Drowsiness Detection and Alert System Using Deep Learning

Internship Project Report

Submitted by

KONDETI HARSHAVARDHAN

22ECB0C18

NIT WARANGAL

Under the guidance of

Prof. L. Anjaneyulu

Professor

Department of Electronics and Communication Engineering

National Institute of Technology Warangal

July 2024

NATIONAL INSTITUTE OF TECHNOLOGY

WARANGAL - 506 004

CERTIFICATE

This is to certify that _KONDETI HARSHAVARDHAN_ of National Institute of Technology Warangal have successfully completed a Project titled "Drowsiness Detection and Alert System using Deep Learning**", as part of Summer Internship Programme under my guidance at National Institute of Technology, Warangal, Telangana, during 20/5/2024 to 30/6/2024.**

Prof L.Anjaneyulu, ECE

Visvesvaraya Centre for Skill Development

NIT, Warangal

Table of Content

Abstract

Driving requires full attention, and distractions like drowsiness greatly increase the risk of accidents. Each year, approximately 1.35 million people die in road traffic crashes, costing most countries 3% of their GDP. Our project aims to detect driver drowsiness or other risky activities using Deep Learning models to classify the driver's state from images.

This project introduces a Drowsy Driver Detection System that uses Deep Learning. It continuously monitors real-time data such as facial expressions, eye movements, and vehicle dynamics. This information is used to recognize patterns and extract features. The system is designed to fit seamlessly into different vehicles and provides alerts (visual, auditory, and haptic) when it detects signs of drowsiness. By focusing on simplicity and efficiency, the system is both accessible and cost-effective, making it suitable for widespread use in the automotive industry.

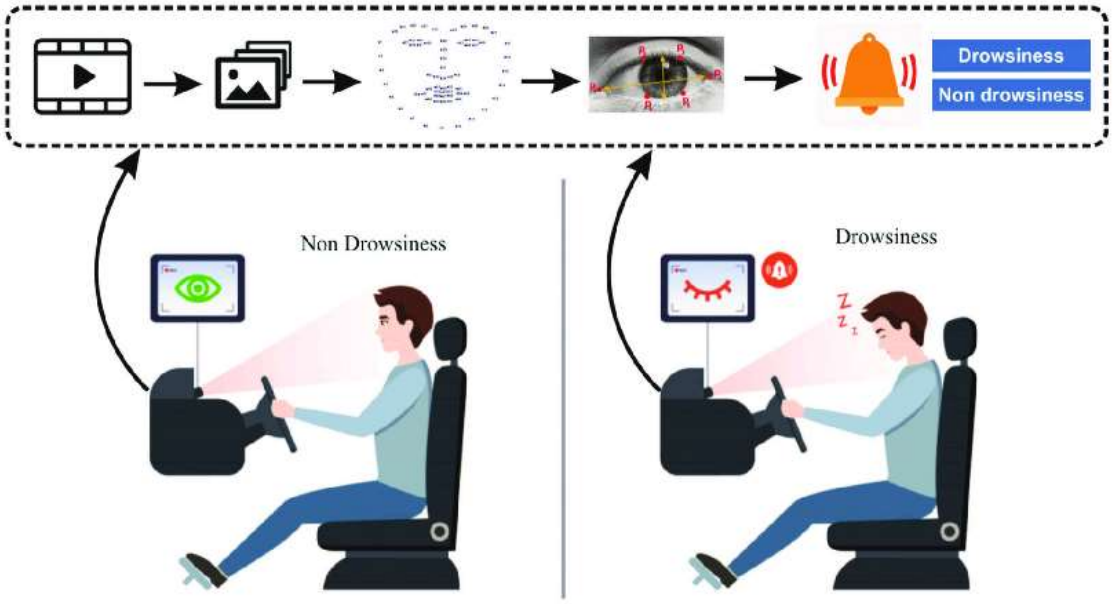
INTRODUCTION

Drowsiness refers to a state of near sleep characterised by a strong desire to rest. It has two distinct aspects: the habitual state that precedes sleep and the chronic state, which occurs regardless of the daily rhythm. Drowsiness is particularly hazardous when performing tasks that require continuous focus, such as driving. When a driver feels excessively tired, they become drowsy, increasing the likelihood of road accidents. Developing technologies to detect or prevent driver drowsiness is a significant challenge in accident prevention systems.

Given the danger that drowsiness poses on the road, it is crucial to develop methods to counter its effects. The goal of this project is to develop a simulation of a drowsiness detection system. The focus will be on designing a system that accurately monitors the state of the driver's eyes and mouth. By tracking these indicators, early signs of driver drowsiness can be detected, potentially preventing traffic accidents. Yawn detection is also a method used to assess driver fatigue, as tired individuals frequently yawn to increase oxygen intake before becoming sleepy.

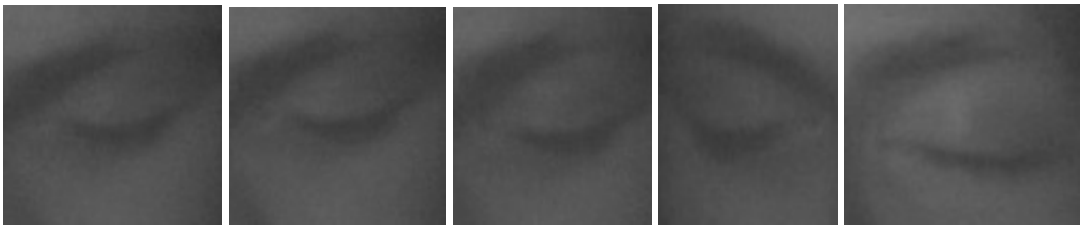
Fatigue and sleepiness detection involves analysing a sequence of facial images and observing the duration for which the eyes and mouth remain open or closed. One effective method for detecting eye closure is PERCLOS, which measures the percentage of time the eyes are closed over a specific period. Facial image analysis, a popular research area with applications such as facial recognition and human identification for security systems, is integral to this project. The project focuses on the localization of the eyes and mouth within facial images, applying image processing algorithms to determine their positions.

Once the system locates the eyes, it can ascertain whether the eyes and mouth are open or closed, thereby detecting signs of fatigue and sleepiness. This approach aims to provide an effective solution for monitoring driver alertness and enhancing road safety.

