DROWSINESS DETECTION

A Project Report

Submitted by

M. Yasawini Sai	*20221BCA0044*
E. Kushal	*20221BCA0053*
Y. Gnana Pavani	*20221BCA0060*
Velpula Usha	*20221BCA0185*

Under the guidance of

Mr. Sakthi S

Assistant Professor, Presidency School of Computer Science Engineering

in partial fulfillment for the award of the degree

of

BACHELOR OF COMPUTER APPLICATIONS

At

GAIN MORE KNOWLEDGE
REACH GREATER HEIGHTS

SCHOOL OF INFORMATION SCIENCE

PRESIDENCY UNIVERSITY

BENGALURU

MAY 2025

BACHELOR OF COMPUTER APPLICATIONS

SCHOOL OF INFORMATION SCIENCE

PRESIDENCY UNIVERSITY

GAIN MORE KNOWLEDGE
REACH GREATER HEIGHTS

CERTIFICATE

This is to certified that the University Project Report **"Drowsiness Detection"** being submitted by ***M. Yasawini Sai, E. Kushal, Y. Gnana Pavani, Velpula Usha*** bearing roll number ***20221BCA0044, 20221BCA0053, 20221BCA0060, 20221BCA00185*** in partial fulfillment of requirement for the award of degree of **Bachelor of Computer Applications** is a bonafide work carried out under my supervision.

Mr. Sakthi S.
Assistant Professor,
Presidency School of CSE&IS
Presidency University

Dr. Vetrimani Elangovan
Head of the Department (PSIS),
Presidency School of CSE&IS
Presidency University

Dr. R Mahalakshmi
Associate Dean,
Presidency School of IS
Presidency University

Dr. Md. Sameeruddin Khan
Pro-VC and Dean,
Presidency School of CSE&IS
Presidency University

ABSTRACT

Road accidents caused by driver drowsiness remain a critical concern in traffic safety, accounting for a significant percentage of fatal crashes worldwide. Traditional methods of detecting driver fatigue, such as vehicle-based or physiological signal-based systems, are often intrusive or limited in effectiveness. This project proposes a non-intrusive, real-time drowsiness detection system using deep learning techniques implemented in Python, aiming to provide a cost-effective and efficient solution for enhancing driver safety.

The proposed system utilizes a webcam to continuously monitor the driver’s face and analyze visual cues indicative of drowsiness, such as eye closure duration, blinking frequency, yawning, and head position. The approach is based on a Convolutional Neural Network (CNN) model trained on publicly available datasets such as the Yawn Detection Dataset, Closed Eyes in the Wild (CEW), and MRL Eye Dataset. These datasets contain annotated images of facial states (open/closed eyes, yawning), which are essential for training the deep learning model to accurately classify drowsy and non-drowsy states.

The system architecture consists of multiple stages: face detection using Haar cascades or Dlib facial landmarks, region-of-interest (ROI) extraction for eyes and mouth, and feature classification using CNNs built with TensorFlow/Keras. The model processes video frames in real-time using OpenCV, determining whether the driver's eyes are closed for a prolonged period (based on Eye Aspect Ratio - EAR) or if yawning is detected. If the detected behavior crosses predefined thresholds, an alarm is triggered to alert the driver, thereby preventing potential accidents.

TABLE OF CONTENTS

Abstract		i
Table of Contents		ii
List of Figures		iv
Acknowledgement		v
Chapter No.	**Topic Name**	**Page No.**
1	**INTRODUCTION**	1
	1.1 Overview	1
	1.2 Scope of the Study	2
2	**REQUIREMENT ANALYSIS**	4
	2.1 Functional Requirements	5
	2.2 Non- Functional Requirements	5
	2.3 Technical Requirements	5
	2.4 User Requirements	6
3	**LITERATURE REVIEW**	8
	3.1 Drowsiness Detection System for Masked Face	8
	3.2 Enhanced Driver’s Drowsiness Detection	8
	3.3 Driver Drowsiness Detection System With OpenCV and Keras	9
	3.4 Deep Learning Based Driver	9
	3.5 Vision Based Driver Drowsiness Detection Using Deep Learning	10
	3.6 Investigation on Driver Drowsiness Detection Using Deep Learning	10
4	**PROPOSED SYSTEM**	12
	4.1 Proposed System Architecture	12
	4.1.1 Input Driving Videos	12
	4.1.2 Feature Extraction and Classification	14
	4.1.3 Data Pre-Processing and Augmentation	14
	4.1.4 Model Execution	17
	4.1.5 Model Evaluation	17
5	**SYSTEM DESIGN**	20
	5.1 System Architecture Diagram	20
	5.1.1 Input Acquisition(video input/camera)\	21
	5.1.2 Face Detection	21

	5.1.3 Eye and Mouth Region Extraction	23
	5.1.4 Calculation of EAR and MAR	24
	5.1.5 Eye State Classification Using CNN	24
	5.1.6 Drowsiness Detection Logic	25
	5.1.7 Alert Mechanism	26
	5.1.8 System Performance and Optimization	27
	5.1.9 Scalability and Integration	27
6	**IMPLEMENTATION**	29
	6.1 Environment Setup and Library Installation	29
	6.2 Real Time Video Capture with OpenCV	30
	6.3 Face Detection and Facial Landmarking Extraction	32
	6.4 Eye and Mouth Region Localization	33
	6.5 calculation of EAR and MAR	34
	6.6 CNN Based Eye State Classification	36
	6.7 Decision Logic for Drowsiness Detection	37
	6.8 Alert Generation	39
	6.9 System Loop and Real Time Optimization	40
	6.10 Testing and Validation	41
7	**TESTING**	43
	7.1 Aim	43
	7.2 Setting Up the Test Environment	43
	7.3 Test Scenarios	44
	7.4 Performance Metrics	45
	7.5 Evaluation of the EAR and MAR Calculations	46
	7.6 CNN Model Testing	47
	7.7 Alert System Testing	47
	7.8 Long Duration and Real Time Performance Testing	48
	7.9 User Feedback and Observations	49
	7.10 Summary of Testing Results	50
8	**CONCLUSION**	53
	8.1 Conclusion	53
	APPENDIX	55
	Coding	55
	Screenshots	63

	REFERENCES	66

LIST OF FIGURES

Figure No.	Caption	Page No.
4.1	Deep Learning Based Drowsiness Detection Pipeline	12
4.2	Flowchart of Drowsiness Detection	17
4.3	Model Evaluation	18
5.1	System Architecture Diagram	20
5.2	Input Acquisition	21
5.3	Eye and Mouth Region Extraction	23
5.4	EAR and MAR Calculations	24
5.5	Drowsiness Detection	25
5.6	Alert Sound and Message	27

ACKNOWLEDGEMENT

The completion of project work brings with great sense of satisfaction, but it is never completed without thanking the persons who are all responsible for its successful completion. First and foremost we indebted to the GOD ALMIGHTY for giving us the opportunity to excel our efforts to complete this project on time. We wish to express our deep sincere feelings of gratitude to our Institution, Presidency University, for providing us opportunity to do our education.

We express our sincere thanks to our respected dean **Dr. Md. Sameeruddin Khan,** Pro-Vice Chancellor, School of Engineering, and Dean, Presidency School of CSE and IS, Presidency University for getting us permission to undergo the project.

We record our heartfelt gratitude to our beloved professor **Dr. R Mahalakshmi**, Associate Dean, Presidency School of Information Science, **Dr. Vetrimani Elangovan**, Professor and Head, Presidency School of Information Science, Presidency University for rendering timely help for the successful completion of this project.

We sincerely thank our project guide, **Mr. Sakthi S**, Assistant Professor, Presidency School of Computer Science Engineering, for his guidance, help and motivation. Apart from the area of work, we learnt a lot from him, which we are sure will be useful in different stages of our life. We would like to express our gratitude to Faculty Coordinators and Faculty, for their review and many helpful comments.

We would like to acknowledge the support and encouragement of our friends.

Student Name	***ID Number***
M. Yasaswini Sai	***20221BCA0044***
E. Kushal	***20221BCA0053***
Y. Gnana Pavani	***20221BCA0060***
Velpula Usha	***20221BCA0185***

CHAPTER 1
INTRODUCTION

Drowsy driving is a critical issue that contributes significantly to road accidents worldwide, often resulting in severe injuries and fatalities. Detecting driver fatigue early is essential for preventing such mishaps and ensuring road safety. Traditional approaches to monitoring drowsiness—such as observing steering behavior or integrating vehicle sensors—are either invasive, costly, or hardware-dependent. With the advancement of computer vision and artificial intelligence, deep learning offers a more effective and software-based solution to this problem. This project aims to implement a vision-based drowsiness detection system using deep learning techniques in Python. The system captures live video input from a webcam to monitor the driver's facial features in real-time.

By using libraries like OpenCV and Dlib, the program detects the driver's face and extracts crucial landmarks around the eyes and mouth. It then calculates Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR) to identify signs of fatigue, such as prolonged eye closure or frequent yawning. Deep learning models, particularly Convolutional Neural Networks (CNNs), are employed to classify the driver's state as either "alert" or "drowsy."

If the system detects signs of drowsiness, it immediately triggers an alert through sound or visual cues to regain the driver's attention. This Python-based implementation is cost-effective, scalable, and does not rely on external IoT devices or specialized hardware, making it a practical solution for integration into personal vehicles and public transport systems. Overall, this project demonstrates how artificial intelligence can enhance driver safety and reduce the risks associated with fatigue-induced driving.

1.1 Overview

Drowsiness detection is a crucial area of research in intelligent transportation systems, aimed at preventing accidents caused by fatigued or sleepy drivers. This project leverages deep learning and computer vision techniques to build a non-intrusive, real-time system that monitors a driver's alertness level using only a webcam and Python-based tools—without the need for IoT devices or expensive sensors.

The project begins by capturing a live video feed using a standard webcam (via OpenCV's cv2.VideoCapture). From each video frame, the system detects the driver's face using facial detection methods such as Dlib's HOG (Histogram of Oriented Gradients) detector or OpenCV's Haar cascades. Once the face is located, facial landmarks are extracted—focusing on key regions like the eyes and mouth. These landmarks are crucial for calculating behavioral cues like the Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR).

The EAR helps in identifying prolonged eye closures, a reliable indicator of drowsiness. Similarly, MAR helps detect yawning, another common sign of fatigue. The system continuously monitors these values over time. If the EAR falls below a predefined threshold for several consecutive frames, or if yawning is detected based on MAR or a CNN classifier, the system concludes that the driver is drowsy.

To enhance accuracy, a deep learning model—usually a CNN—is optionally used to classify eye states (open vs. closed) and mouth states (yawning vs. not yawning). This model is trained on labeled image datasets containing diverse examples of facial expressions and eye/mouth states under different lighting and angle conditions.

Once drowsiness is detected, the system immediately issues an alert through audio (using Python libraries like playsound, pygame, or winsound) or visual cues to draw the driver's attention. This real-time alert mechanism plays a vital role in preventing road mishaps due to driver fatigue.

This project is highly practical due to its software-based design, requiring no specialized hardware beyond a basic camera. It is implemented entirely in Python, utilizing open-source libraries like OpenCV, Dlib, NumPy, and TensorFlow/Keras. The modular design also allows for future enhancements, such as head pose estimation, blink frequency analysis, or integration with vehicle safety systems.

1.2 Scope of the Study

The extent of this research is concerned with the creation of a real-time drowsiness detection system based on Convolutional Neural Networks (CNNs), with a focus on improving driver safety through the detection of early indicators of fatigue. The system adopts a camera-based, non-intrusive method that constantly observes the facial features of the driver via a video stream recorded by the use of a webcam or vehicle-mounted camera.

The CNN model is trained to recognize important visual indicators of drowsiness, including extended eye closure, excessive blinking, yawning, and small head movements like tilting or nodding. Through the extensive image analysis power of CNNs, the system can effectively classify whether the driver is awake or drowsy based on facial patterns in the video frames. The model can make use of personalized architectures or pre-trained networks (e.g., VGG or MobileNet) to identify meaningful features and enhance detection performance. Real-time application is one of the major elements of this research, in which software such as OpenCV and dlib are employed for handling video input and facial landmark detection. Upon discovering drowsiness signs, the system sends automatic feedback through alarm or notice so as to hinder possible accidents. The research involves environments like a car where drivers' alertness is very significant and employs commonly available datasets like YawDD, NTHU Drowsy Driver Dataset for training as well as validating. Physiological techniques like EEG or ECG are intentionally avoided to maintain the system non-invasive, feasible, and affordable for field deployment.

The extent of this research is concerned with the creation of a real-time drowsiness detection system based on Convolutional Neural Networks (CNNs), with a primary focus on enhancing driver safety by identifying early signs of fatigue. The study centers on developing a non-intrusive, camera-based monitoring system that continuously observes the facial behavior of drivers via a live video stream, captured through either a webcam or a vehicle-mounted camera.

The CNN model is specifically trained to detect key visual indicators of drowsiness, such as:

- Prolonged eye closure
- Frequent or abnormal blinking
- Yawning
- Subtle head movements like tilting or nodding

These indicators are analyzed in real time through frame-by-frame inspection using computer vision techniques and facial landmark detection. Advanced tools such as OpenCV, dlib, and potentially MediaPipe are integrated for efficient processing of video input and extraction of facial features, including eye aspect ratio (EAR), mouth aspect ratio (MAR), and head pose estimation.

To ensure robust and accurate detection, the CNN model can utilize either custom-built architectures or pre-trained deep learning networks like VGG16, MobileNet, or ResNet, allowing for transfer learning and faster convergence on limited datasets. The model is trained and validated using publicly available benchmark datasets such as:

- YawDD (Yawning Detection Dataset)
- NTHU Drowsy Driver Detection Dataset
- Closed Eyes in the Wild (CEW)

CHAPTER 2
REQUIRMENT ANALYSIS

In recent years, significant strides have been made in the field of computer vision for sign language recognition, ushering in a new era of inclusive communication and accessibility. Notable research from 2014 by Yan, Ji, and Li highlighted the real-time capabilities of sign language recognition using depth sensors such as Kinect, which revolutionized gesture-based communication by effectively capturing and processing sign language gestures. Furthermore, the work of Malik and Zhang in the same year introduced the incorporation of wearable devices for American Sign Language recognition, offering portability and convenience. Their wearable computer-based video system has made it possible for users to engage in seamless sign language communication on the go.

A pivotal factor behind these transformative advancements has been the integration of machine learning, particularly deep learning techniques. Research by Som and Tiwari in 2018 demonstrated the efficacy of Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks in real-time fingerspelling recognition, significantly enhancing the accuracy and efficiency of gesture interpretation. These collective achievements in the field of computer vision and machine learning have opened new possibilities for bridging communication gaps and promoting accessibility for the deaf and hard-of-hearing communities, marking a remarkable shift in sign language recognition technology.

As the field of sign language recognition continues to evolve, the fusion of advanced hardware with intelligent software algorithms has become increasingly vital. The deployment of depth-sensing cameras, wearable sensors, and sophisticated computer vision techniques allows systems to not only identify hand gestures but also interpret facial expressions and body posture—key components of sign language. Recent advancements have also seen the rise of lightweight and mobile-compatible models, enabling real-time recognition on smartphones and embedded devices, which further enhances user accessibility and convenience.

Beyond gesture recognition, attention is now shifting toward multilingual and context-aware sign language systems. For instance, researchers are exploring models capable of distinguishing between different sign languages and dialects, as well as systems that adapt to individual signing styles. Integration with natural language processing (NLP) also paves the way for seamless translation between sign language and spoken or written text. Furthermore, the use of datasets comprising diverse signers from various demographics ensures that the systems are inclusive and robust, minimizing bias and maximizing effectiveness in real-world applications.

