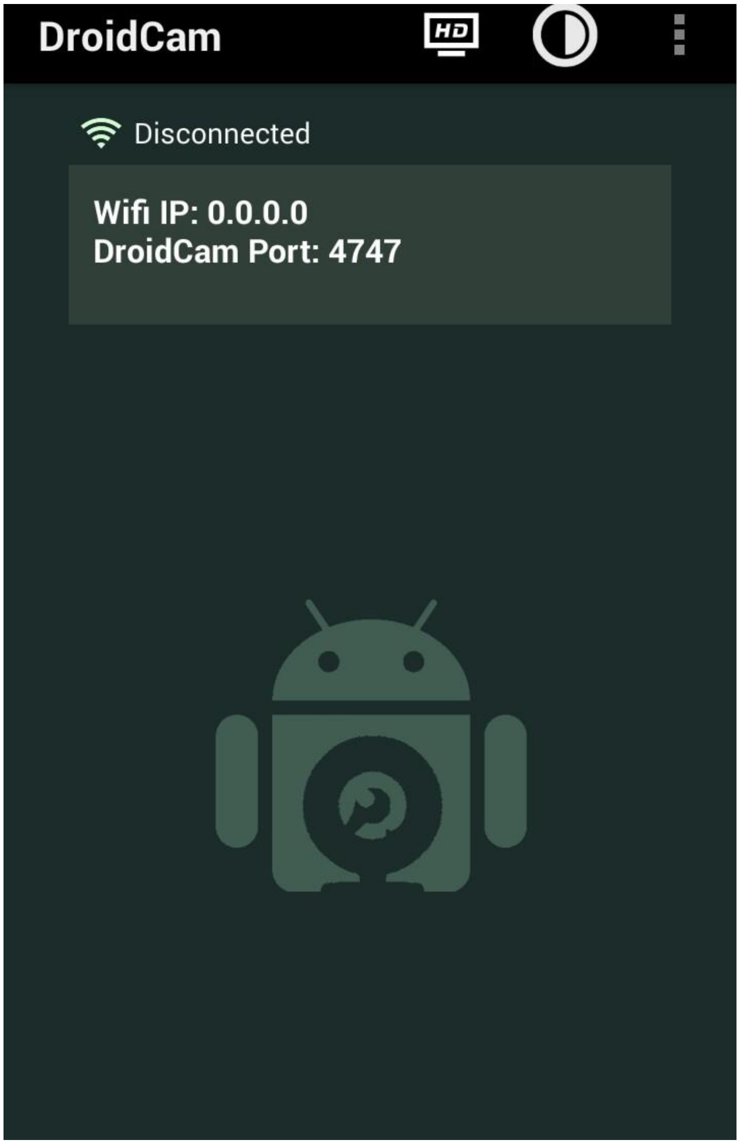

Figure 18. DroidCam application on mobile

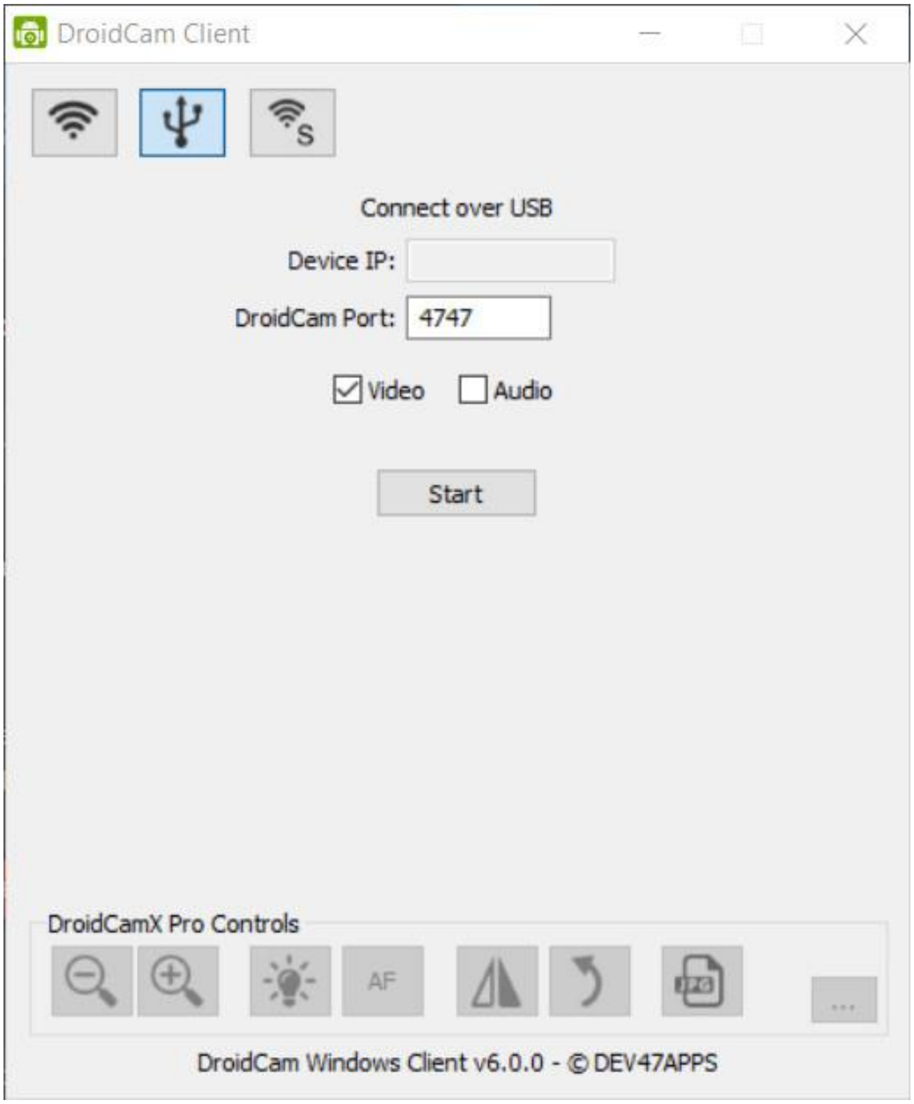

Figure 19. DroidCam client on PC

Step 2.4: Click on Start button in the DroidCam client.

Step 2.5: Make sure that the face is positioned right in the captured video. If the face is rotated, rotate the mobile accordingly so the face position is right.

Step 3: Run Driver Drowsiness Detection project from PyCharm IDE. Run the gui.py file. Press the green arrow button.

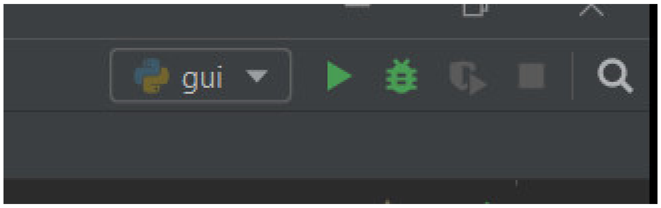

Figure 20. How to run the project

Step 4: To open the camera, click on Start button in the project window.