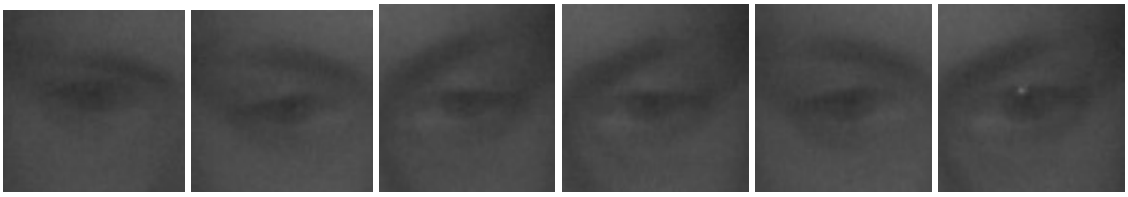

Drowsiness
Non drowsiness
Non Drowsiness
Drowsiness

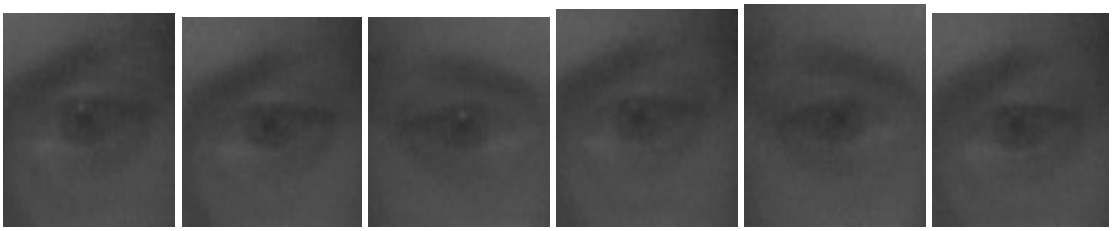
Training Data Images

Closed Eyes



Open Eyes



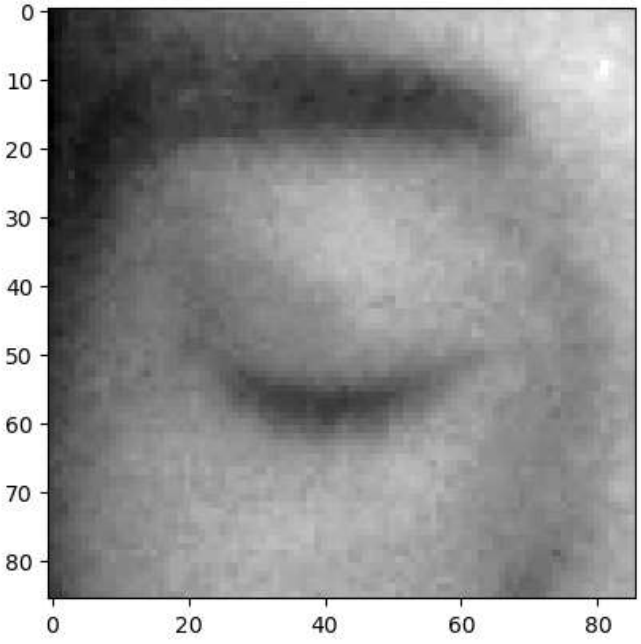


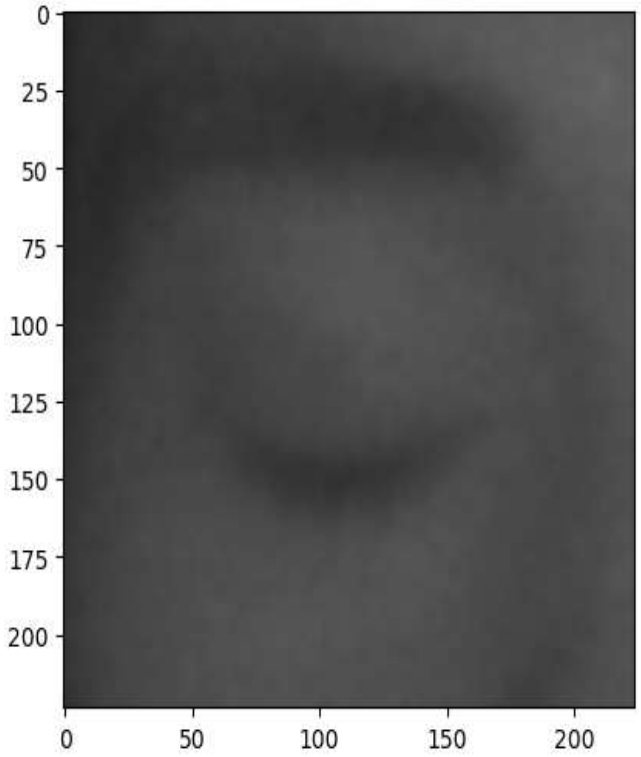
Data Preprocessing

Data preprocessing is a fundamental step in preparing data for machine learning, particularly in computer vision tasks like drowsiness detection. The first phase involves collecting a comprehensive and diverse dataset of images or videos showing drivers with varying states of alertness, such as open and closed eyes. This dataset must include a wide range of lighting conditions, angles, and facial variations to ensure the model can generalize well to different real-world scenarios. Once the data is collected, the next step is data cleaning, which involves removing corrupted or irrelevant images that could degrade model performance. It is also essential to verify that the images are correctly labeled, as mislabeled data can lead to incorrect model predictions.

Following data cleaning, the next crucial step is detecting faces and eyes within each image using Haar cascade classifiers. This technique isolates the regions of interest (ROIs), specifically the eyes, which are critical for detecting drowsiness. The detected ROIs are then cropped and resized to a standard dimension, typically 224x224 pixels, to ensure uniform input size for the neural network. Resizing helps in reducing computational complexity and ensures that the model can process the images efficiently.

After resizing, the images are normalized, scaling the pixel values to a range of [0, 1]. Normalization improves the convergence of the neural network during training by ensuring that the input data is on a consistent scale. Data augmentation techniques such as rotation, flipping, and zooming may also be applied to increase the diversity of the training dataset, which helps in making the model more robust to variations in the input images. Through these preprocessing steps, the data is transformed into a suitable format that enhances the performance and accuracy of the drowsiness detection system.


0
10
20
30
40
50
60
70
80
0
20
40
60
80


0
25
50
75
100
125
150
175
200
0
50
100
150
200

Working of Training Model

The system primarily focuses on detecting the driver's eye state (open or closed) to determine if they are drowsy. This solution employs the OpenCV library, which is renowned for its extensive computer vision functionalities, including face and eye detection using pre-trained Haar cascade classifiers.

The program starts by importing the necessary library, cv2, which is the OpenCV library in Python. OpenCV is extensively used for real-time image processing and computer vision tasks. The Haar cascade classifiers used in this code are pre-trained models included in the OpenCV library, which detect objects in an image. These classifiers are based on the Haar feature-based cascade classifier algorithm, introduced by Paul Viola and Michael Jones.

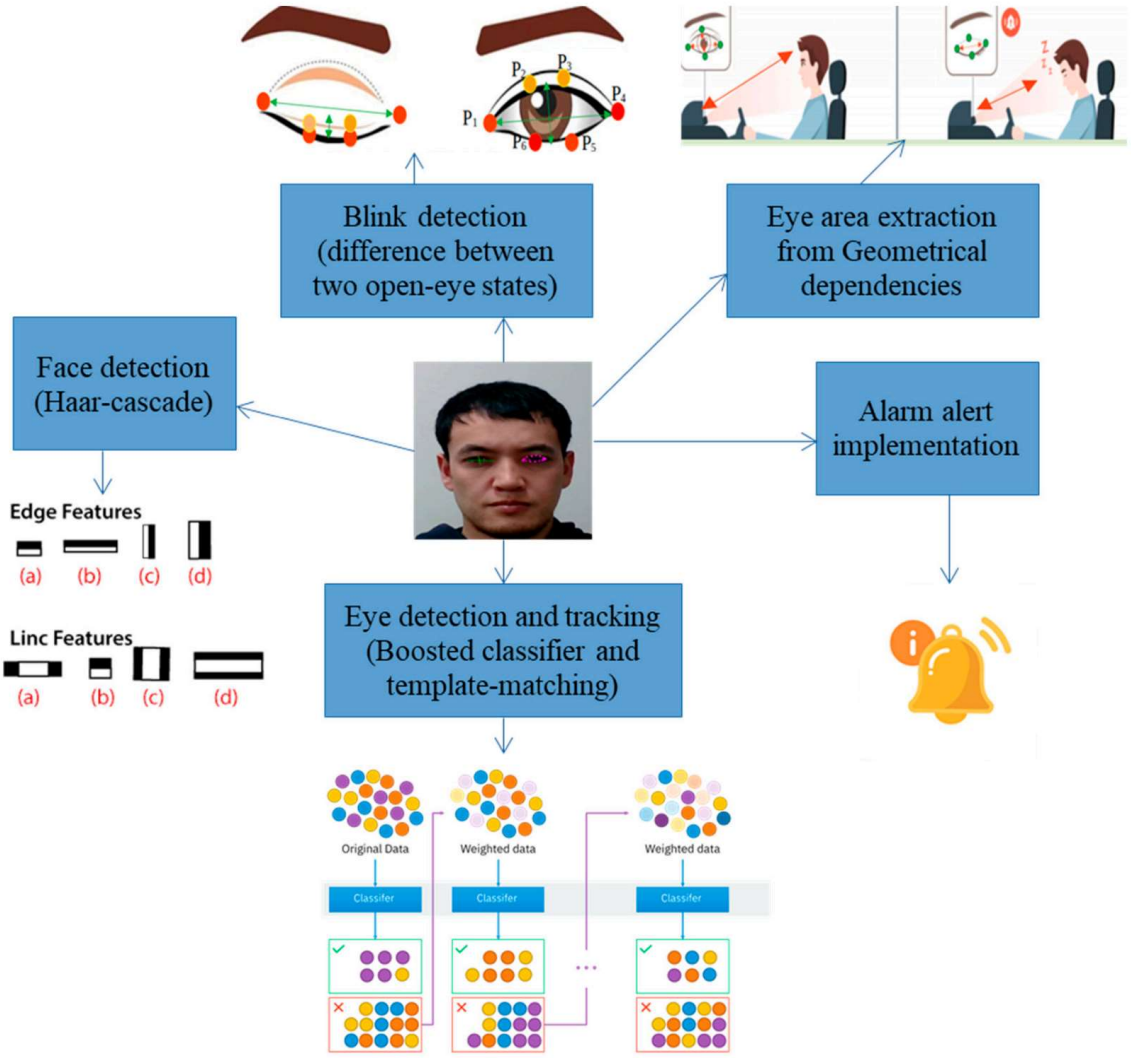
First, the code specifies the path to the Haar cascade classifier for detecting faces. The classifier is loaded using the cv2.CascadeClassifier method, which reads the XML file containing the pre-trained model. In this case, haarcascade_frontalface_default.xml is used to detect faces.