2.1 Functional Requirements

- Real-Time Drowsiness Detection, the system must capture video input through a webcam and analyze it to detect signs of drowsiness such as prolonged eye closure or yawning.
- Eye and Facial Feature Tracking; the system must detect facial landmarks (especially around the eyes and mouth) using computer vision techniques.
- Alarm Trigger, if the driver is detected to be drowsy (e.g., eyes closed for a predefined duration), an alert/alarm must be triggered to wake them.
- Live Camera Feed Integration, the application should work with a live camera feed (from a laptop or USB webcam) to analyze frames in real time.
- Display Status**,** the system should indicate the current status (e.g., "Alert", "Drowsy", "Sleeping") on the screen as feedback.

2.2 Non-Functional Requirements

- **Accuracy**- The detection system should maintain high accuracy in recognizing eye closure and yawning even in low-light or partial occlusion conditions.
- **Performance**- The system must process video frames in real time with minimal delay.
- **Usability**- The interface should be simple and user-friendly, requiring minimal setup to start monitoring.
- **Scalability**- Though developed for a single-user setup (e.g., one driver), the system should be modular enough for future enhancement or integration into multi-camera environments.

2.3 Technical Requirements

- **Programming Language**- Python 3.x
- **Libraries/Tools Used**
    - OpenCV for image processing

    - Dlib and/or Mediapipe for facial landmark detection
    - TensorFlow/Keras for any deep learning components
    - Pygame or playsound for triggering alarms

- **Hardware Requirements**
    - A computer with a working camera
    - Minimum 4GB RAM
- **Software Requirements**
    - Python IDE (like VS Code)
    - Required Python libraries installed via pip

2.4. User Requirements

- Minimal Setup and Easy Execution, the system should be easy to install and run using basic Python commands and a standard webcam without requiring specialized hardware or IoT devices.
- Real-Time Monitoring Interface the application should automatically launch the webcam feed and process video frames in real time without the need for manual start/stop buttons.
- Visual Feedback the system must display the live camera feed with overlaid facial landmarks (eyes, mouth) and show warning messages like "Drowsiness Detected" directly on the video frame using OpenCV.
- Auditory Alerts when drowsiness is detected based on predefined thresholds (EAR and MAR), the system must immediately trigger a sound alert (beep or audio message) to alert the user.
- Robust Across Conditions the system should function effectively under various lighting conditions, head angles, and even when the user wears glasses.
- Customizable Thresholds and Parameters to improve adaptability for different users, the system should allow basic configuration of drowsiness detection thresholds such as Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR). This flexibility ensures that sensitivity can be adjusted depending on individual physiological differences or

user preferences.

- Fail-Safe Behavior, in case the system fails to detect facial landmarks (e.g., due to poor lighting, obstructions, or absence of the user), it should issue a visual or auditory notification indicating tracking loss or detection failure. This ensures the user is aware that the monitoring system is not currently active.
- Low Resource Consumption to ensure smooth performance on mid-range systems, the application should optimize frame processing and model loading to avoid CPU overload. Techniques such as frame skipping, model quantization (if deep learning is used), or asynchronous threading may be implemented to enhance efficiency.
- Error Handling and Feedback the system should be equipped to handle common errors (e.g., camera not found, missing dependencies) gracefully. Clear error messages or pop-ups should guide the user to resolve issues without needing advanced technical knowledge.

CHAPTER 3
LITERATURE REVIEW

3.1 Drowsiness Detection System for Masked Face

The Drowsiness Detection System for Masked Face, developed by Eryl Nanda Pratama and Wikky Fawwaz Al Maki in 2022, focuses on a crucial problem in the field of driver safety—detecting drowsiness even when a person's face is partially covered by a mask. The system addresses the growing need for such solutions, especially during health crises like the COVID-19 pandemic when wearing face masks became a norm.

To build and evaluate their model, the authors used three distinct datasets. Dataset 1 consists of grayscale images without human faces, obtained publicly from Kaggle, likely used for training a background or non-face detector. Dataset 2 comprises 33 videos, a mix of self-recorded content and publicly available YouTube videos, offering diverse real-world conditions. Dataset 3 is the Drowsiness Detection Dataset from Kaggle, containing 4,000 labeled images, which likely helped train and validate the model on key visual indicators of drowsiness.

Challenges:

- Detecting faces with and without masks.
- Real-time detection with resource constraints.

Advantages:

- Achieved 81% accuracy using DNN and Haar Cascade.
- Robust against variations in facial coverage.

Disadvantages:

- Limited generalizability to diverse lighting and environmental conditions.

3.2 Enhanced Driver's Drowsiness Detection

The study conducted by Mandapati Ankitha, Jarabala Sindhu Bhargavi, et al. in 2023 presents a sophisticated approach to drowsiness detection, emphasizing the integration of advanced deep learning techniques with real-time video analysis. The researchers aimed to develop a system capable of accurately identifying signs of drowsiness even in dynamic and uncontrolled environments, such as while driving or operating machinery.

Although the dataset used in the study is not explicitly specified, the researchers employed data augmentation techniques to expand and diversify the training data. This helped the model generalize better across various conditions. In addition, the system incorporated real-time video input, enabling continuous monitoring and detection, which is essential for practical deployment in real-world scenarios.

Challenges:

- Robustness under various lighting conditions.

Advantages:

- Utilized CNN for feature extraction with a 97% accuracy.
- Real-time processing with head movement detection.

Disadvantages:

- High computational cost for advanced CNN models.

3.3 Driver Drowsiness Detection System with OpenCV and Keras

The Driver Drowsiness Detection System with OpenCV and Keras, developed by R Syed Ali Fathima, Kota Naga Jyothi, et al. in 2024, focuses on creating a practical and cost-effective solution for monitoring driver fatigue. This system leverages computer vision and deep learning frameworks to enhance road safety by detecting signs of drowsiness before accidents occur.

The researchers utilized a dataset consisting of real-world images captured under various lighting conditions, enabling the system to be trained on a range of environmental scenarios. This helped in making the model more applicable to real-life driving conditions, though some limitations remained under extreme lighting variances.

Challenges:

- Differentiating between drowsiness and other facial behaviours.

Advantages:

- Uses Viola-Jones algorithm for efficient eye detection.
- Low-cost implementation suitable for real-time use.

Disadvantages:

- Limited accuracy in poor lighting conditions.

3.4 Deep Learning Based Driver

The Deep Learning-based Driver Drowsiness Detection system, developed by Parth P. Patel, Chirag L. Pavesha, et al. in 2022, aims to enhance road safety through the detection of driver fatigue using computer vision techniques. The system specifically focuses on key visual indicators such as eye blinks and yawning, which are among the most prominent and early signs of drowsiness.

The model was trained and tested on a dataset comprising eye images, allowing it to learn the patterns associated with open and closed eye states, as well as prolonged eye closure indicative of fatigue. By centering on these visual features, the system was able to deliver reliable predictions in real-time applications.

Challenges:

- Achieving consistent results across different individuals.

Advantages:

- Lightweight model suitable for embedded systems.
- High accuracy of 96% in real-time scenarios.

Disadvantages:

- Only focuses on eye-related features; ignores other cues.

3.5 Vision Based Driver Drowsiness Detection Using Deep Learning

The research titled Vision-Based Driver Drowsiness Detection Using Deep Learning, conducted by S. Spandana, M. Srividhya, et al. in 2024, presents an advanced approach to monitoring driver alertness using computer vision. The study emphasizes real-time detection of drowsiness indicators through facial cues, aiming to enhance driver safety through timely intervention.

The system was trained on a dataset containing 6,000 images of drivers captured under varied conditions, including instances of yawning and eye closure. This diverse dataset enabled the model to learn a range of drowsiness symptoms and improve its generalization across different users and environments.

Challenges:

- Capturing subtle signs like partial yawning or blinking.

Advantages:

- High accuracy (97%) with Mobile Net and SSD architecture.
- Efficient for deployment in low-cost embedded devices.

Disadvantages:

- Struggles with complex real-world scenarios like extreme

3.6 Investigation on Driver Drowsiness Detection Using Deep Learning

The study titled Investigation on Driver Drowsiness Detection using Deep Learning Approaches by Dakshnakumar G S and J. Anitha, published in 2023, explores the effectiveness of various deep learning models in detecting driver drowsiness. The research emphasizes using high-performing neural networks and transfer learning to build a robust and efficient detection system.

The model was trained on a dataset of 4,000 images obtained from Kaggle, which were categorized based on eye states—either open or closed. This binary classification served as the basis for determining alertness levels, making eye status a central feature in the drowsiness detection process.

Challenges:

- Adapting pre-trained models for drowsiness-specific datasets.

Advantages:

- Exceptional accuracy: EfficientNetB7 (99.87%) and MobileNetV2 (99%).
- Uses transfer learning for reduced training time.

Disadvantages:

- Computationally intensive for larger models like EfficientNet.

CHAPTER 4
PROPOSED SYSTEM

4.1 Proposed System Architecture

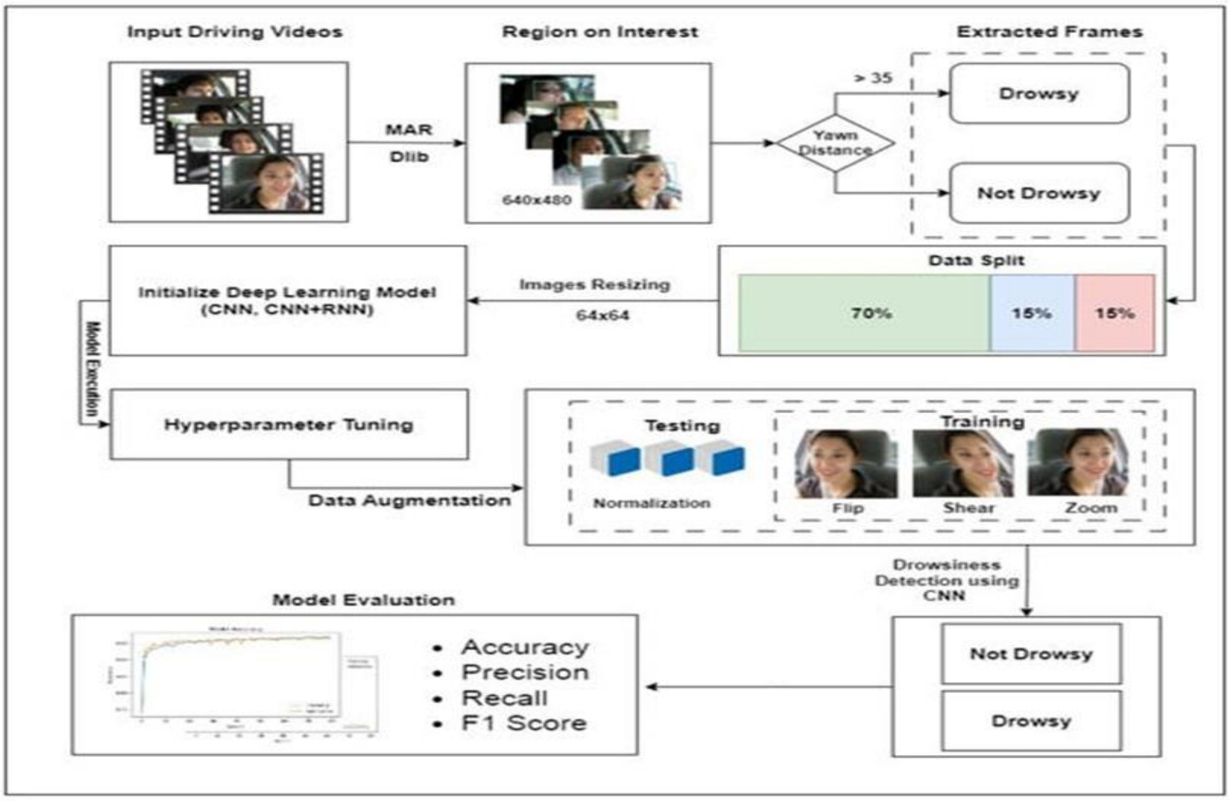


Fig 4.1 Deep Learning Based Drowsiness Detection Pipeline

The above shown fig 4.1 illustrates a deep learning-based drowsiness detection system using CNN and CNN+RNN models. It begins with input driving videos from which facial regions of interest are extracted using Dlib and MAR (Mouth Aspect Ratio). Frames are classified as "Drowsy" or "Not Drowsy" based on a yawn distance threshold (>35). These frames are resized to 64x64 pixels and split into training (70%), validation (15%), and testing (15%) datasets. Data augmentation techniques such as flipping, shearing, and zooming are applied to enhance the model's robustness. The system undergoes hyperparameter tuning and is evaluated using metrics like accuracy, precision, recall, and F1 score. The final trained CNN model classifies the driver’s state in real-time as either "Drowsy" or "Not Drowsy".

4.1.1 Input Driving Videos

Function

The input driving videos serve as the starting point of the drowsiness detection system as shown in fig 4.2. The primary responsibility of this module is to acquire continuous visual data in the form of real-time or recorded video streams, capturing the driver's face for monitoring and analysis. The quality, consistency, and reliability of this video input are critical for accurate facial feature detection and drowsiness classification. The system processes each frame of the video to extract features indicative of drowsiness, such as eye closures and yawning.

Details

- Continuous-video-capture: The system continuously captures frames from the camera feed, typically at a frame rate of 10–15 frames per second (FPS). This frame rate is optimized to strike a balance between the need for real-time responsiveness and the computational load.
- Frame-Extraction: Each captured video frame is treated as an individual static image. The system processes these frames one by one, identifying the driver’s face and extracting relevant facial landmarks for further analysis as shown in fig 4.1(e.g., eyes, mouth).
- Face-Detection: A critical part of this module is checking for the presence of a face in each frame. If no face is detected, the frame is discarded to avoid unnecessary processing, reducing computational overhead and improving efficiency.
- Controlled-Frame-Rate: To maintain performance and ensure that only the most relevant images are processed, the system operates at a controlled frame rate (e.g., 10–15 FPS). This reduces unnecessary computational burden while ensuring that time-sensitive events, such as blinking or yawning, are captured effectively.
- Frame-Skipping-for-Efficiency: Not every frame needs to be analyzed for drowsiness detection. A frame sampling technique is used where the system processes every third or fourth frame, reducing the load on the system and making it more resource-efficient. This is particularly useful for devices with limited computational capacity.
- Resolution-Management: Input frames are resized to standardized dimensions, such as 640x480 or 320x240, to ensure uniformity and faster computation without significant loss in feature visibility. Lower-resolution frames are particularly useful for real-time applications, especially when running facial landmark detection models and deep learning networks that require consistent and manageable input sizes.

Tools and Techniques

OpenCV

OpenCV, an open-source computer vision library, is used to initialize the webcam, continuously capture video frames, and convert them into formats suitable for further processing, such as grayscale images. OpenCV also supports integrating graphical user interface (GUI) elements to display the live video feed during real-time analysis.

Pre-processing Enhancements

- Grayscale-Conversion: The system converts the captured frames to grayscale to simplify the image data. Grayscale images are less computationally expensive to process and are sufficient for face detection and facial landmark extraction.
- Frame-Skipping: The system uses frame sampling to skip frames, processing only a subset of frames (e.g., every 3rd or 4th frame). This reduces the workload while still enabling the system to detect significant drowsiness events.

4.1.2 Feature Extraction and Classification

Function

Focuses on identifying relevant facial features and classifying the driver’s state (drowsy or not).