Figure 21. Runtime when camera is open example

Step 5: To close the camera, click on Stop button in the project window.

6- Conclusion and Future Work

6.1 Conclusion

Driver Drowsiness Detection project aims to save as many lives as possible on the road. The system uses laptop or mobile camera to capture a live video of the driver, processes each frame in the video to extract the drivers' face and its key features. From the extracted features, the system detects eyes, classifies them, and acts based on the classification. If the eyes are classified as closed for 5 sequential frames, the system plays a very loud sound to alert the drive, so they can stay awake or take a break from driving.

There are 2 deep leaning CNN models implemented in the project using TensorFlow, the first one is for extracting the face important features, and the second model detects the eyes from the previously extracted features and classifies them.

The project's GUI is simple and user friendly, to simplify the project operation.

The system was tested on 30 different persons in various conditions and obtained correct results of overall accuracy 74.16% for the face model and overall accuracy 70% for the eye model.

6.2 Future Work

Some improvements can be done to upgrade the performance of the project.

- Infrared camera may be integrated into the system to improve the detection and classification in poor lighting condition.
- Adding a functionality to deal with the case when the driver's face moves and disappears from front of the camera.

To make the system easier for daily basis usage, a mobile application may be developed to make the system available for installation and running on mobile devices.

7- Appendices

Main Code Segments

1. Facial key points model.
   1.1. Facial key-points dataset preprocessing.

data.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# import visualkeras


import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential

from sklearn import model_selection
from sklearn import metrics


def percent_missing(df):
    data = pd.DataFrame(df)
    df_cols = list(pd.DataFrame(data))
    dict_x = {}
    for i in range(0, len(df_cols)):
        dict_x.update({df_cols[i]:
round(data[df_cols[i]].isnull().mean() * 100, 2)})

    return dict_x


def preprocessing_Images(data):
    data = data.apply(lambda x: np.fromstring(x,
dtype=int, sep=' ').reshape(96, 96))
```

```
    # Normalize the image
    data = data / 255

    # empty array to feed the model of shape(96,96,1)
    temp = np.empty((len(data), 96, 96, 1))

    # expanding dimensions to (96,96,1)
    for i in range(len(data)):
        temp[i,] = np.expand_dims(data[i], axis=2)
    return temp


def images_vis(x, y, loc, y_pred, point_show=True):
    plt.imshow(x[loc], cmap='gray')
    if point_show == True:
        for j in range(0, 28, 2):
            plt.plot(y.iloc[loc][j], y.iloc[loc][j +
1], 'bo', label='Actual values')
            plt.plot(y_pred[loc][j], y_pred[loc][j +
1], 'rx', label='Predicted values')


def images_vis_train(x, y, loc, point_show=True):
    plt.imshow(x[loc], cmap='gray')
    if point_show == True:
        for j in range(0, 28, 2):
            plt.plot(y.iloc[loc][j], y.iloc[loc][j +
1], 'bo', label='Actual values')

def dataGet():
    df = pd.read_csv('F:\senior year CS
department\graduation project\second
demo\dataset/training.zip',
                     compression='zip',
                     header=0,
                     sep=',',
                     quotechar='"')

    all_features = df.columns
```

```
    missing = percent_missing(df)
    df_miss = sorted(missing.items(), key=lambda x:
x[1], reverse=True)
    df.fillna(method='ffill', inplace=True)

    X = preprocessing_Images(df.Image)

    targets = list(df.columns)
    targets.remove("Image")

    y = df[targets]
    # Split the data into 95 : 05 ratio
    x_train, x_test, y_train, y_test =
model_selection.train_test_split(X, y, test_size=0.05,
random_state=42)

    return   df.Image , x_train, x_test, y_train,
y_test
```

1.2. Model training.

model.py

```
from data import *
import tensorflow as tf
import os
import zipfile
from os import path, getcwd, chdir
import PIL
import PIL.Image
from keras.models import load_model
import h5py
import pathlib
import cv2
import time
import os
import zipfile

import matplotlib.image as mpimg
```

```
import matplotlib.pyplot as plt



hh,x_train, x_test, y_train, y_test = dataGet()



DESIRED_ACCURACY = 0.85


class myCallback(tf.keras.callbacks.Callback):
    # Your Code
    def on_epoch_end(self, epoch, logs={}):
        if (logs.get("acc") > DESIRED_ACCURACY):
            self.stop_traning = True


callbacks = myCallback()

model = Sequential([
    tf.keras.layers.Conv2D(16, 3, padding='same',
activation='relu',input_shape=(96,96,1)),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(32, 3, padding='same',
activation='relu'),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(64, 3, padding='same',
activation='relu'),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(128, 3, padding='same',
activation='relu'),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(256, 3, padding='same',
activation='relu'),
    tf.keras.layers.MaxPooling2D(),
```

```
    tf.keras.layers.Conv2D(512, 3, padding='same',
activation='relu'),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.1),

    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.1),

    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.1),

    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.1),

    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.1),

    tf.keras.layers.Dense(30)
])

from tensorflow.keras.optimizers import Adam

model.compile(optimizer=Adam(lr=0.001),
                loss='mean_squared_error',
                metrics='mae')



#
# from tensorflow.keras.callbacks import
ModelCheckpoint

# chechpoint_path = 'F:\\senior year CS
department\\graduation project\\second demo\\'
# chechpoint =
ModelCheckpoint(filepath=chechpoint_path,
#                               frequency = 'epoch',
#                               save_weights_only=True,
```

```
#                              verbose=1
#                              )


start_time = time.time()

history = model.fit(x_train, y_train, epochs=15,
batch_size=64,validation_split=0.2 )



end_time = time.time()

time_taken = end_time - start_time

mae = history.history['mae']
val_mae = history.history['val_mae']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(mae))

plt.plot(epochs, mae, 'r', label='Training mean
absolute error')
plt.plot(epochs, val_mae, 'b', label='Validation mean
absolute error')
plt.title('Training and validation accuracy and the
taken time in training is: ' + str(int(time_taken)) +
"seconds for " +str(epochs) + " epochs" )
plt.legend(loc=0)
plt.show()

plt.plot(epochs, loss, 'r', label='Training mean square
error')
plt.plot(epochs, val_loss, 'b', label='Validation mean
square error')
plt.title('Training and validation accuracy and the
taken time in training is: ' + str(int(time_taken)) +
"seconds for " +str(epochs) + " epochs" )
plt.legend(loc=0)
```

```
plt.show()


export_path_sm =
"./models/{}.h5".format("face_key_point_999")
print(export_path_sm)
tf.saved_model.save(model, export_path_sm)

model.evaluate(x_test,y_test)
print(model.evaluate(x_test,y_test))
```

2. Eye classification model.
    2.1. Eyes dataset preprocessing.

preprocessing.py