Next, the code attempts to open a webcam for capturing live video. It tries to access the default webcam (cap = cv2.VideoCapture(1)), and if that fails, it attempts to access an alternative webcam (cap = cv2.VideoCapture(0)). If the webcam cannot be opened, an IO Error is raised, indicating that the webcam cannot be accessed.

Once the webcam is successfully accessed, the code enters a continuous loop to read frames from the webcam in real-time. Each frame captured from the webcam is processed to detect faces and eyes. The eye detection is performed using another Haar cascade classifier, haarcascade_eye.xml, which is specifically trained to detect eyes.

Within the loop, the captured frame is converted to a grayscale image using cv2.cvtColor. Grayscale conversion simplifies the image data, reducing computational complexity and enhancing the performance of the detection algorithms. The grayscale image is then processed to detect eyes using the detect Multiscale method of the eye cascade classifier. This method returns a list of rectangles where it believes it has found eyes in the image. Each rectangle is defined by its top-left corner coordinates (x, y) and its width and height (w, h).

For each detected eye, a Region of Interest (ROI) is created both in the grayscale image and the color image. This ROI focuses on the detected eye region for further processing. A rectangle is drawn around the detected eye in the color image to visualize the detection.

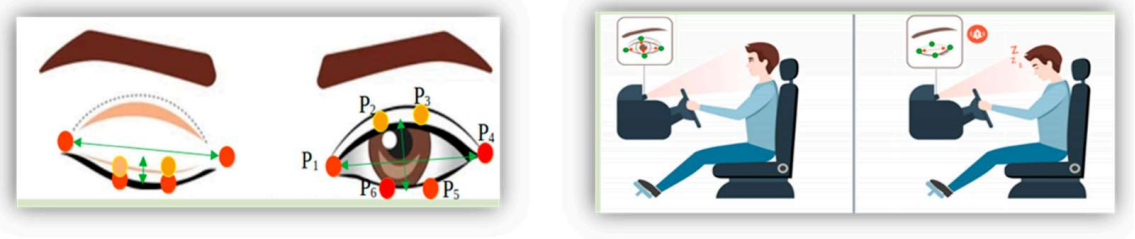


To further refine the detection and avoid false positives, the code performs a second round of eye detection within the eye ROIs. If no eyes are detected in this refined detection step, a message "eyes are not detected" is printed. However, if eyes are detected, their ROIs are extracted and resized to 224x224 pixels. The resizing standardizes the input size for the neural network model that will classify the eye state.

The resized eye image is then prepared for prediction by expanding its dimensions to match the input shape expected by the neural network and normalizing its pixel values to the range [0, 1]. This normalized image is fed into a pre-trained neural network model, newmodel, which predicts whether the eyes are open or closed. The model outputs a prediction score, which is compared to a threshold to determine the eye state. If the prediction score exceeds the threshold, the status is set to "Open Eyes"; otherwise, it is set to "Closed Eyes".

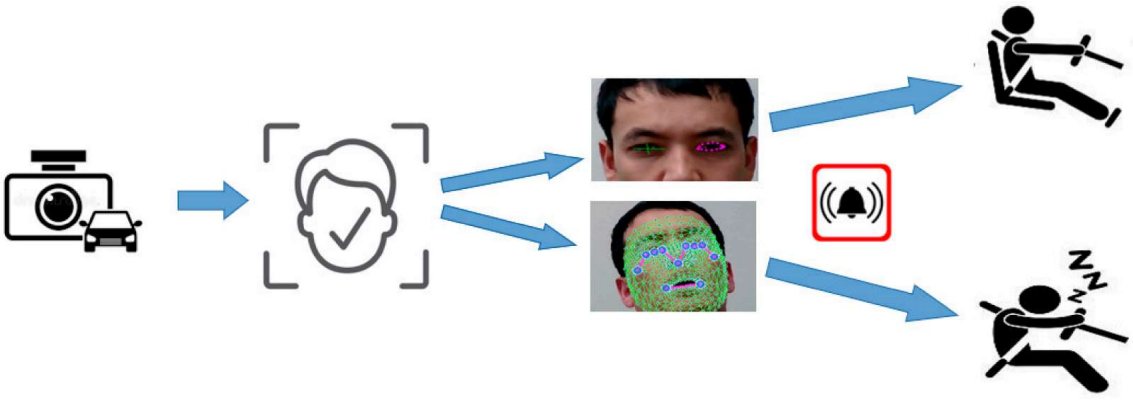
After processing the eyes, the code proceeds to detect faces in the grayscale frame using the face cascade classifier. Similar to eye detection, the detectMultiScale method is used to find faces, and rectangles are drawn around detected faces in the color frame for visualization.

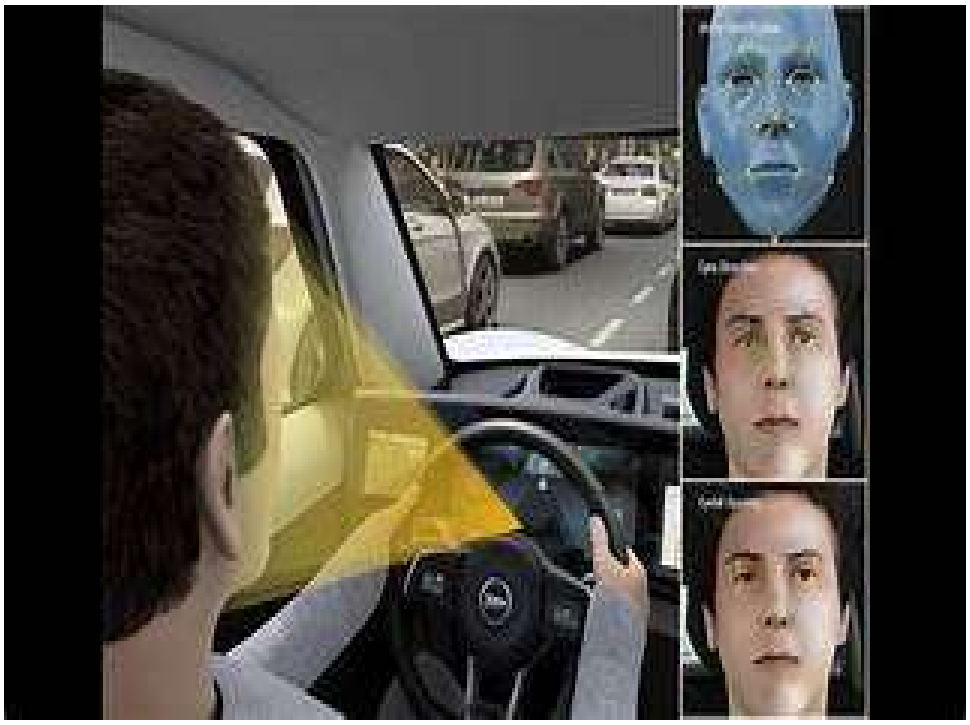
The status of the eyes (open or closed) is then overlaid on the video frame using the cv2.putText method. This method places the text on the frame at specified coordinates, using a specific font, color, and thickness. In this case, the text is placed at coordinates (50, 50) using the cv2.FONT_HERSHEY_SIMPLEX font, with a red color and a thickness of 2.



(a) (b)

Finally, the processed frame, with rectangles around faces and eyes and the status text, is displayed in a window titled "Drowsiness Detection" using the cv2.imshow method. The window continuously updates with new frames from the webcam.





Key Components of the Code:

- Initialization:

- Load the Haar cascade classifiers for face and eye detection.
- Attempt to access the webcam for video capture.

- Frame Capture and Processing:

- Continuously capture frames from the webcam.
- Convert frames to grayscale for simplified processing.
- Detect faces and eyes using the pre-trained classifiers.
-Draw rectangles around detected faces and eyes for visualization.