Details

- Region of Interest (ROI): From the input frames, the driver’s face is localized using tools like Dlib. Only this ROI is processed further.
- Mouth Aspect Ratio (MAR): Used to compute the yawning distance using mouth landmarks as shown in fig 4.1. If the distance exceeds a threshold (e.g., >35), it signals drowsiness.
- Data Labeling: Based on MAR, frames are labeled as "Drowsy" or "Not Drowsy".
- Classification Models: Deep learning models such as CNN or CNN+RNN are initialized to classify the driver's state.

Tools and Techniques

- Dlib for landmark detection.
- MAR calculation for yawning.
- CNN/CNN+RNN architectures for learning patterns in facial expressions.

4.1.3 Data Pre-Processing and Augmentation

Function

Data Pre-Processing is next process as shown in fig 4.2, Prepares the image data to be compatible with the model and increases dataset robustness. The image preprocessing and dataset augmentation function is an essential step in preparing the data to be compatible with the deep learning model and enhancing its ability to generalize across different conditions. It ensures that the model can handle variations in the input images, which improves its robustness and reduces the chances of overfitting.

Details

- Resizing: To ensure uniformity and efficiency, all input images are resized to a standard size, typically 64x64 pixels as we explained in fig 4.2. This size is chosen because it balances the computational cost and ensures that the key features of the image, such as eyes and mouth, are still detectable without unnecessarily large input dimensions. Resizing helps in reducing the input size as shown in fig 4.1, which speeds up training and inference, while still retaining enough detail for the model to make accurate predictions.
- Splitting: The dataset is divided into three subsets:
- Training Set (70%): This portion is used for training the model and learning the features.
- Validation Set (15%): This data is used to validate the model during training, helping to fine-tune hyperparameters and reduce the risk of overfitting.
- Test Set (15%): The test set is kept aside to evaluate the model's performance on unseen data. This provides an unbiased estimate of how well the model is expected to perform in real-world scenarios.
- Augmentation: Data augmentation is a critical technique to artificially increase the diversity of the dataset by applying random transformations to the images. This allows the model to become more robust and prevents it from learning spurious patterns that are not generalizable to new data. Common augmentations include:
- Flip: Horizontally flipping the image simulates changes in viewing angles and provides the model with more varied examples of faces.
- Shear: This transformation skews the image along one axis, helping the model learn to recognize faces even if they are tilted or distorted.
- Zoom: Random zooming of the image mimics changes in the distance from the camera, allowing the model to generalize better across different facial scales.
- These augmentations not only increase the dataset size but also simulate different real-world conditions (such as head tilt, different distances from the camera, or slight camera distortions), improving the model’s ability to generalize and reducing the likelihood of overfitting to the original data.

- Normalization: Normalization is applied to the test images to standardize pixel values. The pixel values of images range from 0 to 255, and normalization ensures that these values are scaled to a 0 to 1 range (dividing by 255) or standardized to have a mean of 0 and a standard deviation of 1 (zero-centered). This helps speed up the training process, improves convergence, and ensures that the model treats each feature (pixel) equally, making the learning process more stable.

Tools and Techniques

- Keras ImageDataGenerator: This class in Keras is commonly used for real-time data augmentation. It can be easily configured to apply transformations such as flipping, shearing, zooming, and rotation to the images on-the-fly during training. By using this tool, the model receives augmented versions of the images in each epoch, which improves its robustness.
- NumPy/PyTorch Preprocessing: Both NumPy and PyTorch offer tools to perform the necessary image preprocessing, including resizing, normalization, and splitting. NumPy can be used for simple image manipulation (such as resizing), while PyTorch provides dedicated functions to handle normalization and tensor conversions. These libraries help streamline the preprocessing pipeline and ensure the model receives well-processed inputs for efficient training.
- Benefits of Image Preprocessing and Augmentation:
- Improved Model Generalization: By augmenting the dataset, the model learns from a wider range of input variations, which allows it to generalize better to new, unseen images.
- Reduced Overfitting: The introduction of transformed images prevents the model from memorizing the training data. The increased variability in training examples forces the model to learn more generalized features.
- Real-World Simulation: The augmentations simulate real-world conditions such as different facial orientations, distances from the camera, and lighting conditions, which makes the model more resilient when deployed in varied environments.
- By combining resizing, data augmentation, and normalization, this preprocessing function ensures that the model is trained with a robust, diverse set of images, leading to better performance and accuracy in real-time drowsiness detection

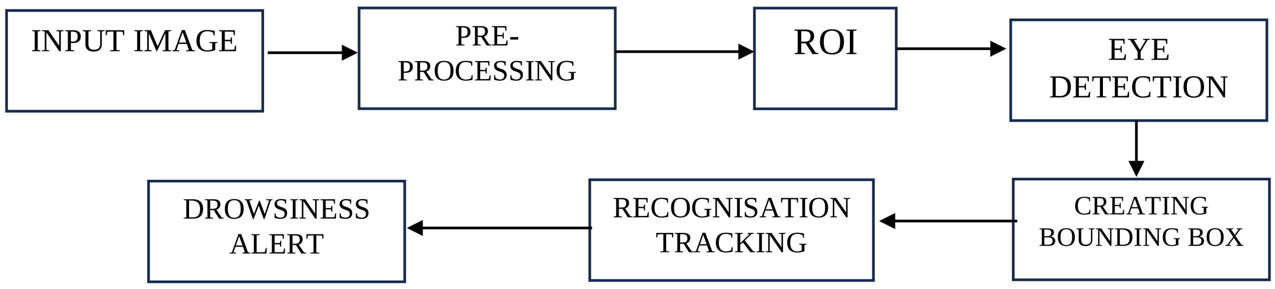


Fig 4.2 Flowchart of Drowsiness Detection

4.1.4 Model Execution

Function

This module handles the actual model training and optimization.

Details

- The initialized CNN or CNN+RNN model is trained using the processed dataset.
- Hyperparameter Tuning: Adjusts learning rate, number of layers, batch size, etc., for better performance.

Tools and Techniques

- TensorFlow or PyTorch for model creation and training.
- Grid search or random search for hyperparameter tuning.

4.1.5 Model Evaluation

Function

Measures how well the model performs on unseen data.

Details

- Performance Metrics: Evaluates the model using
  - Accuracy – How many predictions are correct overall.
  - Precision – How many predicted positives are actually positive.
  - Recall – How many actual positives were correctly predicted.
  - F1 Score – Harmonic mean of precision and recall.

The primary function of model evaluation is to measure how effectively the drowsiness detection model performs on unseen or real-world data. It ensures that the system not only performs well on training data but also generalizes effectively to new inputs—such as diverse faces, lighting conditions, and camera angles.

Video Input Collection/Evaluation

Function

This is the foundational stage of the driver drowsiness detection system. It is responsible for acquiring a continuous stream of visual data from which the presence and behavior of the driver can be monitored.

Output:

- The model classifies incoming video frames as either as shown in fig 4.3:
    - Drowsy
    - Not Drowsy

Tools and Techniques:

- sklearn.metrics for evaluation.
- Plotting tools like Matplotlib or Seaborn to visualize result

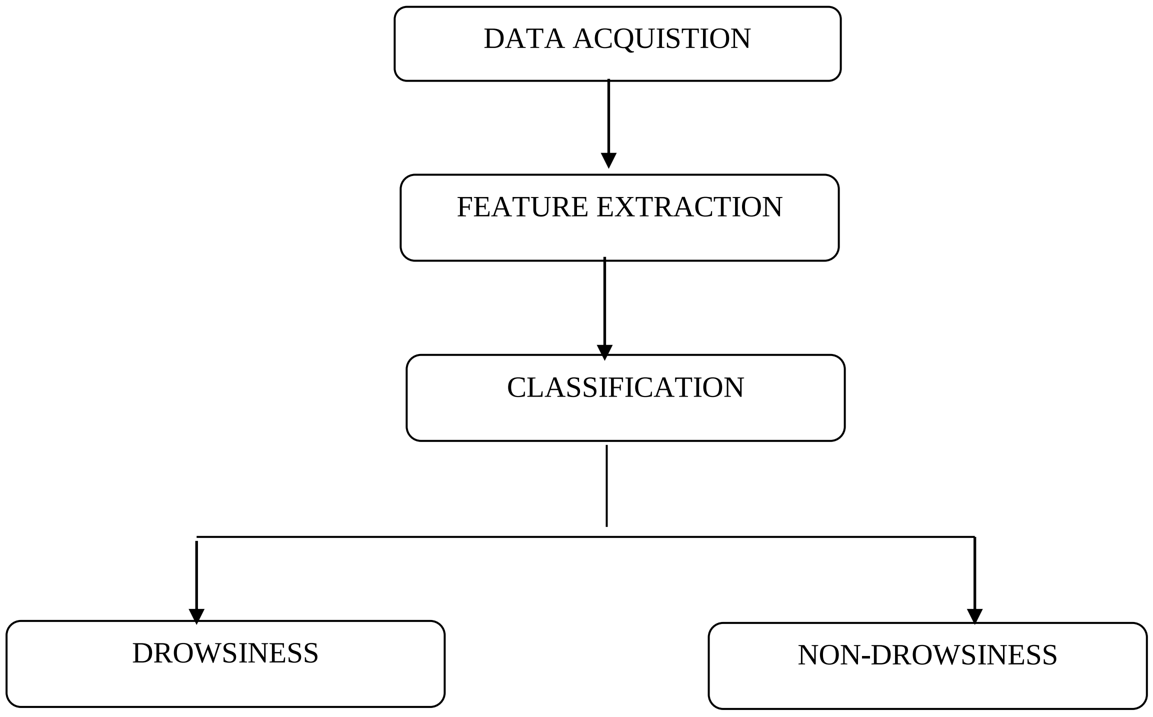


Fig 4.3 Model Evaluation

The primary function of model evaluation is to measure how effectively the drowsiness detection model performs on unseen or real-world data. It ensures that the system not only performs well on training data but also generalizes effectively to new inputs—such as diverse faces, lighting conditions, and camera angles.

Details

Performance Metrics: Evaluation is carried out using a set of statistical performance metrics that provide a comprehensive view of the model's effectiveness in classifying video frames into Drowsy or Not Drowsy:

- Accuracy

  Measures the proportion of correct predictions among the total number of predictions

- Precision

  Indicates the percentage of positive identifications that were actually correct.

- Recall

  Indicates how many actual positives were correctly identified.

CHAPTER 5
SYSTEM DESIGN

5.1 System Architecture Diagram

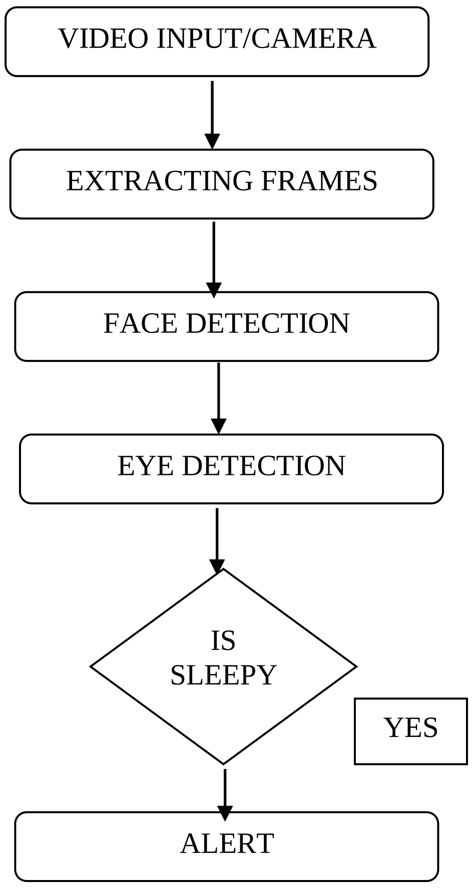


Fig 5.1 System Architecture Diagram

As shown in fig 5.1, the driver drowsiness detection system starts with the video input/camera, which captures real-time footage of the driver. This video is processed by extracting frames, allowing the system to analyze images individually. In each frame, face detection is performed to locate the driver's face, followed by eye detection to identify the eyes. Once the eyes are detected, the system evaluates if the driver is sleepy by analyzing patterns like prolonged eye closure using techniques such as the Eye Aspect Ratio (EAR). If the condition is met (as indicated by the YES decision in the diagram), the system triggers an ALERT to warn the driver, thereby helping to prevent accidents due to drowsiness. This flowchart visually represents the logical sequence and decision-making process involved in real-time drowsiness detection.

5.1.1 Input Acquisition (video input/camera)

Real-time video input acquisition is the first part of the system. A webcam is placed inside the vehicle cabin, most often at the driver's face. The video stream is read frame by frame with the OpenCV library in Python. The frames are processed at an appropriate frame rate (e.g., 20–30 FPS) for optimal responsiveness and computational requirements. The camera needs to have good resolution and perform well under changing lighting conditions since image quality directly influences the downstream tasks like face and eye detection.
Real-time video input acquisition is the foundational step of the drowsiness detection system.

A webcam or in-cabin as shown in fig 5.2 camera is mounted in the vehicle, typically directed at the driver's face to ensure clear visibility of facial features. The OpenCV library in Python is used to continuously read the video stream frame by frame, enabling real-time processing. These frames are generally captured at a frame rate of 20–30 frames per second (FPS), which strikes a balance between system responsiveness and computational efficiency.
For reliable performance, the camera should support high-definition (HD) resolution (e.g., 720p or 1080p) and possess low-light adaptability or infrared (IR) capability to handle varying lighting conditions, such as nighttime driving or sudden brightness changes. Poor lighting or blurry images can severely degrade the accuracy of downstream tasks like face detection, eye tracking, and mouth aspect ratio (MAR) calculations.

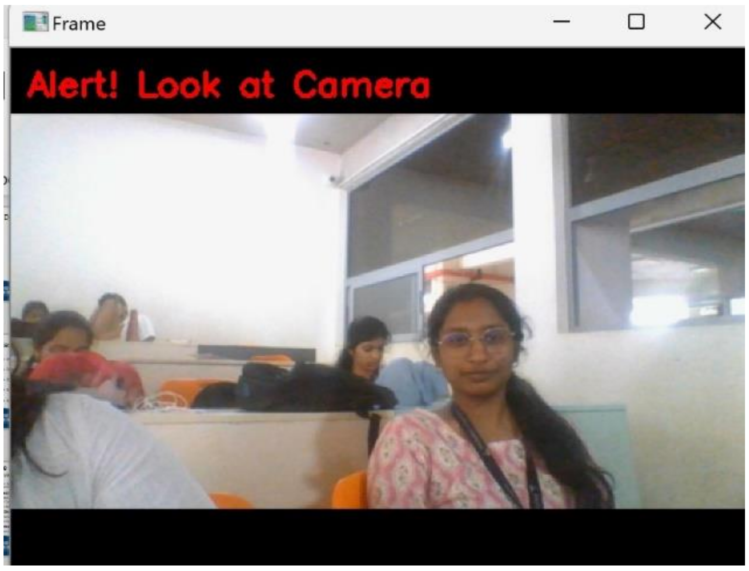


Fig 5.2 Input Acquisition

5.1.2. Face Detection

After video frames are captured, the next task is face detection. This is done with Dlib's pre-trained Histogram of Oriented Gradients (HOG) face detector or a CNN-based face detector, depending on the trade-off between speed and accuracy desired. After detecting the face, Dlib's 68-point facial landmark predictor is used to detect important facial areas such as the eyes, nose, mouth, and jawline. These anchor points are important in determining the precise geometry of the eyes and mouth, which are subsequently utilized to calculate behavioral indicators such as blinking and yawning.