```
import os
import zipfile
from os import path, getcwd, chdir
import PIL
import PIL.Image
import h5py
import pathlib
import cv2
import numpy as np
import os
import zipfile



def load_images_from_folder(folder):
    images = []
    i=0
    no_of_eyes = 0
    for filename in os.listdir(folder):
        img = cv2.imread(os.path.join(folder,filename))

        if img is not None:
            gray = cv2.cvtColor(img,
cv2.COLOR_BGR2GRAY)
```

```
            eyeCascade =
cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_lefteye_2splits.xml')
            eyes = eyeCascade.detectMultiScale(gray)
            img_with_detections = np.copy(img)
            i+=1
            print(i,"-"," has",len(eyes) , " eye in
it")
            no_of_eyes += len(eyes)
            if (len(eyes) > 0):
                for (x, y, w, h) in eyes:
                    roi_color = img[y:y + h, x:x + w]
                    immg = cv2.resize(roi_color, (150,
150))
                    images.append(immg)
            else: continue
    return images , no_of_eyes

folder= "F://senior year CS department//graduation part
2//data//train//Open/"

# images , no_of_eyes= load_images_from_folder(folder)

def adjust_brighness(image, gamma=1.0):
    # build a lookup table mapping the pixel values [0,
255] to
    # their adjusted gamma values
    invGamma = 1.0 / gamma
    table = np.array([((i / 255.0) ** invGamma) * 255
for i in np.arange(0, 256)]).astype("uint8")
    # apply gamma correction using the lookup table
    return cv2.LUT(image, table)



def increaseContrast(img , clipLimit=4):
    # https://learnopencv.com/color-spaces-in-opencv-
cpp-python/
    # -----Converting image to LAB Color model---------
```

```
---------------------------
    lab = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)

    # -----Splitting the LAB image to different
channels-------------------------
    l, a, b = cv2.split(lab)

    # -----Applying CLAHE to L-channel-----------------
---------------------------
    clahe = cv2.createCLAHE(clipLimit, tileGridSize=(8,
8))
    cl = clahe.apply(l)

    # -----Merge the CLAHE enhanced L-channel with the
a and b channel-----------
    limg = cv2.merge((cl, a, b))

    # -----Converting image from LAB Color model to RGB
model--------------------
    contrasted = cv2.cvtColor(limg, cv2.COLOR_LAB2BGR)
    img = contrasted

    return img


def augment_the_images(folder, value=60):

    images = []
    i=0
    for filename in os.listdir(folder):
        img = cv2.imread(os.path.join(folder,filename))
        #
        # brighted_img = adjust_brighness(img,2)
        # brighted_img = cv2.resize(brighted_img, (150,
150))
        #
        # images.append(brighted_img)
        #
        # darked_img = adjust_brighness(img,0.5)
        # darked_img = cv2.resize(darked_img, (150,
```

```
150))
        #
        # images.append(darked_img)
        #
        # fliped_img = cv2.flip(brighted_img, 1)
        # images.append(fliped_img)

        contrusted_img = increaseContrast(img,1.5)
        contrusted_img = cv2.resize(contrusted_img,
(150, 150))

        images.append(contrusted_img)
        #
        # fliped_img = cv2.flip(contrusted_img, 1)
        # images.append(fliped_img)
        i+=1
        print(i)
    return images

images =augment_the_images(folder)

print(len(images))

# print("no_of_eyes: ",no_of_eyes)

path = "F://senior year CS department//graduation part
2//data"

for i in range(len(images)):
    print(str(i)+"th pic  done ")
    cv2.imwrite(str(path)+str(i)+"  .jpg" , images[i])
    cv2.waitKey(0)
```

2.2. Model training – first trial

model - 5.py

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf
import zipfile
from os import path, getcwd, chdir
import PIL
import PIL.Image
from keras.models import load_model
import h5py
import pathlib
import cv2
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import os
import zipfile
import time
base_dir = './data/'

train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'test')

# Directory with our training pictures
train_open_dir = os.path.join(train_dir, 'Open')
train_closed_dir = os.path.join(train_dir, 'Closed')

# Directory with our validation pictures
validation_open_dir = os.path.join(validation_dir,
'Open')
validation_close_dir = os.path.join(validation_dir,
'Closed')

train_open_fnames = os.listdir(train_open_dir)
train_closed_fnames = os.listdir(train_closed_dir)

DESIRED_ACCURACY = 0.99


model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(16, 3, padding='same',
activation='relu',input_shape=(150,150,3)),
```

```
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(32, 3, padding='same',
activation='relu'),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(64, 3, padding='same',
activation='relu'),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(128, 3, padding='same',
activation='relu'),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(256, 3, padding='same',
activation='relu'),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(512, 3, padding='same',
activation='relu'),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.1),

    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.1),

    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.1),

    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.1),

    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.1),

    tf.keras.layers.Dense(2 , activation='softmax')
])
```

```
model.summary()

from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import
binary_crossentropy,sparse_categorical_crossentropy

model.compile(optimizer=Adam(lr=0.0001),
                loss=sparse_categorical_crossentropy,
                metrics=['acc'])

from tensorflow.keras.preprocessing.image import
ImageDataGenerator

train_datagen = ImageDataGenerator(horizontal_flip =
True,
                                    rotation_range =
45,
                                    brightness_range =
[0.8, 1.0],
                                    zoom_range = 0.2
                                    )


test_datagen  = ImageDataGenerator(horizontal_flip =
True,
                                    rotation_range =
45,
                                    brightness_range =
[0.5, 1.0])

# --------------------
# Flow training images in batches of 20 using
train_datagen generator
# --------------------
train_generator =
train_datagen.flow_from_directory(train_dir,

batch_size=32,

class_mode='binary',
```

```
target_size=(150, 150))

# --------------------
# Flow validation images in batches of 20 using
test_datagen generator
# --------------------
validation_generator =
test_datagen.flow_from_directory(validation_dir,

batch_size=32,

class_mode = 'binary',

target_size = (150, 150))

# from tensorflow.keras.callbacks import
ModelCheckpoint

start_time = time.time()

history = model.fit(
    train_generator,
    validation_data = validation_generator,
    epochs=50,
    verbose=1
)

end_time = time.time()

export_path_sm =
"./models/{}.h5".format("driver_drowsiness_softmax_5")
print(export_path_sm)
tf.saved_model.save(model, export_path_sm)

time_taken = end_time - start_time

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
```

```
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation
accuracy')
plt.title('Training and validation accuracy and the
taken time in training is: ' + str(int(time_taken)) + "
seconds")
plt.legend(loc=0)
plt.figure()
plt.show()


plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation
loss')
plt.title('Training and validation accuracy and the
taken time in training is: ' + str(int(time_taken)) +
"seconds")
plt.legend(loc=0)
plt.figure()
plt.show()
```

2.3. Model training – second trial

model - 12.py

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf
import zipfile
from os import path, getcwd, chdir
import PIL
import PIL.Image
from keras.models import load_model
import h5py
```

```
import pathlib
import cv2
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import os
import zipfile
import time
base_dir = './data/'

train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'test')

# Directory with our training pictures
train_open_dir = os.path.join(train_dir, 'Open')
train_closed_dir = os.path.join(train_dir, 'Closed')