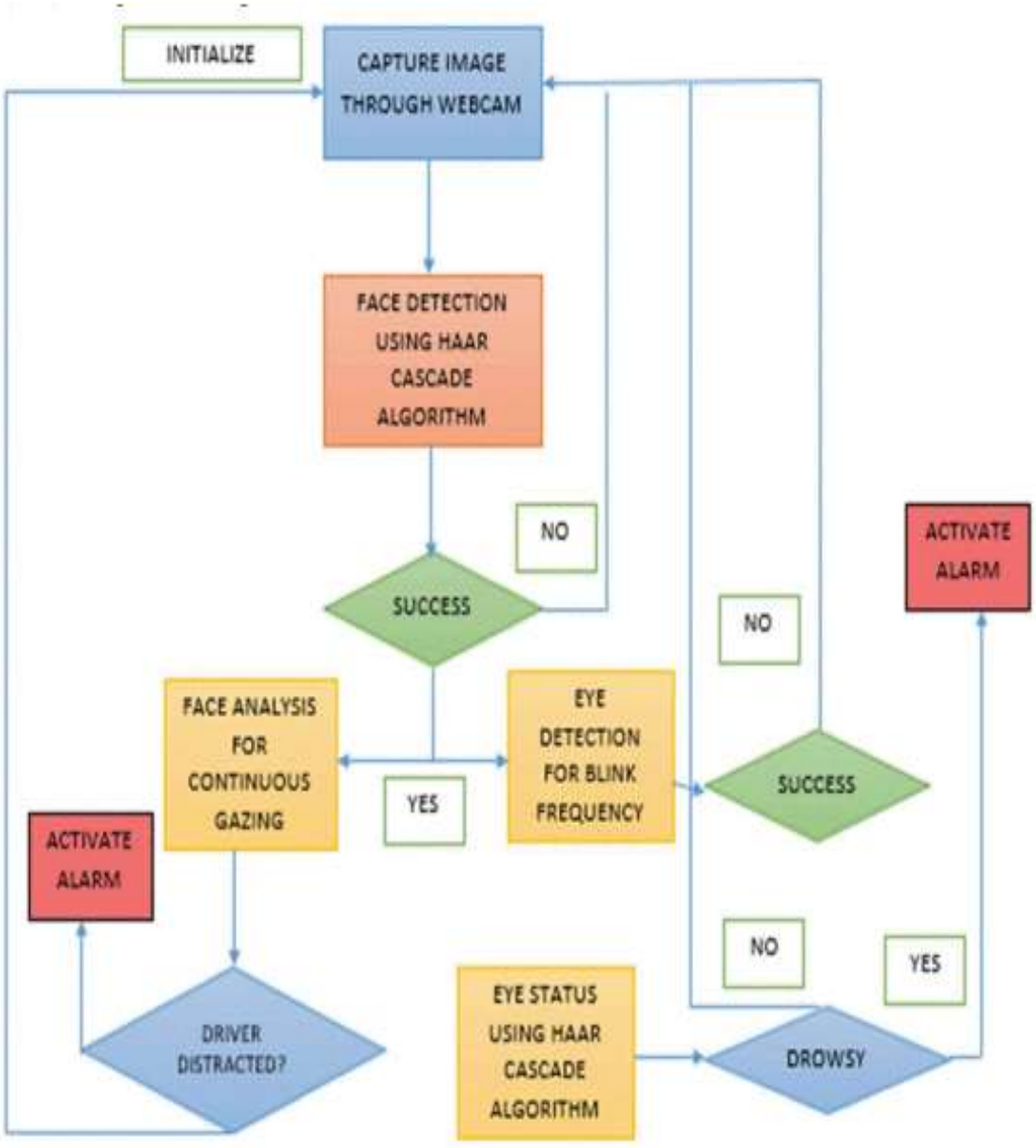
- Eye State Prediction:

- Extract ROIs for detected eyes.
- Resize and normalize the eye ROIs.
- Predict the eye state using a pre-trained neural network model.
- Determine and display the eye state (open or closed).

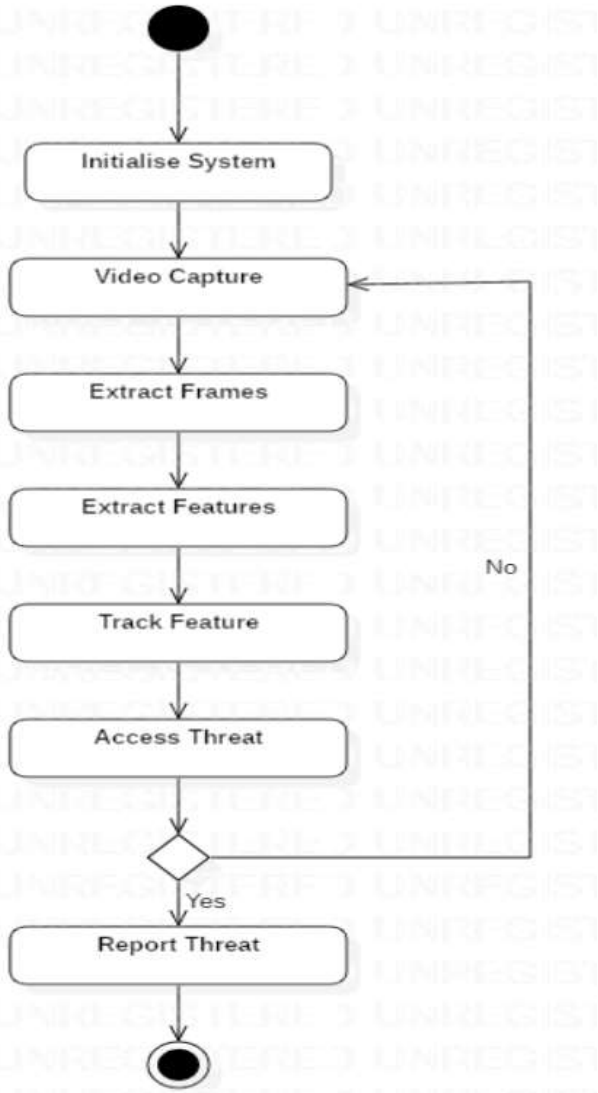
- Visualization and User Interaction:

- Overlay the eye state text on the video frame.
- Display the processed video frame in a window.
- Continuously update the window with new frames until the user presses 'q'.

Flowchart


INITIALIZE
CAPTURE IMAGE THROUGH WEBCAM
FACE DETECTION USING HAAR CASCADE ALGORITHM
NO
SUCCESS
FACE ANALYSIS FOR CONTINUOUS GAZING
YES
EYE DETECTION FOR BLINK FREQUENCY
NO
SUCCESS
ACTIVATE ALARM
ACTIVATE ALARM
DRIVER DISTRACTED?
NO
YES
EYE STATUS USING HAAR CASCADE ALGORITHM
DROWSY

Activity Diagram

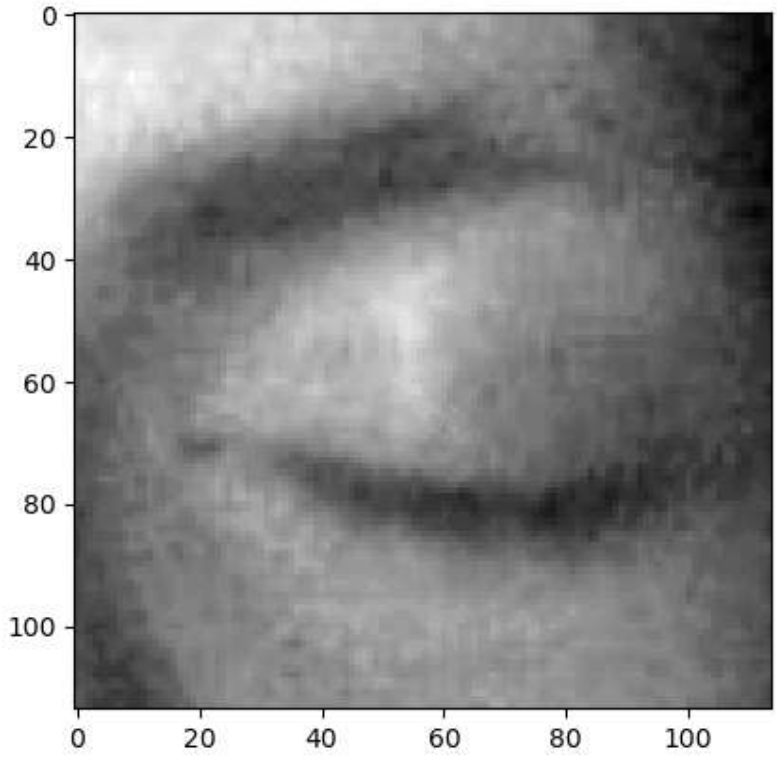


Code:

```
import tensorflow as tf
import cv2
import os
import matplotlib.pyplot as plt
import numpy as np
```

```
img_array=cv2.imread(r"C:\Users\91900\Desktop\project\Test_Dataset\Closed_Eyes\s0002_00001_0_0_0_0_0_01.png",cv2.IMREAD_GRAYSCALE)
```

```
plt.imshow(img_array,cmap="gray")
```