Once individual frames are extracted from the video stream, the next critical step is face detection. This is typically performed using Dlib, a popular machine learning library for computer vision tasks. Dlib offers two main types of face detectors:

Histogram of Oriented Gradients (HOG)-based detector

- Lightweight and fast.
- Suitable for real-time applications where speed is prioritized over precision.
- Works reasonably well under normal lighting but may struggle with profile faces or poor lighting conditions.

CNN-based face detector

- Uses a deep learning model for higher accuracy, especially in detecting partially occluded faces or under challenging lighting.
- More computationally expensive, hence used when accuracy is more important than speed.
- Uses a Convolutional Neural Network to accurately detect faces in images by learning complex facial features, offering higher precision and robustness compared to traditional methods.

After the face is located in the frame, Dlib’s 68-point facial landmark predictor is employed to identify and map key facial regions. These 68 landmarks represent critical features, including:

- Eyes (12 points)
- Eyebrows (10 points)
- Nose (9 points)
- Mouth and lips (20 points)
- Jawline and chin (17 points)

These points provide a structural map of the face, enabling precise tracking of:

- Eye Aspect Ratio (EAR): To monitor eye openness and detect blinks or prolonged closure (a sign of drowsiness).

- Mouth Aspect Ratio (MAR): To detect yawning frequency and intensity.
- Head posture and movements: Using jaw and nose alignment.

5.1.3. Eye and Mouth Region Extraction

From the facial landmarks that have been detected, the coordinates of the eye and mouth areas are extracted. The left eye and right eye are generally denoted by six landmark points each, and the mouth by approximately 20 points. These areas are subsequently cropped out of the frame to create smaller patches of images, which are passed to the CNN to be classified or utilized for geometric calculations such as the Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR). These cropped images are also rescaled and transformed into grayscale or normalized if necessary for uniform input to the CNN.

Once facial landmarks are successfully identified, specific regions of interest—namely the eyes and mouth—are isolated using their corresponding landmark coordinates. Typically, the left and right eyes are defined by 6 points each (landmarks 36–41 and 42–47), and the mouth region is defined by 20 points (landmarks 48–67). These coordinate points are used to extract precise bounding boxes around the eyes and mouth, which are then cropped from the original frame to form smaller, focused image patches as shown in fig5.3 These regions reduce computational load and allow the model to concentrate on features most relevant to drowsiness detection.

To ensure consistent input for the neural network, the cropped patches are often resized (e.g., 64x64 pixels) and converted to grayscale, which simplifies the model by reducing channel complexity while preserving essential spatial features.. These preprocessed patches are either fed directly into a Convolutional Neural Network (CNN) for classification (Drowsy or Not Drowsy), or used in real-time calculations of behavioral metrics like EAR (Eye Aspect Ratio) to detect blinking or eye closure, and MAR (Mouth Aspect Ratio) to identify yawning. This precise and localized region extraction helps the model learn subtle temporal and spatial variations that signal driver fatigue.

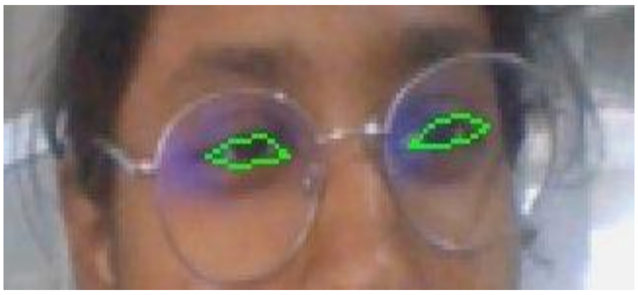


Fig 5.3 Eye and Mouth Region Extraction

5.1.4. Calculation of EAR and MAR

The system identifies eye closure using the Eye Aspect Ratio (EAR). EAR is determined by vertical and horizontal distances between certain eye landmarks. A persistent low EAR across a series of frames is an indicator that the eyes are closed, and this is an indication of drowsiness. The Mouth Aspect Ratio (MAR) is computed to identify yawning as well shown in fig 5.4. If the MAR remains high across multiple frames, it can be inferred that the person is yawning—another strong indicator of fatigue. These ratios are threshold-based and tuned experimentally for accuracy.

To improve reliability, both EAR and MAR thresholds are calibrated based on individual differences and environmental conditions. Combining EAR and MAR allows for a more comprehensive detection of drowsiness symptoms beyond just eye closure.
This approach is lightweight, real-time, and suitable for integration with camera-based monitoring systems in vehicles or workplaces. If the MAR remains high across multiple frames, it can be inferred that the person is yawning—another strong indicator of fatigue. These ratios are threshold-based and tuned experimentally for accuracy. Additionally, the duration and frequency of yawns over time can be used to quantify fatigue levels more reliably.

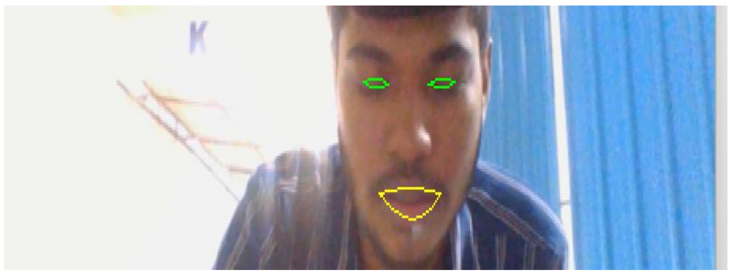


Fig 5.4 EAR and MAR Calculations

5.1.5. Eye State Classification Using CNN

To improve the reliability of drowsiness detection, a Convolutional Neural Network (CNN) model is trained to classify the eye state (open or closed). The CNN is provided the preprocessed and cropped eye images and returns a binary output. This is particularly beneficial in situations where the use of geometric approaches will be defeated, e.g., with glasses, low light, or partial occlusion. The CNN's architecture consists of multiple convolutional layers, pooling layers, and fully connected layers, finally leading to a softmax or sigmoid output layer.

To enhance the robustness and precision of drowsiness detection, especially in challenging real-world conditions, a Convolutional Neural Network (CNN) is employed to classify the eye state as either open or closed. This classification plays a critical role in determining whether a user is alert or exhibiting early signs of fatigue, such as prolonged eye closure—a key indicator of drowsiness

5.1.6. Drowsiness Detection Logic

The system integrates the results of EAR, MAR, and the CNN classifier through a rule-based decision engine. For example, if the EAR is less than a specified threshold for more than a specified number of consecutive frames, and the CNN also indicates closed eyes, the system determines that the driver is drowsy. Likewise, repeated yawning indicated by MAR values above a threshold also adds to the drowsiness score. The system employs a counter to monitor for how long these conditions hold, minimizing false positives from blinking or speaking. The system integrates the results of EAR, MAR, and the CNN classifier through a rule-based decision engine as shown in the fig 5.5. For example, if the EAR drops below a defined threshold (e.g., 0.25) for a continuous number of frames (e.g., 30), and the CNN model confirms closed eyes, the driver is flagged as drowsy. Similarly, a high MAR value (e.g., > 0.6) across multiple frames is used to detect repetitive yawning, contributing to a cumulative drowsiness score. Likewise, repeated yawning indicated by MAR values above a threshold also adds to the drowsiness score. The system employs a counter to monitor for how long these conditions hold, minimizing false positives from blinking or speaking.

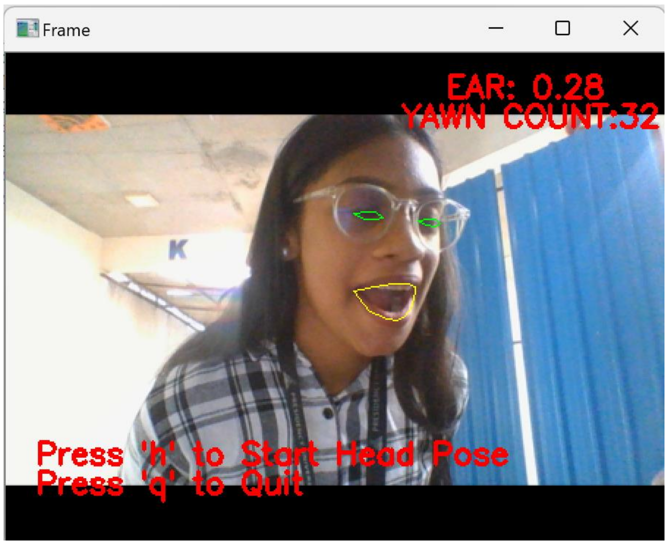


Fig 5.5 Drowsiness Detection

5.1.7. Alert Mechanism

After detecting drowsiness according to the specified logic, the system issues alerts to alert the driver. The warning mechanism can be in the form of screen prompts (e.g., "Drowsiness Detected"), voice reminders through speakers (e.g., beep or voice alert), or vibration feedback if coupled with the hardware (e.g., vibrating steering wheels). This is managed in Python by using the straightforward multimedia libraries of pygame or playsound for sound and cv2.putText() for screen feedback on the video frames.

After detecting drowsiness, the system immediately issues alerts to warn the driver as shown in fig 5.6, alerts can include on-screen prompts like “Drowsiness Detected” using cv2.putText(). Auditory feedback is provided through beeps or voice alerts using libraries like pygame or playsound.. If integrated with hardware, vibration feedback (e.g., in steering wheels) can also be triggered.

Once the system determines that the driver is drowsy based on behavioral indicators such as prolonged eye closure (via EAR), excessive yawning (via MAR), or abnormal head movements, it immediately triggers a real-time alert to notify the driver. The purpose of this alert system is to intervene quickly and prevent accidents due to inattention or microsleep episodes. Visual alerts are generated directly on the video feed using cv2.putText() from OpenCV, displaying clear and prominent messages like “Drowsiness Detected!” in red or bold fonts. This on-screen notification is synchronized with the video display as shown in fig5.6 so the driver sees the warning in real-time.

In addition to visual cues, auditory alerts provide a more intrusive form of feedback to ensure the driver is re-engaged with the driving task. These include short warning beeps, custom alarm tones, or voice messages such as “Wake up, you're drowsy!” played using Python libraries like pygame. mixer or play sound. These libraries are lightweight and allow for easy integration of audio playback with minimal system delay. For vehicles equipped with advanced hardware, the system can also trigger haptic feedback, such as vibration in the steering wheel or driver’s seat, using external microcontrollers like Arduino or Raspberry Pi, which can be interfaced through Python’s serial communication libraries. This multi-modal alert approach ensures that the system remains effective even if the driver misses one form of feedback due to distraction, noise, or fatigue level.

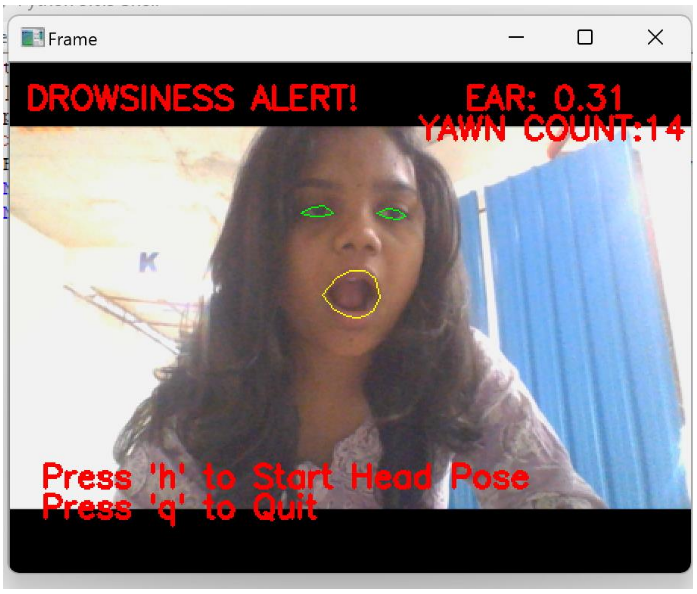


Fig 5.6 Alert Sound and Message

5.1.8. System Performance and Optimization

To achieve real-time performance, the system is optimized by lowering frame resolution, multithreading video capture and processing, and minimizing the CNN model. Key performance indicators like frames per second (FPS), CPU/GPU usage, and memory usage are tracked to ensure proper functioning of the system on normal hardware. Further, the CNN can be quantized or compiled to formats such as TensorFlow Lite for deployment on embedded devices.

Achieving real-time performance is crucial for any driver drowsiness detection system to be effective in real-world scenarios. Delays in processing or lag in response can lead to late alerts, thereby defeating the primary safety objective. To ensure the system runs smoothly on standard computing hardware and is capable of delivering consistent performance.

5.1.9. Scalability and Integration

Although this design is centered around standalone functionality with Python, the system can be scaled for real-world use by integrating with in-car infotainment systems or IoT platforms. It can be implemented on Raspberry Pi, NVIDIA Jetson Nano, or other edge devices. The detection logic can also be modified to accommodate cloud-based dashboards for fleet tracking or driver behavior analysis over time.

Optimization Strategies

1. **Lowering Frame Resolution**

- Input video frames are resized to a lower resolution (e.g., from 1920×1080 to 640×480) before processing.
- This reduces the amount of data per frame, speeding up both detection and classification stages.
- Despite the lower resolution, critical facial features like eyes and mouth are still detectable due to their relative size in the face region.

2. **Multithreading Video Capture and Processing**
    - Using separate threads for frame capture and processing ensures that reading from the camera does not block the model's inference.
    - This parallel approach enhances the system's responsiveness and increases the effective frame rate.
    - Python's threading or multiprocessing libraries, or OpenCV's built-in multithreaded capture, are typically used

CHAPTER 6
IMPLEMENTATION

6.1 Environment Setup and Library Installation

Implementation starts with creating the Python programming environment. That mostly involves installation of major libraries for computer vision, facial landmark detection, and deep learning. These essential libraries are OpenCV (for handling videos and images), Dlib (for detection of faces and facial landmarks), TensorFlow/Keras or PyTorch (for deep learning model implementation), and NumPy (for numerical computing). Packages like imutils, playsound, and pygame also are installed for the purpose of preprocessing and sending an alert.

The initial phase of implementing the drowsiness detection system involves setting up the Python programming environment and installing the necessary libraries for effective operation. The following libraries and packages are crucial for the system's **functionality**

- **OpenCV**
  OpenCV (Open Source Computer Vision Library) is used for video capture, image processing, and feature extraction. It allows real-time image processing and supports various video formats. OpenCV is used to extract frames from the video stream, detect faces, and perform initial image transformations such as resizing, cropping, and filtering.

- **Dlib**
  Dlib is a powerful library for machine learning and computer vision, specifically known for its facial landmark detection capabilities. It helps detect key points on the face (such as eyes, nose, and mouth) and is used to calculate the Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR), which are essential for identifying drowsiness symptoms. Dlib's accuracy in landmark detection ensures that even in challenging conditions, such as partial face occlusion or glasses, the system can still function effectively.

- **Keras-or-PyTorch**
  For deep learning-based facial state classification, TensorFlow/Keras or PyTorch are employed. These frameworks are used for building, training, and deploying Convolutional Neural Networks (CNNs) that can accurately classify whether a person’s eyes are open or closed. TensorFlow and PyTorch offer powerful tools for

model training, optimization, and inference, and they are widely used for computer vision tasks.