# Directory with our validation pictures
validation_open_dir = os.path.join(validation_dir,
'Open')
validation_close_dir = os.path.join(validation_dir,
'Closed')

train_open_fnames = os.listdir(train_open_dir)
train_closed_fnames = os.listdir(train_closed_dir)

DESIRED_ACCURACY = 0.99


model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, 3, padding='same',
activation='relu',input_shape=(150,150,3)),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(64, 3, padding='same',
activation='relu'),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(128, 3, padding='same',
activation='relu'),
    tf.keras.layers.MaxPooling2D(),
```

```
    tf.keras.layers.Conv2D(128, 3, padding='same',
activation='relu'),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(256, 3, padding='same',
activation='relu'),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(512, 3, padding='same',
activation='relu'),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(1024, 3, padding='same',
activation='relu'),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.1),

    tf.keras.layers.Dense(1024, activation='relu'),
    tf.keras.layers.Dropout(0.1),

    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.1),

    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.1),

    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.1),

    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.1),

    tf.keras.layers.Dense(2 , activation='softmax')
])

model.summary()
```

```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import
binary_crossentropy,sparse_categorical_crossentropy

model.compile(optimizer=Adam(lr=0.0001),
                loss=sparse_categorical_crossentropy,
                metrics=['acc'])

from tensorflow.keras.preprocessing.image import
ImageDataGenerator

train_datagen = ImageDataGenerator(horizontal_flip =
True,
                                    rotation_range = 45
)


test_datagen  = ImageDataGenerator(horizontal_flip =
True,
                                    rotation_range = 45
                                   )

# --------------------
# Flow training images in batches of 20 using
train_datagen generator
# --------------------
train_generator =
train_datagen.flow_from_directory(train_dir,

batch_size=32,

class_mode='binary',

target_size=(150, 150))

# --------------------
# Flow validation images in batches of 20 using
test_datagen generator
# --------------------
validation_generator =
```

```
test_datagen.flow_from_directory(validation_dir,

batch_size=32,

class_mode = 'binary',

target_size = (150, 150))

# from tensorflow.keras.callbacks import
ModelCheckpoint

start_time = time.time()

history = model.fit_generator(
    train_generator,
    validation_data = validation_generator,
    epochs=20,
    verbose=1
)

end_time = time.time()

export_path_sm =
"./models/{}.h5".format("driver_drowsiness_softmax_12")
print(export_path_sm)
tf.saved_model.save(model, export_path_sm)

time_taken = end_time - start_time

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation
accuracy')
plt.title('Training and validation accuracy and the
```

```
taken time in training is: ' + str(int(time_taken)) +
"seconds")
plt.legend(loc=0)
plt.figure()

plt.show()
```

2.4. Model training – third trial (the selected model)

model - 13.py

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf
import zipfile
from os import path, getcwd, chdir
import PIL
import PIL.Image
from keras.models import load_model
import h5py
import pathlib
import cv2
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import os
import zipfile
import time
base_dir = './data/'

train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'test')

# Directory with our training pictures
train_open_dir = os.path.join(train_dir, 'Open')
train_closed_dir = os.path.join(train_dir, 'Closed')

# Directory with our validation pictures
validation_open_dir = os.path.join(validation_dir,
'Open')
validation_close_dir = os.path.join(validation_dir,
```

```
'Closed')

train_open_fnames = os.listdir(train_open_dir)
train_closed_fnames = os.listdir(train_closed_dir)


# export_path_sm =
"./models/driver_drowsiness_softmax_11.h5"
# print(export_path_sm)
#
# model = tf.keras.models.load_model(export_path_sm)

from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import
binary_crossentropy,sparse_categorical_crossentropy


from tensorflow.keras.preprocessing.image import
ImageDataGenerator

train_datagen = ImageDataGenerator()
test_datagen  = ImageDataGenerator()

# --------------------
# Flow training images in batches of 20 using
train_datagen generator
# --------------------
train_generator =
train_datagen.flow_from_directory(train_dir,

batch_size=32,

class_mode='binary',

target_size=(150, 150))

# --------------------
# Flow validation images in batches of 20 using
test_datagen generator
# --------------------
```

```
validation_generator =
test_datagen.flow_from_directory(validation_dir,

batch_size=32,

class_mode = 'binary',

target_size = (150, 150))

# from tensorflow.keras.callbacks import
ModelCheckpoint


model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, 3, padding='same',
activation='relu',input_shape=(150,150,3)),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(64, 3, padding='same',
activation='relu'),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(128, 3, padding='same',
activation='relu'),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(256, 3, padding='same',
activation='relu'),
    tf.keras.layers.MaxPooling2D(),


    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.1),

    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.2),

    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.1),
```

```
    tf.keras.layers.Dense(2 , activation='softmax')
])

model.summary()



model.compile(optimizer=Adam(lr=0.00005),
                loss=sparse_categorical_crossentropy,
                metrics=['acc'])



start_time = time.time()

history = model.fit_generator(
    train_generator,
    validation_data = validation_generator,
    epochs=50,
    verbose=1
)

end_time = time.time()

export_path_sm =
"./models/{}.h5".format("driver_drowsiness_softmax_13")
print(export_path_sm)
tf.saved_model.save(model, export_path_sm)

time_taken = end_time - start_time

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation
```

```
accuracy')
plt.plot(epochs, loss, 'r', label='Training accuracy')
plt.plot(epochs, val_loss, 'b', label='Validation
accuracy')
plt.title('Training and validation accuracy and the
taken time in training is: ' + str(int(time_taken)) +
"seconds for " +epochs + " epochs" )
plt.legend(loc=0)
plt.figure()

plt.show()
```

2.5. Model training – fourth trial

model - 14.py

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf
import zipfile
from os import path, getcwd, chdir
import PIL
import PIL.Image
from keras.models import load_model
import h5py
import pathlib
import cv2
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import os
import zipfile
import time
base_dir = './data/'

train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'test')

# Directory with our training pictures
train_open_dir = os.path.join(train_dir, 'Open')
```

```
train_closed_dir = os.path.join(train_dir, 'Closed')

# Directory with our validation pictures
validation_open_dir = os.path.join(validation_dir,
'Open')
validation_close_dir = os.path.join(validation_dir,
'Closed')

train_open_fnames = os.listdir(train_open_dir)
train_closed_fnames = os.listdir(train_closed_dir)

DESIRED_ACCURACY = 0.99

from tensorflow.keras.applications.inception_v3 import
InceptionV3
path_inception =
f"./inception_v3_weights_tf_dim_ordering_tf_kernels_not
op.h5"

local_weights_file = path_inception

pre_trained_model = InceptionV3(input_shape=(150, 150,
3),
                                include_top=False,
                                weights=None)

pre_trained_model.load_weights(local_weights_file)

# Make all the layers in the pre-trained model non-
trainable
for layer in pre_trained_model.layers:
    layer.trainable = False

# Print the model summary
pre_trained_model.summary()

last_layer = pre_trained_model.get_layer('mixed7')
print('last layer output shape: ',
last_layer.output_shape)
last_output = last_layer.output
```