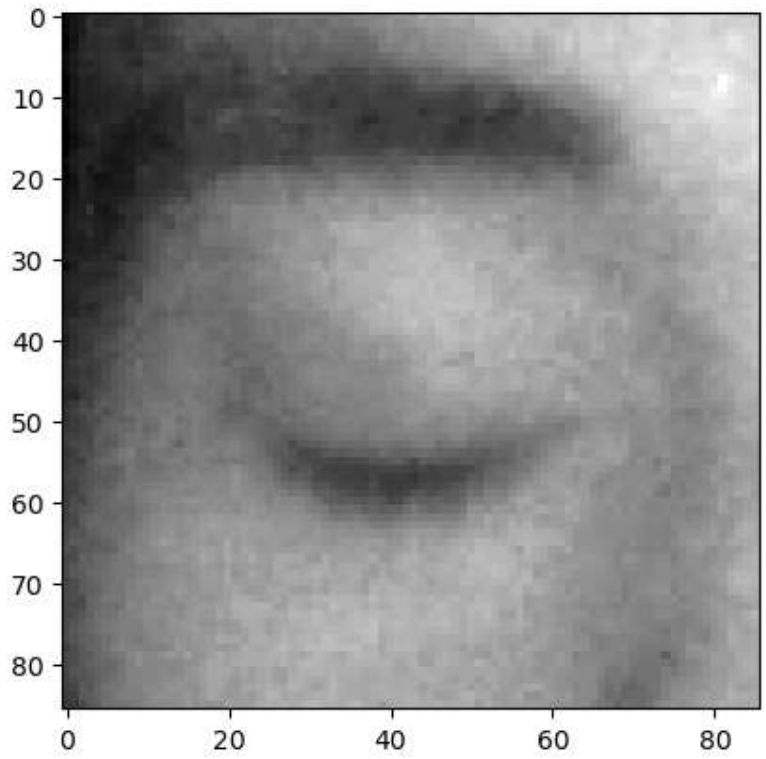



```
img_array.shape
```

```
(114, 114)
```

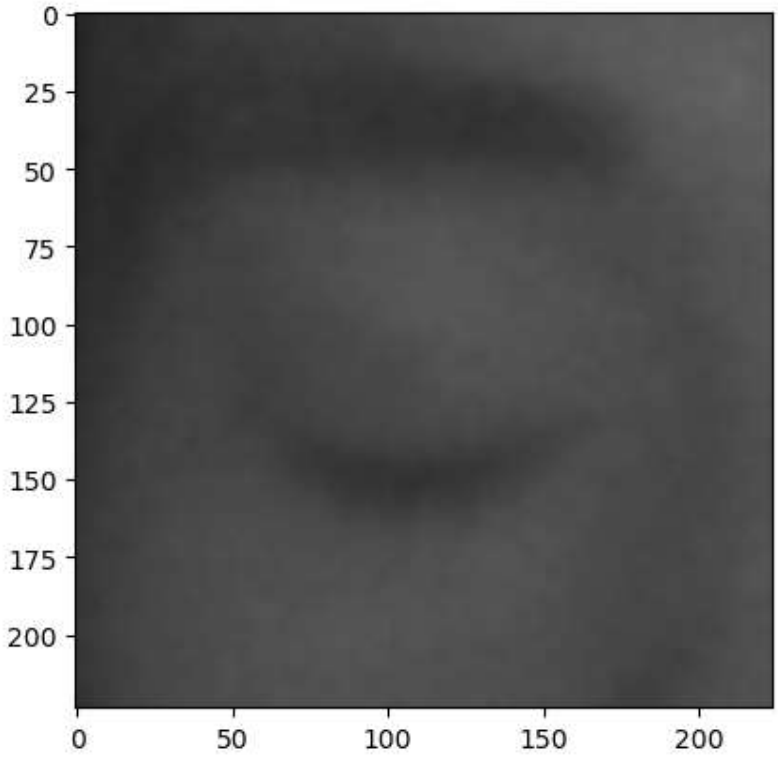
```
Datadirectory=r"C:\Users\91900\Desktop\project\Train_Dataset"
```

```
Classes =["Closed_Eyes","Open_Eyes"]
for category in Classes:
      path=os.path.join(Datadirectory,category)
      for img in os.listdir(path):
      img_array =
cv2.imread(os.path.join(path,img),cv2.IMREAD_GRAYSCALE)
      backtorgb = cv2.cvtColor(img_array,cv2.COLOR_GRAY2RGB)
      plt.imshow(img_array,cmap="gray")
      plt.show()
      break
      break
```




```
img_size=224

new_array=cv2.resize(backtorgb,(img_size,img_size))
plt.imshow(new_array,cmap="gray")
plt.show()
```




```
training_Data=[]

def create_training_Data():
      for category in Classes:
      path=os.path.join(Datadirectory,category)
      class_num=Classes.index(category)
      for img in os.listdir(path):
            try:

img_array=cv2.imread(os.path.join(path,img),cv2.IMREAD_GRAYSCALE)
                  backtorgb=cv2.cvtColor(img_array,cv2.COLOR_GRAY2RGB)
                  new_array=cv2.resize(backtorgb,(img_size,img_size))
                  training_Data.append([new_array,class_num])
            except Exception as e:
                  pass
```

```
create_training_Data()
print(len(training_Data))
```

```
2061
```

```
import random
```

```
random.shuffle(training_Data)
```

```
X=[]
y=[]

for features,label in training_Data:
      X.append(features)
      y.append(label)

X=np.array(X).reshape(-1,img_size,img_size,3)
```

```
X.shape
```

```
(2061, 224, 224, 3)
```

```
X=X/255.0;
```

```
Y=np.array(y)
```

```
import pickle

pickle_out=open("X.pickle","wb")
pickle.dump(X,pickle_out)
pickle_out.close()

pickle_out=open("y.pickle","wb")
pickle.dump(y,pickle_out)
pickle_out.close()
```

```
pickle_in=open("X.pickle","rb")
X=pickle.load(pickle_in)

pickle_in=open("y.pickle","rb")
y=pickle.load(pickle_in)
```