- **NumPy**
  NumPy is a fundamental library for numerical computing in Python. It is used extensively for handling arrays and matrices, which are crucial for processing image data. Operations such as normalization, reshaping, and pixel-level computations are performed using NumPy.
- **Imutils**
  Imutils is a lightweight library that simplifies various computer vision tasks. It provides utility functions for resizing, rotating, and transforming images, which are necessary for preparing data for the drowsiness detection model.
- **Playsound**
  Playsound is a simple library that plays sound files. It is used for triggering auditory alerts when drowsiness is detected, ensuring that the user receives timely notifications.
- **Pygame**
  Pygame is used to create the visual alerts and notifications in the form of pop-up messages, on-screen visual cues, or flashing warnings. It is particularly useful for creating attention-grabbing notifications during drowsiness detection.

```
pip install opencv-python dlib tensorflow keras imutils playsound
```

6.2 Real Time Video Capture with OpenCV

With OpenCV, the system opens a video stream from the webcam. The frames of the video feed are read in real time with cv2.VideoCapture(0). The frames are resized and converted to grayscale to save processing time and enhance the performance of the face detection algorithm. This real-time video loop is the core of the system, allowing real-time observation of the driver's face.

OpenCV's video stream handling plays a crucial role in capturing the real-time feed from the webcam. The system initiates the video stream using the cv2.VideoCapture(0) method, where 0 refers to the default webcam on the system. This opens the camera and starts the live feed, providing a continuous stream of frames that can be processed by the system.

Once the video feed is initialized, the frames are read continuously inside a loop using cap.read(). This function returns two values: a boolean indicating whether the frame was successfully captured and the actual frame itself.

The following steps are involved in processing the frames:

- **Resizing**

  The frames are resized to a smaller resolution to optimize processing speed. Typically, the frame size is reduced to 640x480 or 320x240 pixels. Reducing the frame resolution not only reduces computational load but also helps maintain a smooth frame rate (FPS), ensuring the system

- **Converting-greyscale**

  The frames are converted to grayscale using cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY). This step eliminates the color channels, leaving only intensity information. Since facial detection relies on contrast and shape rather than color, grayscale images simplify the task for algorithms like Dlib's face and landmark detectors, speeding up the process without losing essential information.

- **Face Detection**

  Once the frame is prepared, the next step is to detect the driver's face. OpenCV’s cv2.CascadeClassifier() is used for this purpose. This classifier utilizes a pre-trained Haar cascade model to detect faces in the video frame. The face detection process returns a bounding box around the face, which is essential for isolating facial features like the eyes and mouth.

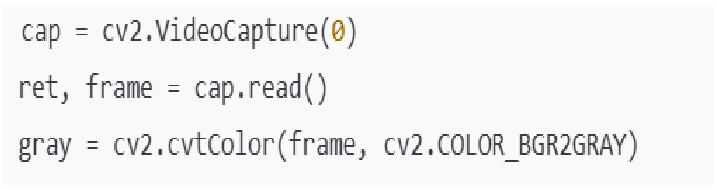
- **RIO**

  Once a face is detected, the system extracts the region of interest (ROI), focusing on specific areas like the eyes and mouth. This helps in reducing the data that needs to be processed and increases the overall efficiency of the detection algorithm.

- **ProcessingAlerts**

  After detecting the face and relevant facial landmarks (like eyes and mouth), the system applies the drowsiness detection algorithms (such as EAR and MAR). If drowsiness is detected based on these metrics or CNN predictions, the system triggers an alert (visual or auditory) to warn the driver.

- **Displaying-Frame**
  After each frame is processed, the system displays the frame with annotations (like bounding boxes around faces or alert messages) using cv2.imshow(). This function continuously updates the window with the live video feed and allows the user to see real-time detection results.
- **Real-time-Frame-Processing**
  The entire process runs inside a loop, processing each video frame in real time. The loop continues until the user manually closes the video window or the system detects a specific exit condition (such as pressing the 'q' key).



```
cap = cv2.VideoCapture(0)
ret, frame = cap.read()
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

6.3 Face Detection and Facial Landmarking Extraction

Dlib's face detector is run on the grayscale frame to find the driver's face. Upon detecting a face, the 68-point facial landmark predictor is used to overlay important facial features. This model is initialized through Dlib's shape_predictor_68_face_landmarks.dat, and it gives coordinates for the eyes, mouth, and other facial structures. These landmarks are essential in the computation of EAR and MAR.

Face Detection and Facial Landmark Extraction (Dlib)

Dlib’s facial landmark detection plays a pivotal role in extracting key facial features necessary for detecting drowsiness signs. Once a face is detected in the grayscale frame using OpenCV, Dlib's powerful face detector and shape predictor are used to extract and annotate facial landmarks, which are essential for accurate drowsiness detection.

Face Detection Using Dlib

Dlib provides a robust face detection algorithm based on Histogram of Oriented Gradients (HOG) and a linear classifier. When a frame is passed through Dlib’s face detector, it returns bounding boxes around the detected faces. This model is fast and accurate, even under varying lighting conditions, making it suitable for real-time applications.

The face detection is performed on the grayscale image, as grayscale images contain less information but still retain essential features such as the contrast between the face and background. This increases efficiency and reduces computational load.After detection, the system iterates over the detected faces to apply further processing, such as landmark extraction.

Facial Landmark Detection Using Dlib's Shape Predictor

Once the face is detected, Dlib's shape predictor is used to identify 68 key facial landmarks. These landmarks are specific points on the face that define the boundaries of facial features such as the eyes, eyebrows, nose, and mouth. Dlib’s shape_predictor_68_face_landmarks.dat file, which contains the pre-trained model for detecting facial landmarks, is used for this task.

Extracting Important Facial Features for EAR and MAR Calculation

The 68 facial landmarks include key points around the eyes and mouth. These points are used for calculating the Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR), which help determine whether the driver is drowsy.

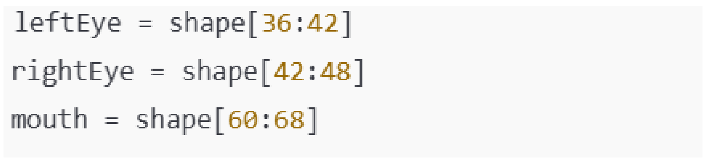
- Eyes: The landmarks around the eyes (e.g., landmarks 36–41 for the left eye and 42–47 for the right eye) are used to calculate the EAR, which detects whether the eyes are closed for prolonged periods, a key indicator of drowsiness.
- Mouth: The landmarks around the mouth (e.g., landmarks 48–59) are used to calculate the MAR, which detects yawning, another significant sign of fatigue.

```
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
```

6.4 Eye and Mouth Region Localization

After landmark extraction, certain coordinates are utilized to isolate the left eye, right eye, and mouth. These regions are cropped from the original frame and stored for subsequent processing. The cropped eye regions are also utilized as inputs to the CNN classifier, while the coordinates are utilized for EAR and MAR calculation as shown in the code.

After landmark extraction, certain coordinates are utilized to isolate the left eye, right eye, and mouth. These regions are cropped from the original frame and stored for subsequent processing. The cropped eye regions are also utilized as inputs to the CNN classifier, while the coordinates are utilized for EAR and MAR calculation by analyzing both pixel-level and shape-based features, the system enhances the overall accuracy and robustness. The code selects facial landmarks for the left eye (points 36–41), right eye (42–47), and mouth (60–67) from a 68-point facial landmark array. These points are used in drowsiness detection to track eye closure (via Eye Aspect Ratio) and yawning, helping identify signs of driver fatigue. The cropped eye regions are also utilized as inputs to the CNN classifier, while the coordinates are utilized for EAR and MAR calculation as shown in the code.



```
leftEye = shape[36:42]
rightEye = shape[42:48]
mouth = shape[60:68]
```

6.5 Calculation of EAR and MAR

With Euclidean distances between the eye landmarks, the Eye Aspect Ratio (EAR) is derived. A low EAR over an extended number of frames (for example, 48 frames) implies that eyes are probably shut. Likewise, Mouth Aspect Ratio (MAR) is calculated in order to spot yawning. Both these figures are utilized cumulatively in order to make a decision on drowsiness. These operations are performed frame by frame so that continuous levels of fatigue may be tracked.

The Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR) are crucial indicators in the drowsiness detection system, as they provide quantifiable measures of eye and mouth states, helping to identify potential signs of fatigue. Both of these measures are computed based on facial landmarks detected by Dlib, which are then used to analyze the subject's state and determine whether drowsiness is occurring.

1. Eye Aspect Ratio (EAR)

The Eye Aspect Ratio (EAR) is a metric used to determine whether a person's eyes are open or closed. It is calculated by measuring the Euclidean distances between specific points around the eyes. Dlib's facial landmark detector provides the coordinates for these points.

$EAR=2\times|p3-p6||p1-p5|+|p2-p4|$

Where:

- p1,p2,p3,p4,p5,p6p_1, p_2, p_3, p_4, p_5, p_6p1,p2,p3,p4,p5,p6 are specific landmark points surrounding the eye.
  - p1,p5p_1, p_5p1,p5: Horizontal landmarks of the eye (corners of the eye).
  - p2,p4p_2, p_4p2,p4: Vertical landmarks (top and bottom eyelids).
  - p3,p6p_3, p_6p3,p6: The mid-points of the vertical axis of the eyelids.

When a person blinks or closes their eyes, the EAR decreases. By continuously monitoring the EAR across multiple frames, the system can detect if the eyes remain closed for a sustained period of time, a common sign of drowsiness.

Thresholding for Drowsiness Detection

- A threshold value is set (usually around 0.2) to determine when the eyes are closed.
- If the EAR value falls below this threshold for a specified number of consecutive frames (e.g., 48 frames), the system flags it as drowsiness.

2. Mouth Aspect Ratio (MAR)

The Mouth Aspect Ratio (MAR) helps to detect yawning, which is another key indicator of fatigue. It is calculated by measuring the Euclidean distances between specific points on the mouth. The landmarks used to calculate the MAR are located on the corners of the mouth, and the upper and lower lip.

$MAR=|p4-p8|\ \ |p2-p10|$

Where:

- p2,p10p_2, p_10p2,p10: Landmarks representing the mouth corners.
- p4,p8p_4, p_8p4,p8: Points at the top and bottom of the mouth.

When a person yawns, the mouth opens wide, increasing the distance between the upper and lower lips. By comparing the distances between these landmarks, the system calculates the MAR. A MAR above a certain threshold (e.g., 0.6) indicates a yawn.

3. Combining EAR and MAR for Drowsiness Detection

The system uses both EAR and MAR to make a comprehensive decision about whether the user is drowsy. Instead of relying on a single indicator, the system combines both metrics, ensuring better detection accuracy.

- EAR: Detects whether the eyes are closed for an extended period.
- MAR: Detects yawning, another indicator of fatigue.

If either the EAR or MAR crosses the respective threshold for a significant number of frames, the system triggers an alert, indicating the driver is potentially drowsy.

4. Frame-by-Frame Calculation for Continuous Monitoring

Since the drowsiness detection system works in real time, these calculations are performed on every frame captured by the webcam. Each frame is processed to extract facial landmarks, calculate EAR and MAR, and check for drowsiness signs. Continuous analysis ensures that any moment of drowsiness is detected quickly, providing real-time feedback to the driver. Formula we use for calculating the EAR .

The system tracks these metrics over time, and by doing so, it can give early warnings even before the drowsiness becomes severe. If the EAR drops too low or the MAR rises above a specific threshold, the system will issue an alert.

To further enhance the robustness of the drowsiness detection system, temporal smoothing and weighted scoring mechanisms can be introduced. These approaches help balance sensitivity and specificity, minimizing false alarms due to brief actions like blinking or talking. For instance, a weighted drowsiness score can be maintained where each instance of low EAR or high MAR increases the score, and normal behavior gradually decreases it. If the score exceeds a predefined threshold, a drowsiness alert is triggered. Additionally, combining EAR and MAR with contextual cues such as head nodding or prolonged gaze deviation can lead to a multi-modal detection system, increasing reliability even in complex driving environments. This kind of adaptive logic ensures that the system maintains high accuracy under various facial expressions and lighting conditions, improving its effectiveness for real-world deployment.

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

6.6 CNN Based Eye State Classification

For making the system more robust against false positives and lighting, a CNN is used to classify if eyes are open or closed. A dataset (e.g., CEW – Closed Eyes in the Wild) is employed for training the model. The architecture of CNN comprises convolutional layers followed by the pooling layers and fully connected layers. After being trained, the model is loaded and utilized real-time to estimate eye state by utilizing the cropped eye images in the video stream.

Purpose and Role in Drowsiness Detection

The core function of the CNN in this system is to determine whether the driver's eyes are open or closed in each video frame. This binary classification plays a crucial role in calculating the Eye Aspect Ratio (EAR) or detecting prolonged eye closures, which are key indicators of drowsiness.

By relying on CNN predictions instead of just landmark-based calculations, the system can:

- Handle poor lighting and varying head orientations more robustly.
- Reduce false positives caused by eye squinting or shadow effects.
- Work even when the driver is wearing glasses or in partially occluded environments.

To ensure real-time performance, the trained CNN model is optimized and deployed using lightweight architectures such as MobileNet, or pruning techniques can be applied to reduce model complexity without sacrificing accuracy. During runtime, cropped eye regions from each video frame are resized (e.g., to 24×24 or 64×64) and normalized before being passed into the CNN. The model then outputs a probability score indicating whether the eyes are open or closed. This prediction is integrated with the rule-based logic of EAR and MAR to form a hybrid system, enhancing decision confidence. Additionally, incorporating CNN-based classification allows the system to maintain accuracy even in noisy backgrounds, motion blur, or low-resolution video, making it more suitable for in-vehicle applications where environmental conditions constantly change.

```
from keras.models import load_model
model = load_model('cnn_eye_model.h5')
prediction = model.predict(processed_eye_image)
```

6.7 Decision Logic for Drowiness Detection

The system employs a counter-based reasoning in which it monitors how many continuous frames the driver's eyes are closed (from EAR and CNN outputs). If the number of counts exceeds a predetermined threshold (e.g., 48 frames, equivalent to 2 seconds at 24 FPS), a drowsiness warning is issued. MAR is also verified, and repeated yawning can be included in the overall drowsiness score. By using rule-based and model-based inputs together, the system becomes more robust.

The decision-making process in the drowsiness detection system is structured to be both adaptive and accurate. The system uses a hybrid approach combining rule-based thresholds and deep learning model outputs to determine the driver’s drowsiness status. This approach ensures robustness in detecting drowsiness even in real-world, variable conditions.

1. Frame Count and Thresholding

The primary decision logic relies on the continuous monitoring of the driver's facial state over time. By tracking the Eye Aspect Ratio (EAR) and using outputs from the CNN classifier (for eye-state classification), the system counts how many frames show the eyes in a closed state. This counter-based system helps to detect prolonged eye closure, a key indicator of drowsiness. The system is designed to trigger a warning when the following conditions are met:

- Threshold for Continuous Eye Closure (EAR-based): If the number of consecutive frames with an EAR below the set threshold (typically 0.2) exceeds a predefined count (e.g., 48 frames), the system flags the user as potentially drowsy. At 24 frames per second (FPS), 48 frames equates to about 2 seconds, a sufficient amount of time to indicate prolonged eye closure.

2. Yawning Detection (MAR-based)

In addition to monitoring eye closure, the system also tracks the Mouth Aspect Ratio (MAR) to detect yawning, another common sign of drowsiness. The system checks if the MAR exceeds a certain threshold (typically 0.6), indicating that the user is yawning. If yawning occurs repetitively over a set number of frames or if a certain number of yawns are detected during the session, it further reinforces the drowsiness alert.

- Threshold for Yawning Detection (MAR-based): If the MAR is greater than the set threshold for a defined number of consecutive frames (e.g., 3 yawns within 10 seconds), the system recognizes the driver as fatigued. This can be combined with EAR measurements for higher accuracy.

3. Hybrid Decision Model

The hybrid decision logic combines EAR and MAR results. Instead of relying on one indicator, both metrics are used to form a more comprehensive assessment of drowsiness. The system can issue an alert when either:

- The EAR threshold for prolonged eye closure is exceeded.
- The MAR threshold for repeated yawning is surpassed.
- This dual approach ensures that the system doesn’t misinterpret short-term eye closures (e.g., brief blinks) or single yawns as signs of drowsiness. Instead, it requires sustained eye closure or multiple yawns to trigger a warning, reducing false positives.

Additionally, the system is designed to consider both the rule-based approach (EAR and MAR thresholds) and model-based outputs (from CNN) to adapt to different user profiles, such as varying facial characteristics (e.g., people with glasses or different skin tones). If the CNN model detects the eye as closed even in cases where EAR might be unreliable due to lighting conditions or occlusions, it strengthens the overall drowsiness detection.

4. Temporal Consistency and Alert Issuance

Once the system detects potential drowsiness based on the defined criteria, it issues an alert. However, the alerting mechanism is designed to avoid false positives by using temporal consistency. For example:

The system checks that the eye closure and yawning conditions are consistent over multiple frame1s. If the EAR value falls below the threshold for more than 2 seconds and the MAR exceeds the yawning threshold multiple times in that period, it triggers an alert.

5. Customization and Calibration

To further improve the decision-making process, the system can be calibrated and customized according to the user's behavior or environmental factors:

Individual Calibration: The thresholds for EAR and MAR can be adjusted for different users, as some people may naturally have a lower EAR even when their eyes are open or may not yawn as frequently. Fine-tuning these parameters for specific users can help the system adapt to diverse conditions.

- Environmental Calibration: Adjusting thresholds based on lighting conditions or face occlusion (e.g., glasses, masks) can help mitigate false negatives or positives.