```
from tensorflow.keras import layers
#calback = myCallback()
# Flatten the output layer to 1 dimension
x = layers.Flatten()(last_output)
# Add a fully connected layer with 1,024 hidden units
and ReLU activation
x = layers.Dense(1024,activation='relu')(x)
# Add a dropout rate of 0.2
x = layers.Dropout(0.2)(x)
# Add a fully connected layer with 512 hidden units and
ReLU activation
x = layers.Dense(512,activation='relu')(x)
# Add a dropout rate of 0.2
x = layers.Dropout(0.2)(x)
# Add a fully connected layer with 256 hidden units and
ReLU activation
x = layers.Dense(256,activation='relu')(x)
# Add a dropout rate of 0.2
x = layers.Dropout(0.2)(x)
# Add a fully connected layer with 128 hidden units and
ReLU activation
x = layers.Dense(128,activation='relu')(x)
# Add a dropout rate of 0.2
x = layers.Dropout(0.2)(x)
# Add a fully connected layer with 64 hidden units and
ReLU activation
x = layers.Dense(64,activation='relu')(x)
# Add a dropout rate of 0.2
x = layers.Dropout(0.2)(x)
# Add a fully connected layer with 10 hidden units and
ReLU activation
x = layers.Dense(10,activation='relu')(x)
# Add a dropout rate of 0.2
x = layers.Dropout(0.2)(x)
# Add a final sigmoid layer for classification
x = layers.Dense(2, activation='softmax')(x)

from tensorflow.keras import Model
model = Model( pre_trained_model.input, x)
```

```
model.summary()

from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import
binary_crossentropy,sparse_categorical_crossentropy

model.compile(optimizer=Adam(lr=0.00),
                loss=sparse_categorical_crossentropy,
                metrics=['acc'])

from tensorflow.keras.preprocessing.image import
ImageDataGenerator

train_datagen = ImageDataGenerator()


test_datagen  = ImageDataGenerator()

# --------------------
# Flow training images in batches of 20 using
train_datagen generator
# --------------------
train_generator =
train_datagen.flow_from_directory(train_dir,

batch_size=32,

class_mode='binary',

target_size=(150, 150))

# --------------------
# Flow validation images in batches of 20 using
test_datagen generator
# --------------------
validation_generator =
test_datagen.flow_from_directory(validation_dir,

batch_size=32,
```

```
class_mode = 'binary',

target_size = (150, 150))

# from tensorflow.keras.callbacks import
ModelCheckpoint

start_time = time.time()

history = model.fit_generator(
    train_generator,
    validation_data = validation_generator,
    epochs=20,
    verbose=1
)

end_time = time.time()

export_path_sm =
"./models/{}.h5".format("driver_drowsiness_softmax_14")
print(export_path_sm)
tf.saved_model.save(model, export_path_sm)

time_taken = end_time - start_time

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation
accuracy')
plt.plot(epochs, loss, 'r', label='Training accuracy')
plt.plot(epochs, val_loss, 'b', label='Validation
accuracy')
plt.title('Training and validation accuracy and the
```

```
taken time in training is: ' + str(int(time_taken)) +
"seconds for " +epochs + " epochs" )
plt.legend(loc=0)
plt.figure()

plt.show()
```

3. Driver Drowsiness Detection Project
    3.1. Project classes.

project_classes.py

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf
import zipfile
from os import path, getcwd, chdir
import PIL
import PIL.Image
from keras.models import load_model
import h5py
import pathlib
import cv2
import os
import zipfile
import numpy as np
# from keras.preprocessing import image
import cv2
# from tensorflow import keras
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import
ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint
from keras.preprocessing import image
from playsound import playsound
import winsound
from pygame import mixer
```

```
import warnings
warnings.filterwarnings("ignore")

#######################################################
#############
#######################################################
#############
# here we load the trained model the give us the facial
key points
#######################################################
#############
#######################################################
#############
export_path_sm = "./models/facial_points.h5"
keypoints_model =
tf.keras.models.load_model(export_path_sm)
#######################################################
#############
#######################################################
#############
# here we load the trained model the predict whether
the eye is an
# open eye or closed eye
#######################################################
#############
#######################################################
#############
export_path_sm =
"./models/driver_drowsiness_softmax_13.h5"
eye_model = tf.keras.models.load_model(export_path_sm)
#######################################################
#############
#######################################################
#############

# this function take the image and the center of the
rectangle wanted to be drawn
# and also the text and draw the rectangle and put the
text above it
def draw_rectangle_with_text(img, x ,y , w , h ,color ,
```

```
text):
    cv2.rectangle(img, (x - w, y - h), (x + w, y + h),
color, 2)
    # write over the rectangle the percentage of being
open
    cv2.putText(img, text, (x - w - 6, y - h - 6),
cv2.FONT_HERSHEY_SIMPLEX, 0.4, color, 2)
    return img

# this fucntion take (96 x 96) and use the facial
keypoint model to get 15 important point in the face
# but we just use the eyes center points
def getpoints(img):
    temp = np.expand_dims(img, axis=2)
    temp=temp/(255.0)
    temp = np.expand_dims(temp, axis=0)
    points = keypoints_model.predict(temp)
    return points

# this is the face class
class FaceAndPoints:
    def __init__(self, frame, x, y, w, h, gray_face):
        self.gray_frame = cv2.cvtColor(frame,
cv2.COLOR_BGR2GRAY)
        self.frame = frame
        self.x = x
        self.y = y
        self.width = w
        self.height = h
        self.gray_face = gray_face
#get_larges_face(self.gray_frame)

    # using this function we call the getpoint function
ti get the points
    # and resize the face to (400 x 400) to be mor
visible
    # then return the np.array of points and the face
    def get_face_keypoints(self):
        face_for_keypoints = cv2.resize(self.gray_face,
(96, 96))
```

```
        keypoints = getpoints(face_for_keypoints)

        face = self.frame[self.y: self.y + self.height,
self.x: self.x + self.width]
        face = cv2.resize(face, (400, 400))

        return face, keypoints

# this is the eye class
class Eye:
    def __init__(self, x_index, y_index, w, h, points,
face_image):
        self.x_index = x_index
        self.y_index = y_index
        self.points = points
        self.x_center =
int(int(int(self.points[0][self.x_index]))*400/96)
        self.y_center =
int(int(int(self.points[0][self.y_index]))*400/96)
        self.width = w
        self.height = h
        self.eye_is_open = False
        self.face_image = face_image

        # extract the eye from the image
        self.eye_img = self.face_image[self.y_center -
h - 10: self.y_center + h, self.x_center - w-5:
self.x_center + w+5]
        self.eye_class = 0
        self.color = (0,0,0)

    # in this  function we predict whether the eye is
closed of open using the eye model
    def classify_the_eye(self):
        # here we resize the  eye image, because the
model accept 150 x 150 images only
        resized_eye_img = cv2.resize(self.eye_img,
(150, 150))
        # here we prepare the eye image for the model
to be np.array(150 x 150 x 1)
```

```
        resized_eye_for_predection =
np.array([resized_eye_img])
        self.eye_class =
eye_model.predict(resized_eye_for_predection)[0][1]

    # in the draw model we draw rectangle around the
eye with green if it is open and red if it is closes
    def draw_eye(self):
        if self.eye_class > 0.5:
            # if self.eye_class <= 0.65 :
self.eye_class = (self.eye_class + 0.35)
            print('this eye: ', self.eye_class)
            self.color = (0, 255, 0)  # green color for
open eyes
            self.eye_is_open = True  # set the boolean
to true


        else:
            print('eye: ', self.eye_class)
            self.color = (0, 0, 255)
            self.eye_is_open = False

        text_on_the_eye = 'this eye is ' +
str(int(float(self.eye_class) * 100)) + "% open"
        self.face_image =
draw_rectangle_with_text(self.face_image,
self.x_center, self.y_center,

self.width, self.height, self.color,
text=text_on_the_eye)



        self.face_image =
draw_rectangle_with_text(self.face_image,
self.x_center, self.y_center,
                                                       1,
1, self.color, text="")
```

```
        #
        # j=4
        # while(j<29):
        #
        #     self.face_image =
draw_rectangle_with_text(self.face_image,
int(int(int(self.points[0][j])) * 400 / 96),
        #
int(int(int(self.points[0][j+1])) * 400 / 96),
        #
1, 1, self.color, text="")
        #     j+=2


        return self.face_image
```