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

```
model = tf.keras.applications.mobilenet.MobileNet()
```

```
model.summary()
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 3)	0
conv1 (Conv2D)	(None, 112, 112, 32)	864
conv1_bn (BatchNormalization)	(None, 112, 112, 32)	128
conv1_relu (ReLU)	(None, 112, 112, 32)	0
conv_dw_1 (DepthwiseConv2D)	(None, 112, 112, 32)	288
conv_dw_1_bn (BatchNormalization)	(None, 112, 112, 32)	128
conv_dw_1_relu (ReLU)	(None, 112, 112, 32)	0
conv_pw_1 (Conv2D)	(None, 112, 112, 64)	2,048
conv_pw_1_bn (BatchNormalization)	(None, 112, 112, 64)	256
conv_pw_1_relu (ReLU)	(None, 112, 112, 64)	0
conv_pad_2 (ZeroPadding2D)	(None, 113, 113, 64)	0
conv_dw_2 (DepthwiseConv2D)	(None, 56, 56, 64)	576
conv_dw_2_bn (BatchNormalization)	(None, 56, 56, 64)	256

conv_dw_2_relu (ReLU)	(None, 56, 56, 64)	0
conv_pw_2 (Conv2D)	(None, 56, 56, 128)	8,192
conv_pw_2_bn (BatchNormalization)	(None, 56, 56, 128)	512
conv_pw_2_relu (ReLU)	(None, 56, 56, 128)	0
conv_dw_3 (DepthwiseConv2D)	(None, 56, 56, 128)	1,152
conv_dw_3_bn (BatchNormalization)	(None, 56, 56, 128)	512
conv_dw_3_relu (ReLU)	(None, 56, 56, 128)	0
conv_pw_3 (Conv2D)	(None, 56, 56, 128)	16,384
conv_pw_3_bn (BatchNormalization)	(None, 56, 56, 128)	512
conv_pw_3_relu (ReLU)	(None, 56, 56, 128)	0
conv_pad_4 (ZeroPadding2D)	(None, 57, 57, 128)	0
conv_dw_4 (DepthwiseConv2D)	(None, 28, 28, 128)	1,152
conv_dw_4_bn (BatchNormalization)	(None, 28, 28, 128)	512
conv_dw_4_relu (ReLU)	(None, 28, 28, 128)	0

conv_pw_4 (Conv2D)	(None, 28, 28, 256)	32,768
conv_pw_4_bn (BatchNormalization)	(None, 28, 28, 256)	1,024
conv_pw_4_relu (ReLU)	(None, 28, 28, 256)	0
conv_dw_5 (DepthwiseConv2D)	(None, 28, 28, 256)	2,304
conv_dw_5_bn (BatchNormalization)	(None, 28, 28, 256)	1,024
conv_dw_5_relu (ReLU)	(None, 28, 28, 256)	0
conv_pw_5 (Conv2D)	(None, 28, 28, 256)	65,536
conv_pw_5_bn (BatchNormalization)	(None, 28, 28, 256)	1,024
conv_pw_5_relu (ReLU)	(None, 28, 28, 256)	0
conv_pad_6 (ZeroPadding2D)	(None, 29, 29, 256)	0
conv_dw_6 (DepthwiseConv2D)	(None, 14, 14, 256)	2,304
conv_dw_6_bn (BatchNormalization)	(None, 14, 14, 256)	1,024
conv_dw_6_relu (ReLU)	(None, 14, 14, 256)	0
conv_pw_6 (Conv2D)	(None, 14, 14, 512)	131,072

conv_pw_6_bn (BatchNormalization)	(None, 14, 14, 512)	2,048
conv_pw_6_relu (ReLU)	(None, 14, 14, 512)	0
conv_dw_7 (DepthwiseConv2D)	(None, 14, 14, 512)	4,608
conv_dw_7_bn (BatchNormalization)	(None, 14, 14, 512)	2,048
conv_dw_7_relu (ReLU)	(None, 14, 14, 512)	0
conv_pw_7 (Conv2D)	(None, 14, 14, 512)	262,144
conv_pw_7_bn (BatchNormalization)	(None, 14, 14, 512)	2,048
conv_pw_7_relu (ReLU)	(None, 14, 14, 512)	0
conv_dw_8 (DepthwiseConv2D)	(None, 14, 14, 512)	4,608
conv_dw_8_bn (BatchNormalization)	(None, 14, 14, 512)	2,048
conv_dw_8_relu (ReLU)	(None, 14, 14, 512)	0
conv_pw_8 (Conv2D)	(None, 14, 14, 512)	262,144
conv_pw_8_bn (BatchNormalization)	(None, 14, 14, 512)	2,048
conv_pw_8_relu (ReLU)	(None, 14, 14, 512)	0

conv_dw_9 (DepthwiseConv2D)	(None, 14, 14, 512)	4,608
conv_dw_9_bn (BatchNormalization)	(None, 14, 14, 512)	2,048
conv_dw_9_relu (ReLU)	(None, 14, 14, 512)	0
conv_pw_9 (Conv2D)	(None, 14, 14, 512)	262,144
conv_pw_9_bn (BatchNormalization)	(None, 14, 14, 512)	2,048
conv_pw_9_relu (ReLU)	(None, 14, 14, 512)	0
conv_dw_10 (DepthwiseConv2D)	(None, 14, 14, 512)	4,608
conv_dw_10_bn (BatchNormalization)	(None, 14, 14, 512)	2,048
conv_dw_10_relu (ReLU)	(None, 14, 14, 512)	0
conv_pw_10 (Conv2D)	(None, 14, 14, 512)	262,144
conv_pw_10_bn (BatchNormalization)	(None, 14, 14, 512)	2,048
conv_pw_10_relu (ReLU)	(None, 14, 14, 512)	0
conv_dw_11 (DepthwiseConv2D)	(None, 14, 14, 512)	4,608
conv_dw_11_bn (BatchNormalization)	(None, 14, 14, 512)	2,048

conv_dw_11_relu (ReLU)	(None, 14, 14, 512)	0
conv_pw_11 (Conv2D)	(None, 14, 14, 512)	262,144
conv_pw_11_bn (BatchNormalization)	(None, 14, 14, 512)	2,048
conv_pw_11_relu (ReLU)	(None, 14, 14, 512)	0
conv_pad_12 (ZeroPadding2D)	(None, 15, 15, 512)	0
conv_dw_12 (DepthwiseConv2D)	(None, 7, 7, 512)	4,608
conv_dw_12_bn (BatchNormalization)	(None, 7, 7, 512)	2,048
conv_dw_12_relu (ReLU)	(None, 7, 7, 512)	0
conv_pw_12 (Conv2D)	(None, 7, 7, 1024)	524,288
conv_pw_12_bn (BatchNormalization)	(None, 7, 7, 1024)	4,096
conv_pw_12_relu (ReLU)	(None, 7, 7, 1024)	0
conv_dw_13 (DepthwiseConv2D)	(None, 7, 7, 1024)	9,216
conv_dw_13_bn (BatchNormalization)	(None, 7, 7, 1024)	4,096
conv_dw_13_relu (ReLU)	(None, 7, 7, 1024)	0

conv_pw_13 (Conv2D)	(None, 7, 7, 1024)	1,048,576
conv_pw_13_bn (BatchNormalization)	(None, 7, 7, 1024)	4,096
conv_pw_13_relu (ReLU)	(None, 7, 7, 1024)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1, 1, 1024)	0
dropout (Dropout)	(None, 1, 1, 1024)	0
conv_preds (Conv2D)	(None, 1, 1, 1000)	1,025,000
reshape_2 (Reshape)	(None, 1000)	0
predictions (Activation)	(None, 1000)	0

Total params: 4,253,864 (16.23 MB)

Trainable params: 4,231,976 (16.14 MB)

Non-trainable params: 21,888 (85.50 KB)

```
base_input = model.layers[1].input
```

```
base_output = model.layers[-4].output
```

```
Flat_layer= layers.Flatten()(base_output)
final_output = layers.Dense(1)(Flat_layer)
final_output = layers.Activation('sigmoid')(final_output)
```

```
new_model = keras.Model(inputs = base_input,outputs= final_output)
```

```
new_model.summary()
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 3)	0
conv1 (Conv2D)	(None, 112, 112, 32)	864
conv1_bn (BatchNormalization)	(None, 112, 112, 32)	128
conv1_relu (ReLU)	(None, 112, 112, 32)	0
conv_dw_1 (DepthwiseConv2D)	(None, 112, 112, 32)	288
conv_dw_1_bn (BatchNormalization)	(None, 112, 112, 32)	128
conv_dw_1_relu (ReLU)	(None, 112, 112, 32)	0
conv_pw_1 (Conv2D)	(None, 112, 112, 64)	2,048
conv_pw_1_bn (BatchNormalization)	(None, 112, 112, 64)	256
conv_pw_1_relu (ReLU)	(None, 112, 112, 64)	0

conv_pad_2 (ZeroPadding2D)	(None, 113, 113, 64)	0
conv_dw_2 (DepthwiseConv2D)	(None, 56, 56, 64)	576
conv_dw_2_bn (BatchNormalization)	(None, 56, 56, 64)	256
conv_dw_2_relu (ReLU)	(None, 56, 56, 64)	0
conv_pw_2 (Conv2D)	(None, 56, 56, 128)	8,192
conv_pw_2_bn (BatchNormalization)	(None, 56, 56, 128)	512
conv_pw_2_relu (ReLU)	(None, 56, 56, 128)	0
conv_dw_3 (DepthwiseConv2D)	(None, 56, 56, 128)	1,152
conv_dw_3_bn (BatchNormalization)	(None, 56, 56, 128)	512
conv_dw_3_relu (ReLU)	(None, 56, 56, 128)	0
conv_pw_3 (Conv2D)	(None, 56, 56, 128)	16,384
conv_pw_3_bn (BatchNormalization)	(None, 56, 56, 128)	512
conv_pw_3_relu (ReLU)	(None, 56, 56, 128)	0
conv_pad_4 (ZeroPadding2D)	(None, 57, 57, 128)	0

conv_dw_4 (DepthwiseConv2D)	(None, 28, 28, 128)	1,152
conv_dw_4_bn (BatchNormalization)	(None, 28, 28, 128)	512
conv_dw_4_relu (ReLU)	(None, 28, 28, 128)	0
conv_pw_4 (Conv2D)	(None, 28, 28, 256)	32,768
conv_pw_4_bn (BatchNormalization)	(None, 28, 28, 256)	1,024
conv_pw_4_relu (ReLU)	(None, 28, 28, 256)	0
conv_dw_5 (DepthwiseConv2D)	(None, 28, 28, 256)	2,304
conv_dw_5_bn (BatchNormalization)	(None, 28, 28, 256)	1,024
conv_dw_5_relu (ReLU)	(None, 28, 28, 256)	0
conv_pw_5 (Conv2D)	(None, 28, 28, 256)	65,536
conv_pw_5_bn (BatchNormalization)	(None, 28, 28, 256)	1,024
conv_pw_5_relu (ReLU)	(None, 28, 28, 256)	0
conv_pad_6 (ZeroPadding2D)	(None, 29, 29, 256)	0
conv_dw_6 (DepthwiseConv2D)	(None, 14, 14, 256)	2,304