```
if EAR < EAR_THRESHOLD:
    eye_closed_counter += 1
    if eye_closed_counter >= EAR_CONSEC_FRAMES:
        alert = True
else:
    eye_closed_counter = 0
```

6.8 Alert Generation

After detecting drowsiness, an alert system is triggered to inform the driver. This involves:

- **Visual Alert:** A warning message on the frame via OpenCV.
- **Audio Alert:** An auditory (e.g., beep or voice warning) played via the playsound or pygame module.

This two-mode alert ensures a quick grab of the driver's attention to avoid potential accidents.

User Alert Mechanism

Once drowsiness is detected, the system activates both visual and auditory alerts to ensure that the driver is warned in time. The visual alert appears as a prominent notification on the screen, and the auditory alert produces a sound to draw attention. The system might also provide a prompt to the driver (e.g., "Take a break or pull over") to help prevent potential accidents.

In conclusion, the decision logic for drowsiness detection is a multi-layered process that combines facial geometry-based rule detection (EAR and MAR) with deep learning outputs (CNN-based eye detection). This hybrid system enhances robustness and reduces the risk of false positives, providing a more accurate and reliable drowsiness detection solution.

```
cv2.putText(frame, "DROWSINESS ALERT!", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
playsound('alarm.wav')
```

6.9 System Loop and Real Time Optimization

The whole process is wrapped within a loop that processes one frame at a time in real time. Optimization for performance includes:

- Decreasing video frame resolution to speed up processing.
- Employing multi-threading for frame capture and model inference.
- Restricting the rate of CNN calls by employing them only when EAR falls.
- This enables the system to operate efficiently even on low-processing power devices.

To achieve real-time processing of drowsiness detection, the system is designed to continuously monitor the driver’s facial features frame by frame. To ensure the system operates efficiently even on low-processing power devices, several optimization techniques are employed. These optimizations focus on reducing the computational load while maintaining the system’s ability to deliver timely alerts.

1. Video Frame Resolution Adjustment

One of the primary optimization strategies involves decreasing the video frame resolution. Processing high-resolution video frames can significantly slow down the system, especially on devices with limited processing power. By reducing the resolution (e.g., downsampling to 640x480 or 320x240 pixels), the amount of data to be processed per frame is minimized, speeding up both face detection and facial landmark extraction. This is especially beneficial when the system is processing video frames at high frame rates.

Frame Resolution Adjustment: Lowering the resolution not only speeds up processing but also reduces memory consumption, helping to ensure that the system performs optimally without causing system crashes or slowdowns.

2. Multi-Threading for Performance Enhancement

To further optimize performance, multi-threading is used for handling the separate tasks of frame capture and model inference. By employing separate threads for these operations, the system can process video frames and execute the deep learning model concurrently, improving the overall throughput of the system.

- **Frame Capture Thread:** Captures frames from the webcam in real time and passes them to the next stage.
- **Model Inference Thread:** Handles the processing of each frame (face detection, facial landmark extraction, EAR and MAR calculations, and drowsiness detection) and decision-making, running in parallel to the frame capture process.

3. Optimizing CNN Inference Calls

To minimize computational overhead, CNN inference calls are restricted. Instead of running the deep learning model on every single frame, the system only invokes the CNN when the Eye Aspect Ratio (EAR) falls below a certain threshold, indicating that the eyes are likely closed or there may be a potential drowsiness situation.

- Conditional CNN Calls: The CNN is only invoked if EAR drops below a predefined threshold (e.g., EAR < 0.2), indicating that further analysis is needed.

4. Frame Skipping for Reduced Load

In some cases, skipping frames can help reduce the computational load while maintaining real-time performance. If the system detects that the current frame is not significantly different from the previous one (e.g., no significant change in facial landmarks), it may skip processing that frame, opting to process the next one. This reduces redundant computations without sacrificing system performance.

- **Frame Skipping Logic:** This can be implemented by tracking the differences between consecutive frames and processing only those frames that show notable changes in the driver's facial state.

6.10. Testing and Validation

Once the system is built, it is tested with different users under various lighting and head positions. Metrics such as accuracy of detection, false alarm rate, and reaction time are measured. Based on testing results, thresholds and model parameters are fine-tuned to improve performance and reduce false positives or missed detections.

Once the drowsiness detection system is developed and integrated, comprehensive testing and validation are critical to ensure it functions reliably in real-world scenarios. This stage confirms that the system performs well under diverse conditions, minimizes false alarms, and responds promptly to signs of drowsiness.

Testing Scenarios

To simulate realistic use cases, the system is tested under the following controlled and uncontrolled scenarios:

- **Lighting Conditions**
  - Daylight, low-light, and artificial lighting.
  - Bright backlighting and shadowed environments.
- **Head Movements and Angles**
  - Slight tilts, turns, and movement during driving.
  - Eye visibility at different angles

CHAPTER 7
TESTING

7.1 Aim

The primary goal of the testing stage is to analyze the effectiveness, accuracy, and reliability drowsiness detection system across various real-world scenarios. Testing allows for determining how effectively the system identifies drowsy activities like eye closure and yawning, how the system functions with varied users, and whether the system can consistently provide timely alerts. This stage guarantees that the system not only functions in test environments but also performs well under variable and uncertain conditions.

The testing stage also aims to evaluate the system's robustness in handling diverse external factors that could affect its performance. For instance, the system is assessed under different lighting conditions (e.g., bright daylight, dim environments, or low light), various head poses, and even facial obstructions such as glasses, hats, or beards. These factors can introduce challenges like occlusion or altered facial landmarks, which may cause traditional facial feature extraction methods to fail. The goal is to ensure that the system can adapt and still accurately detect drowsiness signals despite these challenges. It is crucial that the system performs reliably across a wide range of scenarios to provide a consistent user experience.

Additionally, the aim of testing extends to evaluating the system's ability to minimize false positives and false negatives. False alarms could be distracting to the driver and lead to decreased trust in the system, while missed detections could have more severe safety implications. Therefore, the testing phase also involves fine-tuning the detection thresholds and model parameters to achieve an optimal balance between sensitivity and specificity. The feedback obtained from real-world testing allows for iterative improvements, ensuring that the system is not only accurate but also practical and responsive in critical driving situations. By closely analyzing the test results, adjustments are made to improve the overall efficiency and effectiveness of the system.

7.2. Setting Up the Test Environment

We tested using an ordinary laptop containing a built-in webcam with full implementation of the drowsiness detection system that was coded using Python. Test subjects took part in test sessions, sitting directly in front of the webcam as if driving in simulated driving scenarios.

The testing environment conditions varied between tests to encompass:

- Regular daylight

- Poor light/night scenes
- Facial shadows or backlit facial features
- Test subjects wearing spectacles or wearing hats

This enabled the evaluation of the system's performance under various lighting conditions and face appearances. The test sessions were designed to mimic real driving scenarios. Volunteers participated in these tests by sitting at a fixed distance (~50–70 cm) from the webcam, emulating the typical position of a driver in front of a dashboard camera. The environment was kept quiet and free from distractions to focus solely on the system’s input-output behavior

further simulate real-world conditions, the test subjects were asked to perform a variety of actions during the test sessions, such as blinking, yawning, and shifting their gaze, in order to evaluate how well the system responds to natural driver behavior. Some subjects were also instructed to intentionally simulate signs of fatigue, like prolonged eye closure or exaggerated yawning, to test the system’s ability to detect these specific signs of drowsiness. The test was designed to cover different driver behaviors, including sudden head movements or brief moments of looking away from the camera, as these factors could influence the detection accuracy of the system.

To assess the robustness of the system, the test sessions were repeated under different environmental settings. For example, tests were conducted in both day and night scenarios, using artificial lighting or low-light environments to understand how the system handles variations in visibility and shadows. Subjects with glasses or hats were included to test the system’s performance in scenarios where facial features may be partially obstructed. These diverse conditions helped provide a comprehensive understanding of the system's strengths and limitations, and ensured that it can be deployed in real-world situations, where environmental variables are unpredictable.

7.3. Test Scenarios

For simulating realistic drowsiness events, test scenarios were planned as follows:

- Intentional blinking and prolonged eye closure to simulate sleepiness
- Yawning as a secondary sign of fatigue
- Normal alert behavior with open eyes and without yawning
- Distractions like looking away from the camera

For every situation, the system's predictions (i.e., "drowsy" or "not drowsy") were recorded and compared against the subject's actual behavior, which was recorded manually.

This assisted in computing important performance metrics.

In addition to these primary scenarios, some test subjects were asked to simulate rapid head movements or look at the camera from different angles to assess how well the system handles changes in head position or orientation. This also included testing the system's ability to maintain accurate drowsiness detection while the subject was wearing glasses or a hat, which could potentially obscure facial features. The system's ability to differentiate between true signs of drowsiness and other non-fatigue related behaviors (such as laughing or normal speech movements) was also tested to ensure that false positives were minimized. This variety of scenarios allowed for a comprehensive evaluation of the system's performance across different conditions, ensuring its robustness and reliability in detecting drowsiness in real-world driving situations.

7.4. Performance Metrics

A number of performance metrics were employed to test the system:

- Accuracy: Tracks how frequently the system accurately identified drowsiness or normality.
- False Positives: Those cases in which the system raised an alarm even when the subject was alert and awake.
- False Negatives: Cases in which the subject was obviously exhibiting the signs of drowsiness but the system could not pick it up.
- Latency: Duration between the occurrence of drowsiness symptoms and the production of the alarm.

For instance, the system correctly identified eye closure in approximately 92–95% of the instances, and the CNN eye-state classifier greatly minimized false negatives over employing EAR alone.

Additionally, precision and recall were used to evaluate the system's performance in a more balanced way. Precision refers to the proportion of drowsy alerts that were correct (true positives) out of all the alerts triggered by the system, while recall measures the ability of the system to correctly identify all true instances of drowsiness. The system's precision and recall scores were crucial in determining how well the system handled edge cases, such as when a user might be momentarily distracted but still showing subtle signs of drowsiness.

Another key metric was the Frames Per Second (FPS) during real-time processing, which directly impacts the system's ability to maintain smooth and consistent detection during normal driving conditions. Higher FPS rates ensured that the system could analyze frames quickly enough to make real-time decisions. Testing under various hardware specifications, including standard laptops and high-performance machines, provided insights into how the system could be optimized for broader deployment in both consumer-grade and professional settings.

7.5. Evaluation of the EAR and MAR Calculations

The EAR and MAR equations were individually tested to assess their sensitivity and reliability. The EAR measure performed well when the subject looked straight at the camera and had good lighting. Yet in low lighting or side viewing angles, the EAR by itself occasionally misidentified closure of the eyes. The MAR measure correctly identified yawning but required delicate threshold tuning to separate it from normal speech or mouth activity. These experiments established that although EAR and MAR are valuable, they perform best when complemented by the CNN classifier.

EAR (Eye Aspect Ratio) Evaluation

The EAR calculation was extensively tested by analyzing a variety of scenarios, including:

- Direct frontal gaze under normal lighting conditions, where the EAR metric demonstrated high consistency in detecting both open and closed eye states.
- In low-light conditions, the eye contours were not always well-defined, leading to minor fluctuations or misdetections, especially when shadows obscured the eye region.
- During side-angle views or when the head was slightly tilted, the accuracy of EAR dropped, particularly if one eye was partially occluded or distorted due to perspective shifts.
- Subjects wearing eyeglasses or anti-glare lenses occasionally caused incorrect landmark placement by Dlib/Mediapipe, slightly affecting EAR computation.

Despite these limitations, the EAR metric remained computationally efficient and fast, allowing for near-instantaneous feedback in ideal conditions.

MAR (Mouth Aspect Ratio) Evaluation

The MAR metric was tested by having subjects:

- Perform normal speech movements, yawn intentionally, and keep the mouth relaxed. The system logged MAR values throughout these actions.
- Yawning sequences consistently produced a distinct spike in MAR values, allowing reliable detection when an appropriate threshold was applied.

- However, MAR was also triggered by talking, chewing, or coughing, resulting in false positives unless the threshold was finely tuned or combined with contextual cues (e.g., prolonged mouth opening).
- Differences in facial structure and beard growth among users also slightly impacted MAR's reliability, making it less robust across diverse populations without further training.

7.6. CNN Model Testing

The CNN model was tested independently and in conjunction with the system. It was tested on a labeled test data set of eye images (closed and open). The CNN performed well in offline testing (~96%) and even in real-time when input with cropped eye images. Its greatest strength was particularly evident in those instances where the landmark detection by Dlib was marginally misplaced or when illumination was not even. This endorsed the advantage of employing a hybrid strategy—rule-based and deep learning-based.

Offline Testing Performance

In offline testing, the CNN model showed impressive results, achieving an accuracy of approximately 96%. The model was trained using a diverse set of eye images under different conditions (e.g., varying lighting, angle, and occlusion), which made it highly adaptable to real-world scenarios. The model's performance during offline testing was evaluated by comparing its predictions with the ground truth labels in the test dataset.

- Accuracy: The model's ability to correctly classify the eye state (open or closed) was strong across a range of scenarios, with 96% accuracy indicating a highly effective model.
- Generalization: The CNN demonstrated strong generalization capabilities, correctly identifying the eye state even when faces were slightly tilted or when the eyes were partially obscured by glasses or headgear.

7.7. Alert System Testing

The last phase of the tests involved sensitivity of the alert mechanism. When it sensed drowsiness, the system presented visual warnings (e.g., red warning messages) and a sound alarm. The visual warnings were tested for timing, audibility, and how well they could recapture the attention of a drowsy or distracted user. The sound warnings were verified to be loud and clear enough to be heard right away, even amid distractions.

Once drowsiness was detected, the system displayed a prominent red warning message (e.g., "Drowsy Detected! Please Take a Break") on the screen. The timing of the alert was carefully adjusted to ensure it appeared immediately after the detection, with a delay of no more than 1 second from the moment drowsiness was detected. The message was large enough to grab attention and clearly visible, even when the user was engaged in other tasks (like listening to music or talking).

In several test scenarios, distracted or partially alert users were involved. The system's effectiveness at recapturing attention was assessed by having the test subjects ignore the warning intentionally (e.g., continuing to look at the screen or avoid responding). It was observed that the duration of the visual warning (3–5 seconds) was long enough to bring the user's focus back to the screen, especially when coupled with an auditory signal

7.8. Long Duration and Real Time Performance Testing

The system was also tested for long-duration reliability by continuously running it for more than 30 minutes. Memory consumption, frame processing rate (FPS), and CPU usage were tracked. The system was stable, with a mean FPS of 15–20 on an average machine. These results proved that the program was optimized well enough to use in real time without needing high-end hardware.

Test Setup

The system was run for more than 30 minutes continuously, simulating a long driving session. During this time, the following key performance indicators were closely monitored:

- Memory Consumption: This is an essential factor for ensuring that the system can run without causing memory leaks or significant slowdowns over time.
- Frame Processing Rate (FPS): Since the system is designed for real-time drowsiness detection, the ability to process video frames efficiently was a priority. The FPS rate indicates how many frames are processed per second, directly affecting the system's responsiveness.
- CPU Usage: CPU usage was tracked to ensure that the system did not overburden the hardware, which would lead to crashes, slowdowns, or system unresponsiveness.

During the long-duration test, the system demonstrated consistent performance over the entire 30-minute session. The memory usage remained stable, with no signs of memory leaks or performance degradation. The system was able to run efficiently on a standard laptop, utilizing a reasonable amount of memory (less than 60% of available RAM) while maintaining responsiveness. This indicated that the system is suitable for continuous operation, which is crucial for real-world applications where drivers may rely on the system for extended periods.

The frame processing rate (FPS) was another critical factor tested during the long-duration test. With an average FPS ranging between 15–20 on an average machine, the system maintained a smooth video analysis pipeline. While this FPS may seem modest compared to high-end gaming applications, it was sufficient for drowsiness detection, as the system did not require high-resolution frames or intricate facial features to make accurate predictions. Additionally, CPU usage remained relatively low throughout the test, with no significant spikes, indicating that the system's design was well-optimized for real-time performance on standard hardware. This performance monitoring ensured that the system could operate effectively even under long-term use, which is essential for real-world deployment in a driving context.

7.9. User Feedback and Observations

Test session participants gave qualitative feedback. The majority found the alerts from the system to be useful and timely. A few users who wore glasses mentioned that the system sometimes had difficulty detecting eye state accurately, and that more fine-tuning or increasing the dataset would enhance performance. Users also recommended adding head tilt or nod detection in subsequent versions to make it more reliable.

After completing the test sessions, participants provided valuable qualitative feedback that helped identify areas for improvement and areas where the system performed well.

Positive Feedback

The majority of test subjects found the alert system—both visual and auditory warnings—to be timely and effective in grabbing their attention. Many participants appreciated the real-time nature of the system, with immediate notifications when drowsiness was detected. The sound alarms were particularly noted for being loud enough to be heard even when users were distracted, and the visual alerts were effective in drawing attention when they appeared on the screen.

Several participants noted that the system increased their awareness about their fatigue levels and felt that it would be a useful feature in real-world scenarios, especially for long-distance drivers or individuals working on tasks that required sustained focus.

Positive-Feedback

The majority of test subjects found the alert system—both visual and auditory warnings—to be timely and effective in grabbing their attention. Many participants appreciated the real-time nature of the system, with immediate notifications when drowsiness was detected. The sound alarms were particularly noted for being loud enough to be heard

even when users were distracted, and the visual alerts were effective in drawing attention when they appeared on the screen. The alerts were perceived as clear and unambiguous, offering enough time for the user to react before the situation could become critical.

Several participants noted that the system increased their awareness about their fatigue levels and felt that it would be a useful feature in real-world scenarios, especially for long-distance drivers or individuals working on tasks that required sustained focus. The combination of visual and auditory signals was praised for providing a multi-layered alert system that worked well in various environments, from quiet settings to noisy or distracting environments. A few participants even mentioned that they could see the system being helpful not only for driving but also for applications like office work or long study sessions, where focus and attention are critical.

Areas-for-Improvement

Although the majority of feedback was positive, a few test subjects, particularly those wearing glasses, mentioned that the system sometimes struggled to accurately detect their eye state. This was especially true when the glasses created reflections or occlusions that hindered the facial landmarks. Some of these participants suggested that the system could be improved by expanding the dataset to include more diverse face types and lighting conditions, which would help it handle these cases more effectively. Additionally, users recommended incorporating additional sensors or algorithms to detect head tilt or nods as supplementary indicators of drowsiness, as this could further improve accuracy in detecting fatigue when the eyes alone might not provide sufficient information.

In particular, those who wore hats or had their faces partially obscured also reported that the system occasionally failed to detect key facial features or misinterpreted their alertness. This feedback highlighted the need for the system to be more robust to diverse user conditions, including varying facial attributes and different headgear. Participants were also interested in a more customized alert system that could adjust based on the user's preferences, such as alert thresholds for different levels of drowsiness or an option for vibration alerts for situations where audio warnings might not be effective.

7.10. Summary of Testing Results

In general, the test phase proved that the system of drowsiness detection is capable of effectively detecting fatigue symptoms through a combination of facial geometry (EAR and MAR) and deep learning (CNN classifier). Although the system works well in most typical situations, there is space for optimization in edge situations such as extremely poor lighting, fast head motion, or partial face occlusion. However, the hybrid detection reasoning and real-time alert mechanism render it a realistic and functional prototype for driver safety software.

The testing phase of the drowsiness detection system demonstrated that the system was capable of effectively identifying fatigue symptoms in real-time through a combination of facial geometry-based features (Eye Aspect Ratio - EAR and Mouth Aspect Ratio - MAR) and deep learning techniques (Convolutional Neural Network - CNN classifier). These methods allowed the system to provide accurate detection of eye closure and yawning, which are key indicators of driver drowsiness.

System Performance in Typical Conditions

In typical driving conditions, such as normal lighting and steady head positioning, the system exhibited high accuracy in detecting fatigue. The use of facial feature tracking, alongside the CNN model, significantly improved the system's robustness, particularly in detecting eye states

Strengths of the CNN Model

The CNN model proved to be especially beneficial in situations where traditional rule-based methods, such as landmark detection by Dlib, might fail:

- Robustness to Landmark Misplacement: Landmark detection can sometimes be imprecise, especially when the face is partially obscured, or when the subject is wearing glasses or hats. In these situations, the CNN was able to compensate for minor errors in landmark detection by focusing on local features such as the shape and movement of the eyes.
- Illumination Variations: The CNN model also showed strength in handling variations in lighting conditions. While traditional methods like Haar cascades or Dlib landmarks might struggle under poor lighting or shadows, the CNN was able to effectively adapt to these changes and still classify the eye state accurately.

Hybrid Approach: Rule-Based + Deep Learning

The testing reinforced the value of using a hybrid approach in the system:

- Rule-Based Methods: The combination of geometrical analysis using the EAR (Eye Aspect Ratio) and MAR (Mouth Aspect Ratio) with deep learning techniques provided a more robust system. Rule-based methods help to handle basic drowsiness signs (e.g., eye closure or yawning) and are computationally inexpensive.
- Deep Learning: The CNN brings the flexibility to handle edge cases that might otherwise lead to false positives or missed detections in more traditional systems. For example, when a subject wears glasses or their eyes are partially covered, the CNN

can still detect the eye state effectively by relying on its ability to learn complex patterns from large datasets.

The CNN model proved to be a valuable addition to the drowsiness detection system, enhancing its accuracy and robustness, particularly in difficult conditions such as poor lighting, partial occlusion, or landmark misalignment. Its integration into the real-time detection pipeline validated its ability to classify eye states accurately and efficiently. This hybrid strategy of combining rule-based detection with deep learning allowed for better performance across a wide range of scenarios, making the system more reliable for real-world deployment in driver safety applications. The results indicate that the CNN model is well-suited for tasks requiring high accuracy and real-time performance, while compensating for challenges faced by traditional methods.

CHAPTER 8
CONCLUSION

8.1 Conclusion

The construction and deployment of a “**drowsiness detection system with deep learning in Python”** have given a real-world and effective method for addressing one of the most significant causes of road accidents—driver fatigue. Capitalizing on the advantages of computer vision and deep learning methods, the system can easily observe a driver's facial activity, especially eye and mouth movement, to identify drowsiness. This solution offers an early warning system that can assist in averting possible accidents due to slow reaction times or temporary loss of consciousness while driving.

The application of Dlib's facial landmark detection enabled precise identification of facial features like the eyes and mouth, which are the primary indicators of a person's alertness. By calculating Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR), the system could detect extended closure of the eyes and yawning—two most prevalent indicators of tiredness. Still, observing that rule-based measures alone can be untrusted in every possible case, the project also employed a Convolutional Neural Network (CNN) for classifying the eye status (open/closed) by using image information. This deep learning module greatly enhanced the system's resilience, particularly under circumstances where geometric computation could break down because of low light conditions, occlusions (e.g., glasses), or camera view angle changes.

The system was deployed with Python, supplemented with important libraries such as OpenCV, Dlib, and TensorFlow/Keras. Video input was handled in real time frame by frame to pull out facial characteristics, run prediction models, and issue alerts on detection of drowsiness. The project showed that integrating real-time processing with intelligent decision-making is not only possible but is also efficient on common hardware without having to use high-performance computing capabilities. The alerting mechanism, both visual and auditory, also serves to ensure that drivers are adequately informed when fatigue is actually detected, hence maximizing the chance of avoidance of accidents.

Finally, this project demonstrates that deep learning-based drowsiness detection is not only technically feasible but also practical for deployment in the real world. It can serve as a baseline model that can be scaled, optimized, and integrated into automotive systems or standalone driver assistance technologies. With further research and development, such systems could be standard in cars, with the potential to enhance road safety, lower accident numbers, and save lives. The successful completion of this project showcases the potential of artificial intelligence and machine learning to deliver intelligent, safety-improving technologies.

The successful development and deployment of a drowsiness detection system using deep learning in Python signifies a major step forward in improving road safety by addressing driver fatigue—one of the leading causes of road accidents worldwide. Through the utilization of computer vision and deep learning techniques, this system can accurately monitor a driver's facial movements, particularly around the eyes and mouth, which are key indicators of fatigue and drowsiness. By detecting subtle signs such as prolonged eye closure (indicative of sleepiness) and excessive yawning, the system is able to trigger early warnings, potentially preventing accidents caused by slowed reaction times or momentary lapses in consciousness while driving.

The core of this system lies in the Dlib facial landmark detection technology, which allows the precise tracking of facial features—primarily the eyes and mouth. By calculating the Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR), the system identifies the signs of drowsiness through the detection of eye closure and yawning. These two factors are among the most common physical indicators of fatigue. However, recognizing that rule-based methods alone may not provide sufficient accuracy in every scenario, especially when faced with challenges like low lighting, partial facial occlusion (e.g., due to glasses), or changes in head angle, the system integrates a Convolutional Neural Network (CNN). The CNN classifies the eye state (open or closed) based on visual data, enhancing the system's ability to handle difficult situations where traditional geometric methods might fail.

CHAPTER 9
APPENDIX

CODING

Main Code