3.2. Project run.

usethemodel.py

```
import os

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf
import zipfile
from os import path, getcwd, chdir
import PIL
import PIL.Image
from keras.models import load_model
import h5py
import pathlib
import cv2
import tensorflow.keras as keras
import os
import zipfile
import numpy as np
from keras.preprocessing import image
import cv2
from tensorflow import keras
```

```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import
ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint
from keras.preprocessing import image
import matplotlib.pyplot as plt
from playsound import playsound
import winsound
from pygame import mixer
import project_classes as PC


#######################################################
#############
#######################################################
#############
# input = gray frame
# this function gets the larges face in the frame
# according to the area of the faces
# and return its , output = x,y,width,height
#######################################################
#############
#######################################################
#############
def get_larges_face(gray_frame):
    facecascade =
cv2.CascadeClassifier(cv2.data.haarcascades +
"haarcascade_frontalface_default.xml")
    faces = facecascade.detectMultiScale(gray_frame)
    # to make sure that there is a face in the image
    if len(faces) > 0:
        # to get the biggest face in the image
        max_w = 0
        max_h = 0
        index = -1
        for i in range(len(faces)):
            (x, y, w, h) = faces[i]
            # we compare between the area of the faces
```

```
if there is more than one face
            if ((w * h) > (max_w * max_h)):
                max_w = w
                max_h = h
                index = i
        # here we get the width and the hight of the
face to crop the face from the image
        (x, y, width, height) = faces[index]

        gray_face = gray_frame[y: y + height, x: x +
width]
        return x, y, width, height, gray_face
    # if the number of faces is zero "no face exist
return all zeros"
    else:
        return 0, 0, 0, 0, gray_frame


#######################################################
#############
#######################################################
#############
# input = img, x, y, width , height , color of
rectangle ,
# ,text on the top of the rectangle , thickness of the
rectangle side
#######################################################
#############
# this function draw a rectangle inside the img where
x,y is its center
# and after that it put text on the top of the
rectangle
# and return the img after the modification (after
drawing)
#######################################################
#############
#######################################################
#############
def draw_rectangle_with_text(img, x, y, w, h, color,
text, thickness):
```

```
    cv2.rectangle(img, (x, y), (x + w, y + h), color,
2)
    # write over the rectangle the percentage of being
open
    cv2.putText(img, text, (x - 10, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.4, color, thickness)
    return img


#######################################################
#######################################################
#
# in this function we check how many eye are closed and
how many are open and acourding to that
# we draw the rectangle around the face
# if both are open the color of rectangle is green and
decrease the drowsycounter gradualy and the thickness
# as well as the drowsy counter
# else if both are closed the color of rectangle is red
and increase the drowsycounter gradualy and the
thickness
# aand if one is closed and one is open the color is
blue
#######################################################
#######################################################
#
def eyesanddrowsyness(left_eye, right_eye, img,
drowsycounter):
    if left_eye.eye_is_open and right_eye.eye_is_open:
        color = (0, 255, 0)  # green
        thickness = int((
                            drowsycounter + 2)
/ 2)  # decrease the thickness, and the "+2" is just to
make sure that it will one or more
        drowsycounter = (drowsycounter > 0) *
(drowsycounter - 1)  # decrement the drowsycounter by
one

        # draw and write on the img , the image is 400
x 400 , so i used 15, 15, to leave some space from up
```

```
and left
        # and made the width and height 370 so that
will leave the same space from right and down
"15+370=385"
        img = draw_rectangle_with_text(img, 15, 15,
370, 370, color, text='both are open',
thickness=thickness)


    elif left_eye.eye_is_open or right_eye.eye_is_open:
        color = (255, 0, 0)  # blue ... "yes, in opencv
it is (B,G,R) not (R,G,B)" so this is blue not red
        thickness = int((drowsycounter + 2))
        drowsycounter = (drowsycounter > 0) *
(drowsycounter - 1)  # decrement the drowsycounter by
one

        # draw and write on the img , the image is 400
x 400 , so i used 15, 15, to leave some space from up
and left
        # and made the width and height 370 so that
will leave the same space from right and down
"15+370=385"
        img = draw_rectangle_with_text(img, 15, 15,
370, 370, color, text="one is open",
thickness=thickness)

    else:
        color = (0, 0, 255)  # red ... "yes, in opencv
it is (B,G,R) not (R,G,B)" so this is red not blue
        thickness = int((drowsycounter + 2) / 1.5)  #
increase the thickness,by 2
        drowsycounter = (drowsycounter + 1 * (
                    drowsycounter <= 20))  # increase
the thickness,by 1 + make sure it will not exceed 20

        # draw and write on the img , the image is 400
x 400 , so i used 15, 15, to leave some space from up
and left
        # and made the width and height 370 so that
```

```
will leave the same space from right and down
"15+370=385"
        img = draw_rectangle_with_text(img, 15, 15,
370, 370, color, text="both are closed",
thickness=thickness)

    print('drowsy counter: ', drowsycounter)
    return img, drowsycounter


#######################################################
#######################################################
##
#######################################################
#######################################################
##
# we can consider this function as a main function
# it takes the frame and the drowsycounter
# get the largest face parameters ( x , y , w , h )
using get_larges_face() function
# check if width and height are zeros which means there
is no faces in thr frame
# so return the frame directly with no changes
# else if there is a face in the frame
# we create face object using face and point class in
the project_classes.py file