conv_dw_6_bn (BatchNormalization)	(None, 14, 14, 256)	1,024
conv_dw_6_relu (ReLU)	(None, 14, 14, 256)	0
conv_pw_6 (Conv2D)	(None, 14, 14, 512)	131,072
conv_pw_6_bn (BatchNormalization)	(None, 14, 14, 512)	2,048
conv_pw_6_relu (ReLU)	(None, 14, 14, 512)	0
conv_dw_7 (DepthwiseConv2D)	(None, 14, 14, 512)	4,608
conv_dw_7_bn (BatchNormalization)	(None, 14, 14, 512)	2,048
conv_dw_7_relu (ReLU)	(None, 14, 14, 512)	0
conv_pw_7 (Conv2D)	(None, 14, 14, 512)	262,144
conv_pw_7_bn (BatchNormalization)	(None, 14, 14, 512)	2,048
conv_pw_7_relu (ReLU)	(None, 14, 14, 512)	0
conv_dw_8 (DepthwiseConv2D)	(None, 14, 14, 512)	4,608
conv_dw_8_bn (BatchNormalization)	(None, 14, 14, 512)	2,048
conv_dw_8_relu (ReLU)	(None, 14, 14, 512)	0

```
┼──────────┼
│ conv_pw_8 (Conv2D)                  │ (None, 14, 14, 512)           │
262,144 │
├─────────────────────────────────────┼───────────────────────────────┼──
┼──────────┼
│ conv_pw_8_bn (BatchNormalization)   │ (None, 14, 14, 512)           │
     2,048 │
├─────────────────────────────────────┼───────────────────────────────┼──
┼──────────┼
│ conv_pw_8_relu (ReLU)               │ (None, 14, 14, 512)           │
     0 │
├─────────────────────────────────────┼───────────────────────────────┼──
┼──────────┼
│ conv_dw_9 (DepthwiseConv2D)         │ (None, 14, 14, 512)           │
     4,608 │
├─────────────────────────────────────┼───────────────────────────────┼──
┼──────────┼
│ conv_dw_9_bn (BatchNormalization)   │ (None, 14, 14, 512)           │
     2,048 │
├─────────────────────────────────────┼───────────────────────────────┼──
┼──────────┼
│ conv_dw_9_relu (ReLU)               │ (None, 14, 14, 512)           │
     0 │
├─────────────────────────────────────┼───────────────────────────────┼──
┼──────────┼
│ conv_pw_9 (Conv2D)                  │ (None, 14, 14, 512)           │
262,144 │
├─────────────────────────────────────┼───────────────────────────────┼──
┼──────────┼
│ conv_pw_9_bn (BatchNormalization)   │ (None, 14, 14, 512)           │
     2,048 │
├─────────────────────────────────────┼───────────────────────────────┼──
┼──────────┼
│ conv_pw_9_relu (ReLU)               │ (None, 14, 14, 512)           │
     0 │
├─────────────────────────────────────┼───────────────────────────────┼──
┼──────────┼
│ conv_dw_10 (DepthwiseConv2D)        │ (None, 14, 14, 512)           │
     4,608 │
├─────────────────────────────────────┼───────────────────────────────┼──
┼──────────┼
│ conv_dw_10_bn (BatchNormalization)  │ (None, 14, 14, 512)           │
     2,048 │
├─────────────────────────────────────┼───────────────────────────────┼──
┼──────────┼
│ conv_dw_10_relu (ReLU)              │ (None, 14, 14, 512)           │
     0 │
├─────────────────────────────────────┼───────────────────────────────┼──
┼──────────┼
│ conv_pw_10 (Conv2D)                 │ (None, 14, 14, 512)           │
262,144 │
├─────────────────────────────────────┼───────────────────────────────┼──
┼──────────┼
│ conv_pw_10_bn (BatchNormalization)  │ (None, 14, 14, 512)           │
     2,048 │
├─────────────────────────────────────┼───────────────────────────────┼──
```

conv_pw_10_relu (ReLU)	(None, 14, 14, 512)	0
conv_dw_11 (DepthwiseConv2D)	(None, 14, 14, 512)	4,608
conv_dw_11_bn (BatchNormalization)	(None, 14, 14, 512)	2,048
conv_dw_11_relu (ReLU)	(None, 14, 14, 512)	0
conv_pw_11 (Conv2D)	(None, 14, 14, 512)	262,144
conv_pw_11_bn (BatchNormalization)	(None, 14, 14, 512)	2,048
conv_pw_11_relu (ReLU)	(None, 14, 14, 512)	0
conv_pad_12 (ZeroPadding2D)	(None, 15, 15, 512)	0
conv_dw_12 (DepthwiseConv2D)	(None, 7, 7, 512)	4,608
conv_dw_12_bn (BatchNormalization)	(None, 7, 7, 512)	2,048
conv_dw_12_relu (ReLU)	(None, 7, 7, 512)	0
conv_pw_12 (Conv2D)	(None, 7, 7, 1024)	524,288
conv_pw_12_bn (BatchNormalization)	(None, 7, 7, 1024)	4,096
conv_pw_12_relu (ReLU)	(None, 7, 7, 1024)	0

conv_dw_13 (DepthwiseConv2D)	(None, 7, 7, 1024)	9,216
conv_dw_13_bn (BatchNormalization)	(None, 7, 7, 1024)	4,096
conv_dw_13_relu (ReLU)	(None, 7, 7, 1024)	0
conv_pw_13 (Conv2D)	(None, 7, 7, 1024)	1,048,576
conv_pw_13_bn (BatchNormalization)	(None, 7, 7, 1024)	4,096
conv_pw_13_relu (ReLU)	(None, 7, 7, 1024)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1, 1, 1024)	0
dropout (Dropout)	(None, 1, 1, 1024)	0
flatten_4 (Flatten)	(None, 1024)	0
dense_4 (Dense)	(None, 1)	1,025
activation_4 (Activation)	(None, 1)	0

Total params: 3,229,889 (12.32 MB)

Trainable params: 3,208,001 (12.24 MB)

Non-trainable params: 21,888 (85.50 KB)

%% Total parameters are reduced

```
new_model.compile(loss="binary_crossentropy",optimizer = "adam",
metrics = ["accuracy"])
```

```
new_model.fit(X,Y,epochs=1,validation_split=0.1)
```

58/58 ━━━━━━━━━━━━━━━━━━━━ **130s** 2s/step - accuracy: 0.9954 - loss: 0.0151 - val_accuracy: 0.9324 - val_loss: 0.4023

```
new_model.save('my_model-Drowsiness.keras')
```

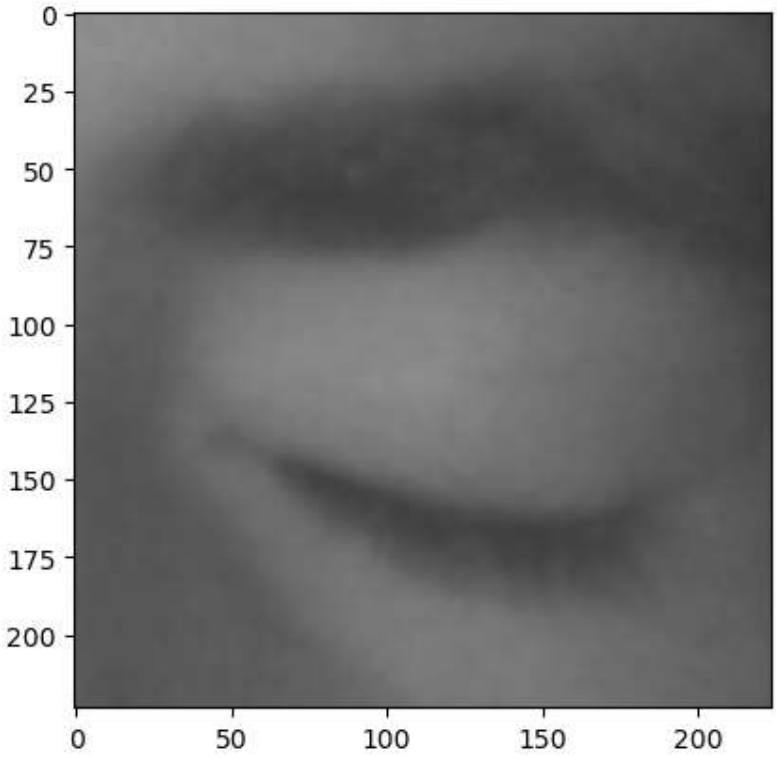
```
new_model =
tf.keras.models.load_model(r'C:\Users\91900\Desktop\project\my_model-Drowsiness.keras')
```

```
img_array =
cv2.imread(r'C:\Users\91900\Desktop\project\s0012_00003_0_0_0_0_1_01.png',cv2.IMREAD_GRAYSCALE)
backtorgb = cv2.cvtColor(img_array,cv2.COLOR_GRAY2RGB)
new_array= cv2.resize(backtorgb,(img_size,img_size))
```

```
X_input = np.array(new_array).reshape(1,img_size,img_size,3)
X_input.shape
```

```
(1, 224, 224, 3)
```

```
plt.imshow(new_array)
```




```
X_input = X_input/255.0
```

```
prediction = new_model.predict(X_input)
```

```
prediction
```

array([[0.00115863]], dtype=float32)

%% Result = 0 i.e., the eyes are closed.