```
# import the necessary packages

from scipy.spatial import distance as dist
from imutils.video import VideoStream
from imutils import face_utils
from threading import Thread
import numpy as np
import playsound
import imutils
import time
import dlib
import cv2
```

Calculate the head pose by tracking facial landmarking and projecting them in 3D space

```
def getheadpose(frame,shape,size):
    #2D image points. If you change the image, you need to change vector
    image_points = np.array([
                            (shape[33, :]),     # Nose tip
                            (shape[8,  :]),     # Chin
                            (shape[36, :]),     # Left eye left corner
                            (shape[45, :]),     # Right eye right corne
                            (shape[48, :]),     # Left Mouth corner
                            (shape[54, :])      # Right mouth corner
                        ], dtype="double")


    # 3D model points.
    model_points = np.array([
                            (0.0, 0.0, 0.0),             # Nose tip
                            (0.0, -330.0, -65.0),        # Chin
                            (-225.0, 170.0, -135.0),     # Left eye left corner
                            (225.0, 170.0, -135.0),      # Right eye right corne
                            (-150.0, -150.0, -125.0),    # Left Mouth corner
                            (150.0, -150.0, -125.0)      # Right mouth corner       ])
```

Convert the3D coordinates to 2D image

```
    # Camera internals
    focal_length = size[1]
    center = (size[1]/2, size[0]/2)
    camera_matrix = np.array(
                        [[focal_length, 0, center[0]],
                        [0, focal_length, center[1]],
                        [0, 0, 1]], dtype = "double"
                        )
    #print ("Camera Matrix :\n {0}".format(camera_matrix))

    dist_coeffs = np.zeros((4,1)) # Assuming no lens distortion
    (success, rotation_vector, translation_vector) = cv2.solvePnP(model_points, image_points,
camera_matrix, dist_coeffs, flags=cv2.SOLVEPNP_ITERATIVE)
```

Project a 3D point (0, 0, 1000.0) onto the image plane.

```
    # We use this to draw a line sticking out of the nose
    (nose_end_point2D, jacobian) = cv2.projectPoints(np.array([(0.0, 0.0, 1000.0)]),
rotation_vector, translation_vector, camera_matrix, dist_coeffs)

    for p in image_points:
        cv2.circle(frame, (int(p[0]), int(p[1])), 3, (0,0,255), -1)

    p1 = ( int(image_points[0][0]), int(image_points[0][1]))
    p2 = ( int(nose_end_point2D[0][0][0]), int(nose_end_point2D[0][0][1]))
    cv2.line(frame, p1, p2, (255,0,0), 2)
    return p1,p2

def notification(value):
    if value:
        print("sending notification")
        url = 'http://blynk-cloud.com/_eTX4XVZPY8CIglkW18kKzaA1kokX47n/notify'
```

```
#paste ur token here and add notification widget in the blynk app
        body = {'body': 'Alert Driver is sleepy'}    #msg u want as notification
        headers = {'content-type': 'application/json'}
        r = requests.post(url, data=json.dumps(body), headers=headers)

def mouth_aspect_ratio(mouth):
    # Compute the euclidean distances between the three sets
    # of vertical mouth landmarks (x, y)-coordinates
    A = np.linalg.norm(mouth[13] - mouth[19])
    B = np.linalg.norm(mouth[14] - mouth[18])
    C = np.linalg.norm(mouth[15] - mouth[17])
    # Compute the euclidean distance between the horizontal
    # mouth landmarks (x, y)-coordinates
    D = np.linalg.norm(mouth[12] - mouth[16])
    # Compute the mouth aspect ratio
    mar = (A + B + C) / (2 * D)
    # Return the mouth aspect ratio
    return mar
def sound_alarm(path):
        # play an alarm sound
        playsound.playsound(path)

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

```
# define two constants, one for the eye aspect ratio to indicate
# blink and then a second constant for the number of consecutive
# frames the eye must be below the threshold for to set off the
# alarm
EYE_AR_THRESH = 0.27
EYE_AR_CONSEC_FRAMES = 48
MOUTH_AR_THRESH = 0.2
MOUTH_AR_CONSECUTIVE_FRAMES = 15
YAWN_COUNT = 0
# initialize the frame counter as well as a boolean used to
# indicate if the alarm is going off
COUNTER = 0
ALARM_ON = False
get_pose = False
MOUTH_COUNTER = 0
YELLOW_COLOR = (0, 255, 255)

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

Grab the indexes of the facial landmarks for the left and right eye