# and initialize it with the face parameters face_OB =
PC.FaceAndPoints(frame, x, y, w, h, gray_face)
# get the extracted face and the facial points using
face_OB.get_face_keypoints() function
# then use the returned (driver Face) and (pointes) to
intialize the two eyes objects
#######################################################
#######################################################
##
#######################################################
#######################################################
##
def project_func(frame, drowsycounter):
    gray_frame = cv2.cvtColor(frame,
```

```
cv2.COLOR_BGR2GRAY)

    # get the larges (nearest) face in the frame if
there are more than one
    # if  one return it
    x, y, w, h, gray_face = get_larges_face(gray_frame)

    # check if width and height are zere means no faces
so retern the fram dirctly with no change
    if w == 0 and h == 0:
        return frame, drowsycounter

    # face object
    face_OB = PC.FaceAndPoints(frame, x, y, w, h,
gray_face)

    # get the extracted face and the facial keypoints
    the_driver_face, the_driver_facial_keypoints =
face_OB.get_face_keypoints()

    # Eye object for the left eye
    # PC.Eye(X_Point, Y_point, Width, Height, points,
driver face image)
    left_eye_OB = PC.Eye(0, 1, 55, 55,
the_driver_facial_keypoints, the_driver_face)

    # predect if the eye is clossed of open
    left_eye_OB.classify_the_eye()

    # draw rectangle around the left eye and put the
percentage on it
    frame_with_lefteye_draw = left_eye_OB.draw_eye()

    # Eye object for the left eye
    # PC.Eye(X_Point, Y_point, Width, Height, points,
driver face image)
    right_eye_OB = PC.Eye(2, 3, 55, 55,
the_driver_facial_keypoints, frame_with_lefteye_draw)

    # predect if the eye is clossed of open
```

```
    right_eye_OB.classify_the_eye()

    # draw rectangle around the right eye and put the
percentage on it
    frame_with_eyes = right_eye_OB.draw_eye()

    # draw rectangle around the face with different
colors and different thikness accourding to the
drowsycounter
    # and the closed and open eyes and increase or
decrease the drowsycounter according to the case
    # then return the img with the drawed rectangle and
the drowsy detection
    final_frame, drowsycounter =
eyesanddrowsyness(left_eye_OB, right_eye_OB,
frame_with_eyes, drowsycounter)

    return final_frame, drowsycounter


def adjust_brighness(image, gamma=1.0):
    # build a lookup table mapping the pixel values [0,
255] to
    # their adjusted gamma values
    invGamma = 1.0 / gamma
    table = np.array([((i / 255.0) ** invGamma) * 255
for i in np.arange(0, 256)]).astype("uint8")
    # apply gamma correction using the lookup table
    return cv2.LUT(image, table)


def increaseBrightness(img, value=65):
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    h, s, v = cv2.split(hsv)

    lim = 255 - value
    v[v > lim] = 255
    v[v <= lim] += value

    final_hsv = cv2.merge((h, s, v))
```

```
    img = cv2.cvtColor(final_hsv, cv2.COLOR_HSV2BGR)
    return img


def increaseContrast(img , clipLimit=6):
    # https://learnopencv.com/color-spaces-in-opencv-
cpp-python/
    # -----Converting image to LAB Color model---------
--------------------------
    lab = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)

    # -----Splitting the LAB image to different
channels-------------------------
    l, a, b = cv2.split(lab)

    # -----Applying CLAHE to L-channel-----------------
--------------------------
    clahe = cv2.createCLAHE(clipLimit, tileGridSize=(8,
8))
    cl = clahe.apply(l)

    # -----Merge the CLAHE enhanced L-channel with the
a and b channel-----------
    limg = cv2.merge((cl, a, b))

    # -----Converting image from LAB Color model to RGB
model--------------------
    contrasted = cv2.cvtColor(limg, cv2.COLOR_LAB2BGR)
    img = contrasted

    # r, g, b = cv2.split(img)
    # clahe = cv2.createCLAHE(clipLimit=2.0,
tileGridSize=(8,8))
    # cl1 = clahe.apply(r)
    # clahe = cv2.createCLAHE(clipLimit=2.0,
tileGridSize=(8,8))
    # cl2 = clahe.apply(g)
    # clahe = cv2.createCLAHE(clipLimit=2.0,
tileGridSize=(8,8))
    # cl3 = clahe.apply(b)
```

```
    #
    # limg = cv2.merge((cl1, cl2, cl3))

    return img


def decreaseNoice(img):
    # ----- apply gaussian blurring --------------------
------------------------
    gausian_blured = cv2.GaussianBlur(img, (3, 3), 0)

    # ----- apply median blurring ----------------------
----------------------
    median_blured = cv2.medianBlur(img, 3)

    # img = gausian_blured
    img = median_blured

    # # ----- Create kernals for sharpening -----------
--------------------------------
    # kernel1 = np.array([[-1, -1, -1],
    #                     [-1, 9, -1],
    #                     [-1, -1, -1]])
    #
    # kernel2 = np.array([[-1.5, -1.5, -1.5],
    #                     [-1.5, 13.5, -1.5],
    #                     [-1.5, -1.5, -1.5]])
    #
    # kernel3 = np.array([[-0.5, -0.5, -0.5],
    #                     [-0.5, 4.5, -0.5],
    #                     [-0.5, -0.5, -0.5]])
    #
    # kernel4 = np.array([[-2, -2, -2],
    #                     [-2, 18, -2],
    #                     [-2, -2, -2]])
    #
    # kernel5 = np.array([[-0.25, -0.25, -0.25],
    #                     [-0.25, 2.25, -0.25],
    #                     [-0.25, -0.25, -0.25]])
    #
```

```
    # # -----Applying the kernal------------------------
--------------------
    # sharpened = cv2.filter2D(img, -1, kernel3)
    # img = sharpened
    return img


def preprocess_the_frame(img, condition=0):
    # conition = 0 for high light
    # conition = 1 for good light
    # conition = 2 for bad light

    # if condition==0:
    #     return img
    #
    # elif(condition==1):
    #     img = adjust_brighness(img, 1)
    #     # img = increaseBrightness(img,50)
    #     img = increaseContrast(img,2)
    #     # img = decreaseNoice(img)
    #
    # elif condition==2:
    #     img = adjust_brighness(img, 2.5)
    #     # img = increaseBrightness(img,50)
    #     img = increaseContrast(img,2.5)
    #     # img = decreaseNoice(img)

    img = adjust_brighness(img,7)
    img = increaseContrast(img)
    img = decreaseNoice(img)
    # img = cv2.fastNlMeansDenoisingColored(img , None)
    return img


# laptopcamera = 0
# secondarycamera = 1

# cap = cv2.VideoCapture(0)  # get stream of frames
drowsycounter = 0  # initialization of the drowsyness
counting
```

```
mixer.init()
sound = mixer.Sound('alarm.mp3')  # this is the alarm
sound
```