Testing the Model by giving a random image as input

%% Load the Image

```
img = cv2.imread(r'C:\Users\91900\Downloads\sad_face_women.png')
```

%% show the Image

```
plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
```

%% Face cascading to detect eyes

```
facecascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
```

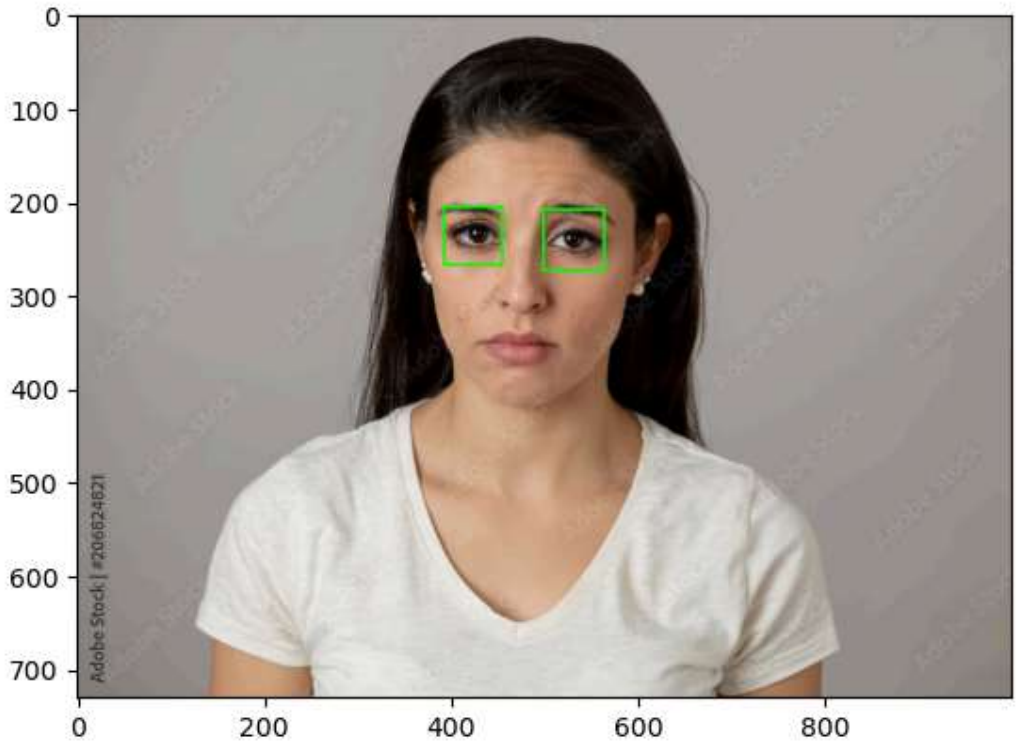
```
eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_eye.xml')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
```

```
eyes= eye_cascade.detectMultiScale(gray,1.1,4)
```

```
for(x,y,w,h) in eyes:
      cv2.rectangle(img,(x,y),(x+w,y+w),(0,255,0),2)
```

%% plot after eyes detection on the face

```
plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
```



%% To identify the eye dimensions

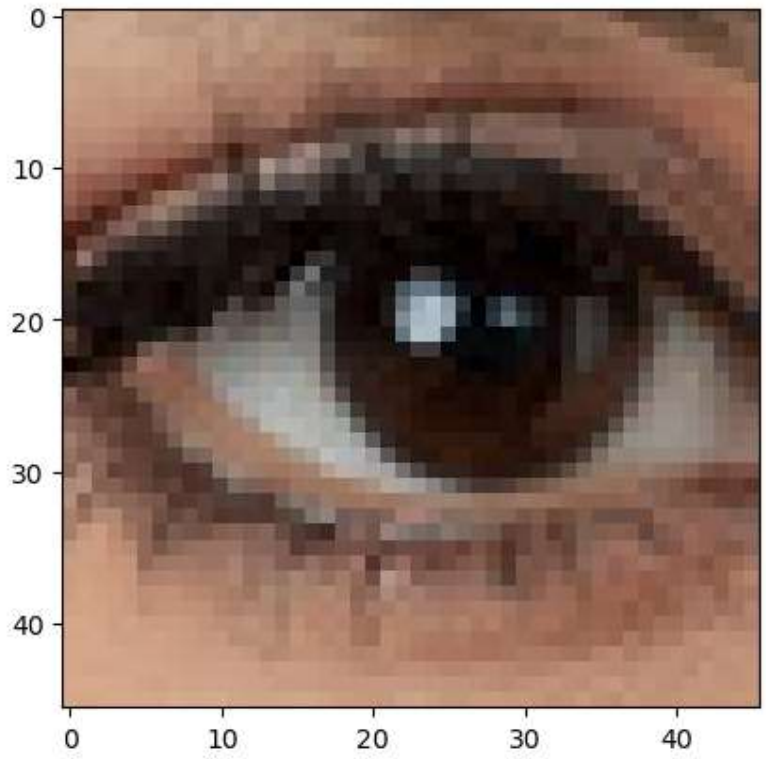
```
eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_eye.xml')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

eyes= eye_cascade.detectMultiScale(gray,1.1,4)
for x,y,w,h in eyes:
      roi_gray = gray[y:y+h, x:x+w]
      roi_color = img[y:y+h, x:x+w]
```

```
    eyess=eye_cascade.detectMultiScale(roi_gray)
    if len(eyess)==0:
    print("eyes are not detected")
    else:
    for(ex,ey,ew,eh) in eyess:
         eyes_roi = roi_color[ey:ey+eh, ex: ex+ew]
```

%% Plot

```
plt.imshow(cv2.cvtColor(eyes_roi,cv2.COLOR_BGR2RGB))
```



%% shape of the eyes

```
eyes_roi.shape
```

```
(46, 46, 3)
```

```
final_image = cv2.resize(eyes_roi,(224,224))
final_image = np.expand_dims(final_image,axis=0)
final_image = final_image/255.0
```

```
%% Shape of the final image
```

```
final_image.shape
```

```
(1, 224, 224, 3)
```

%% Predicting the result:

```
new_model.predict(final_image)
```

```
array([[0.999291]], dtype=float32)
```

Result:

The result shows that the final predicted value is close to 1 i.e., the person's eyes are opened.

Conclusion

The Drowsiness Detection and Alert System using Deep Learning is a significant step forward in improving road safety. By employing advanced machine learning and computer vision techniques, the system effectively monitors the driver's eyes and facial expressions in real-time to detect signs of drowsiness. This project involves thorough data preprocessing, including collection, cleaning, face and eye detection, resizing, normalisation, and augmentation, ensuring the model works efficiently and accurately across various scenarios.

The system's non-intrusive design allows seamless integration into different vehicles, making it practical and cost-effective for widespread use. It provides multimodal alerts—visual, auditory, and haptic—prompting drivers when drowsiness is detected, thereby helping to prevent accidents caused by fatigue.

Overall, this system not only aims to reduce road accidents but also encourages responsible and alert driving. It demonstrates the potential of deep learning and computer vision in creating effective real-world safety solutions. Future enhancements could include additional features like head pose estimation and broader facial expression analysis to further improve its accuracy and reliability.

References

1. Real-Time Deep Learning-Based Drowsiness Detection: Leveraging Computer-Vision and Eye-Blink Analyses for Enhanced Road Safety
2. Drowsiness Detection System Using Deep Learning Methods, August 2022, International Journal of Advanced Research in Science Communication and Technology
3. Driver drowsiness detection and smart alerting using deep learning and IoT
4. Deep Learning Based Drowsiness Detection with Alert System Using Raspberry Pi Pico
5. A Review of Recent Developments in Driver Drowsiness Detection Systems
6. Study on Drowsiness Detection System Using Deep Learning