gui.py

```
# import PIL
# from PIL import Image,ImageTk
# import pytesseract
# import cv2
# from tkinter import *
#
#
# Gui_Win=tk.Tk()
# Gui_Win.title('Gui_Task_1')
# Gui_Win.geometry('800x900')
# Gui_Win.configure(bg='blue')
#
#
# width, height = 800, 600
# cap = cv2.VideoCapture(0)
# cap.set(cv2.CAP_PROP_FRAME_WIDTH, width)
# cap.set(cv2.CAP_PROP_FRAME_HEIGHT, height)
#
# root = Tk()
# root.bind('<Escape>', lambda e: root.quit())
# lmain = Label(root)
# lmain.pack()
#
# def show_frame():
#     _, frame = cap.read()
#     frame = cv2.flip(frame, 1)
#     cv2image = cv2.cvtColor(frame,
cv2.COLOR_BGR2RGBA)
#     img = PIL.Image.fromarray(cv2image)
#     imgtk = ImageTk.PhotoImage(image=img)
#     lmain.imgtk = imgtk
#     lmain.configure(image=imgtk)
#     lmain.after(10, show_frame)
```

```
#
# show_frame()
# root.mainloop()


import sys
import cv2
import threading
import tkinter as tk
import tkinter.ttk as ttk
from queue import Queue
from PIL import Image
from PIL import ImageTk
from pygame import mixer
import usethemodel as pjbody



class App(tk.Frame):
    def __init__(self, parent, title):
        tk.Frame.__init__(self, parent)
        self.is_running = False
        self.thread = None
        self.queue = Queue()
        self.photo =
ImageTk.PhotoImage(Image.new("RGB", (800, 600),
"white"))
        parent.wm_withdraw()
        parent.wm_title(title)
        self.create_ui()
        self.grid(sticky=tk.NSEW)
        self.bind('<<MessageGenerated>>',
self.on_next_frame)
        parent.wm_protocol("WM_DELETE_WINDOW",
self.on_destroy)
        parent.grid_rowconfigure(0, weight = 1)
        parent.grid_columnconfigure(0, weight = 1)
        parent.wm_deiconify()

    def create_ui(self):
```

```
        self.button_frame = ttk.Frame(self)

        self.stop_button =
ttk.Button(self.button_frame, text="Stop",
command=self.stop)
        self.stop_button.pack(side=tk.RIGHT)

        self.start_button =
ttk.Button(self.button_frame, text="Start",
command=self.start)
        self.start_button.pack(side=tk.RIGHT)

        self.view = ttk.Label(self, image=self.photo)
        self.view.pack(side=tk.TOP, fill=tk.BOTH,
expand=True)
        self.button_frame.pack(side=tk.BOTTOM,
fill=tk.X, expand=True)

    def on_destroy(self):
        self.stop()
        self.after(20)
        if self.thread is not None:
            self.thread.join(0.2)
        self.winfo_toplevel().destroy()

    def start(self):
        self.is_running = True
        self.thread =
threading.Thread(target=self.videoLoop, args=())
        self.thread.daemon = True
        self.thread.start()

    def stop(self):
        self.is_running = False

    def videoLoop(self, mirror=False):
        No=0 #1
        cap = cv2.VideoCapture(No)
        cap.set(cv2.CAP_PROP_FRAME_WIDTH, 800)
        cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 600)
```

```
        drowsycounter = 0  # initialization of the
drowsyness counting
        mixer.init()
        sound = mixer.Sound('alarm.mp3')  # this is the
alarm sound
        while self.is_running:
            ret, to_draw = cap.read()

            # make it flipped like mirror
            to_draw = cv2.flip(to_draw, 1)

            # conition = 0 for high light
            # conition = 1 for good light
            # conition = 2 for bad light
            to_draw =
pjbody.preprocess_the_frame(to_draw, condition=0)

            # get the latest frame with all the drawing
on it after applying the models
            to_draw, drowsycounter =
pjbody.project_func(to_draw, drowsycounter)

            # resize to a fixed size
            to_draw = cv2.resize(to_draw, (600, 600))

            # display///////////
            # cv2.imshow('', frame)

            # if his both eyes was closed for 7 frames
so alarm play
            if drowsycounter >= 5:
                sound.play()

            if drowsycounter < 5:
                sound.stop()
            # self.label_frame = ttk.Frame(self)

            # if cv2.waitKey(1) & 0xFF == ord('q'):
            #     break
```

```
            if mirror is True:
                to_draw = to_draw[:,::-1]
            image = cv2.cvtColor(to_draw,
cv2.COLOR_BGR2RGB)
            self.queue.put(image)
            self.event_generate('<<MessageGenerated>>')

    def on_next_frame(self, eventargs):
        if not self.queue.empty():
            image = self.queue.get()
            image = Image.fromarray(image)
            self.photo = ImageTk.PhotoImage(image)
            self.view.configure(image=self.photo)


def main(args):
    root = tk.Tk()
    app = App(root, "OpenCV Image Viewer")
    root.mainloop()

if __name__ == '__main__':
    sys.exit(main(sys.argv))
```

References

[1] Owens Justin M., Dingus Thomas A., Guo Feng, Fang Youjia, Miguel Perez, Julie McClafferty, Brian Tefft, "Prevalence of Drowsy Driving Crashes: Estimates from a Large-Scale Naturalistic Driving Study," February 2018. [Online]. Available: https://aaafoundation.org/prevalence-drowsy-driving-crashes-estimates-large-scale-naturalistic-driving-study/.

[2] National Center for Statistics and Analysis, "Overview of motor vehicle crashes in 2019," National Highway Traffic Safety Administration, USA, December 2020.

[3] Rateb Jappar, Mohammed Shinoy, Mohamed Kharbeche, Khalifa Al-Khalifax, Moez Krichenz, Kamel Barkaouiy, "Driver Drowsiness Detection Model Using Convolutional Neural Networks Techniques," Qatar University, Qatar, 2020.

[4] Ching-Hua Weng, Ying-Hsiu Lai, Shang-Hong Lai, "Driver Drowsiness Detection Dataset," National Tsuing Hua University, November 2016. [Online]. Available: http://cv.cs.nthu.edu.tw/php/callforpaper/datasets/DDD/.

[5] Massoz, Q., Verly, J., Van Droogenbroeck, "Multi-Timescale Drowsiness Characterization Based on a Video of a Driver's Face," Department of Electrical Engineering and Computer Science, Faculty of Applied Science, University of Liège, Belgium, August 2018. [Online]. Available: https://www.mdpi.com/1424-8220/18/9/2801.

[6] Y. Bengio, "Facial Keypoints Detection," Kaggle, July 2013. [Online]. Available: https://www.kaggle.com/c/facial-keypoints-detection/data.

[7] T. Oliphant, "NumPy," Community project, 2005. [Online]. Available: https://numpy.org/.

[8] Intel, "OpenCV," June 2000. [Online]. Available: https://opencv.org.

[9] Google, "TensorFlow," November 2015. [Online]. Available: https://www.tensorflow.org/.

[10] John D. Hunterr, Darren Dale, Eric Firing, Michael Droettboom, "Matplotlib," 2002. [Online]. Available: https://matplotlib.org/.

[11] P. S. Foundation, "Miscellaneous operating system interfaces," [Online]. Available: https://docs.python.org/3/library/os.html.

[12] W. McKinney, "Pandas," 11 January 2008. [Online]. Available: https://pandas.pydata.org/.