```
(lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
(rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
(mStart, mEnd) = face_utils.FACIAL_LANDMARKS_IDXS["mouth"]
# start the video stream thread
print("[INFO] Starting Video")
vs = VideoStream(0).start()
time.sleep(1.0)

def audioalert():
    global ALARM_ON
    # if the alarm is not on, turn it on
    if not ALARM_ON:
        ALARM_ON = True
        #notification(True)

        # check to see if an alarm file was supplied,
        # and if so, start a thread to have the alarm
        # sound played in the background
        if "alarm.wav" != "":
            t = Thread(target=sound_alarm,
                args=("alarm.wav",))
            t.deamon = True
            t.start()

    # draw an alarm on the frame
    #notification(True)
    cv2.putText(frame, "DROWSINESS ALERT!", (10, 30),
        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
```

Loop over frames from the video stream

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

    # detect faces in the grayscale frame
    rects = detector(gray, 0)
    if(len(rects) < 1):
        cv2.putText(frame, "Alert! Look at Camera", (10,
30),cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
```

```
    else:
        cv2.putText(frame, "", (10, 30),cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255),2)
```

Loop over the face detections

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

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

Average the eye aspect ratio together for both eyes

```
        ear = (leftEAR + righ4tEAR) / 2.0
        # compute the convex hull for the left and right eye, then
        # visualize each of the eyes and mouth
        mouthHull = cv2.convexHull(mouth)
        leftEyeHull = cv2.convexHull(leftEye)
        rightEyeHull = cv2.convexHull(rightEye)
        cv2.drawContours(frame, [mouthHull], -1, YELLOW_COLOR, 1)
        cv2.drawContours(frame, [leftEyeHull], -1, (0, 255, 0), 1)
        cv2.drawContours(frame, [rightEyeHull], -1, (0, 255, 0), 1)

        # check to see if the eye and mouth aspect ratio is below the
        # threshold, and if so, increment the blink frame counter
        if mar > MOUTH_AR_THRESH:
```

```
        MOUTH_COUNTER += 1
        #print(MOUTH_COUNTER)
        if MOUTH_COUNTER >= MOUTH_AR_CONSECUTIVE_FRAMES:
                YAWN_COUNT += 1
                MOUTH_COUNTER = 0
                if YAWN_COUNT > 5:
                    audioalert()
else:
        MOUTH_COUNTER = 0
        notification(False)
if ear < EYE_AR_THRESH:
        COUNTER += 1
        # if the eyes were closed for a sufficient number of frames
        # then sound the alarm
        if COUNTER >= EYE_AR_CONSEC_FRAMES:
            audioalert()
# threshold, so reset the counter and alarm
else:
        COUNTER = 0
        ALARM_ON = False
        notification(False)
if get_pose == True:
    p1,p2 = getheadpose(frame,shape,size)
    #print("pitch" + str(p1[0]) + " yaw" +  str(p2[0]))
    pitch = p1[0]
    look = p2[1]
    if pitch >150 and pitch<200:
        cv2.putText(frame, "looking right".format(ear), (30, 30),
            cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
    elif pitch >270 and pitch <300:
        cv2.putText(frame, "looking left".format(ear), (30, 30),
                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
    if look > 270 and look < 300:
        cv2.putText(frame, "looking down".format(ear), (30, 30),
            cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
    elif look < 100:
        cv2.putText(frame, "looking up".format(ear), (30, 30),
                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
# draw the computed eye aspect ratio on the frame to help
# with debugging and setting the correct eye aspect ratio
# thresholds and frame counters
cv2.putText(frame, "EAR: {:.2f}".format(ear), (300, 30),
        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
cv2.putText(frame, "YAWN COUNT:{}".format(YAWN_COUNT), (270, 50),
        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
cv2.putText(frame, "Press 'h' to Start Head Pose", (20, 280),
        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
cv2.putText(frame, "Press 'q' to Quit", (20, 300),
        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
```

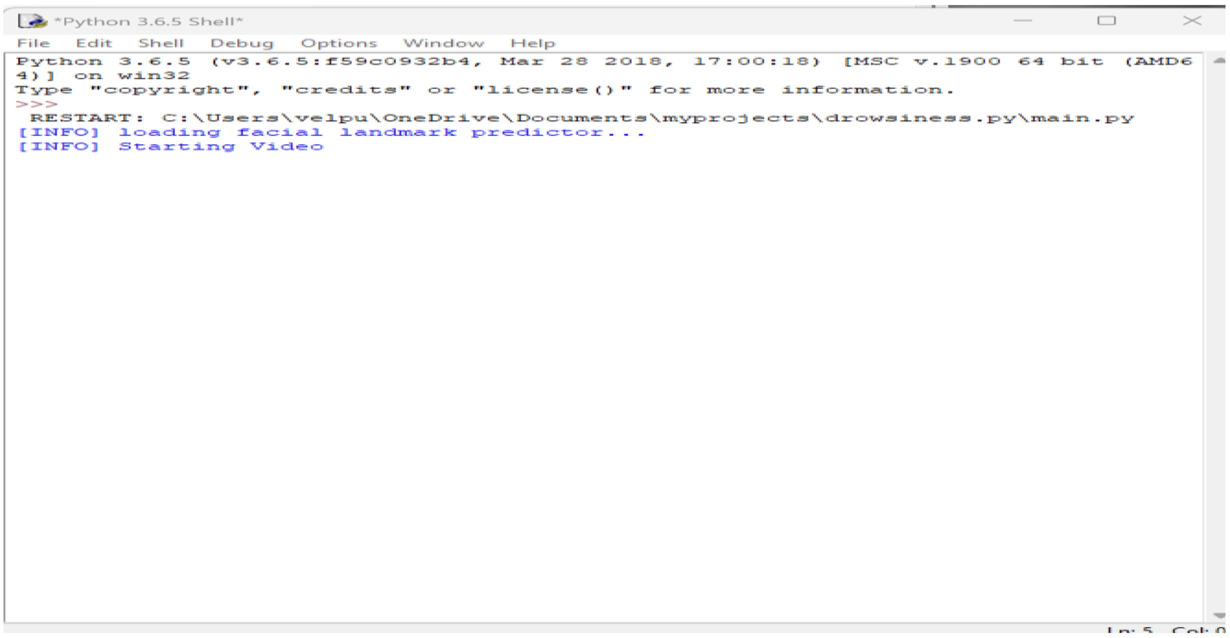
Showing the frame

```
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF

    # if the q key was pressed, break from the loop
    if key == ord("q"):
        break
    if key == ord("h"):
        get_pose = not get_pose

# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()
print("[INFO] Cleaning all")
print("[INFO] Closed")
```

Screenshot



```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64 bit (AMD6
4)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\velpu\OneDrive\Documents\myprojects\drowsiness.py\main.py
[INFO] loading facial landmark predictor...
[INFO] Starting Video
```

Fig 1 Loading Facial Landmarking Prediction

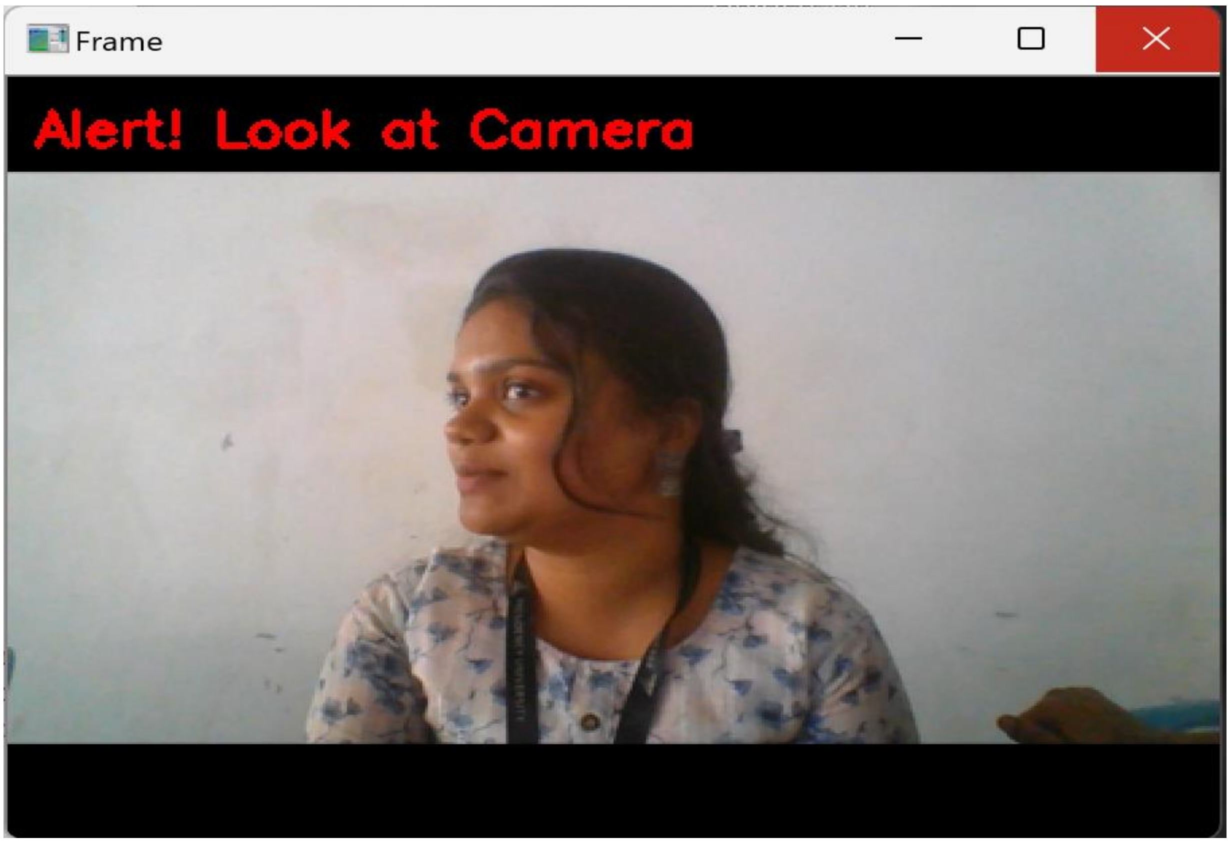


Fig 2 Alert Message

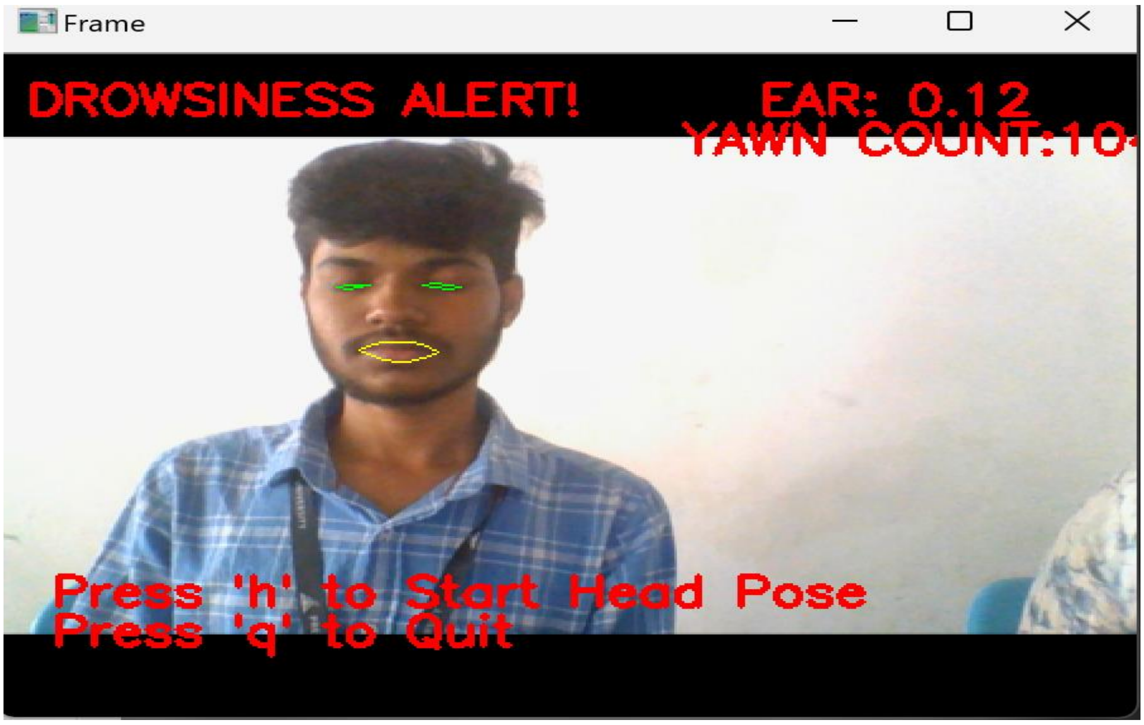


Fig 3 Eye Aspect Ratio

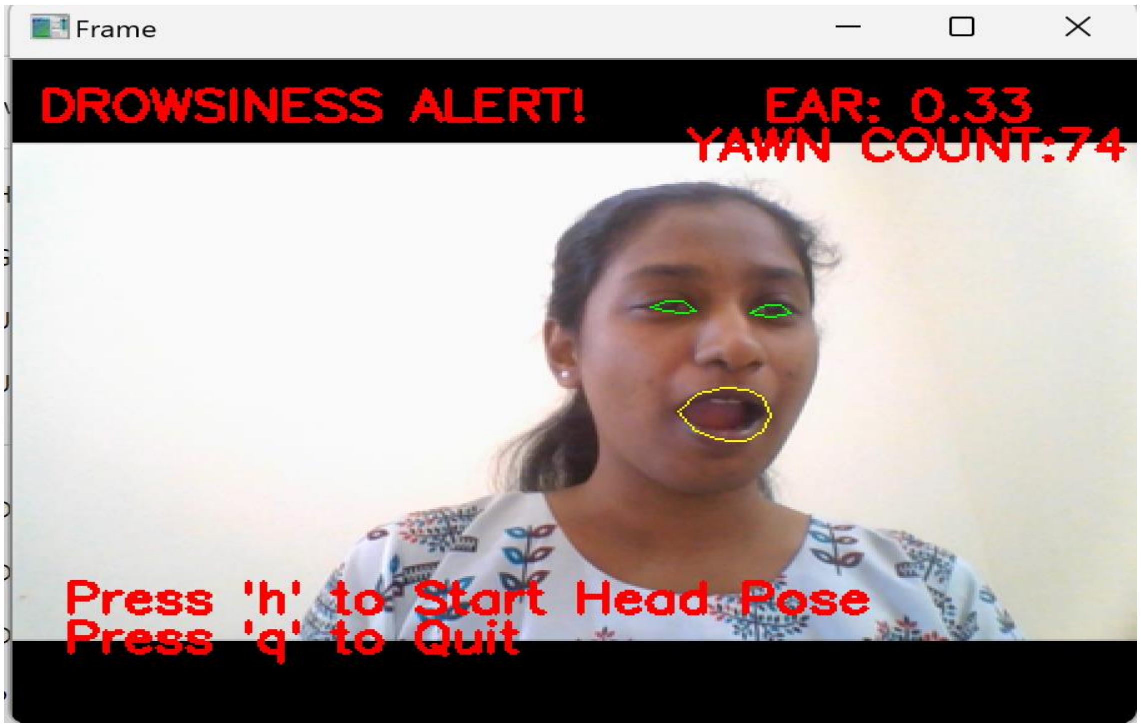


Fig 4 Mouth Aspect Ratio

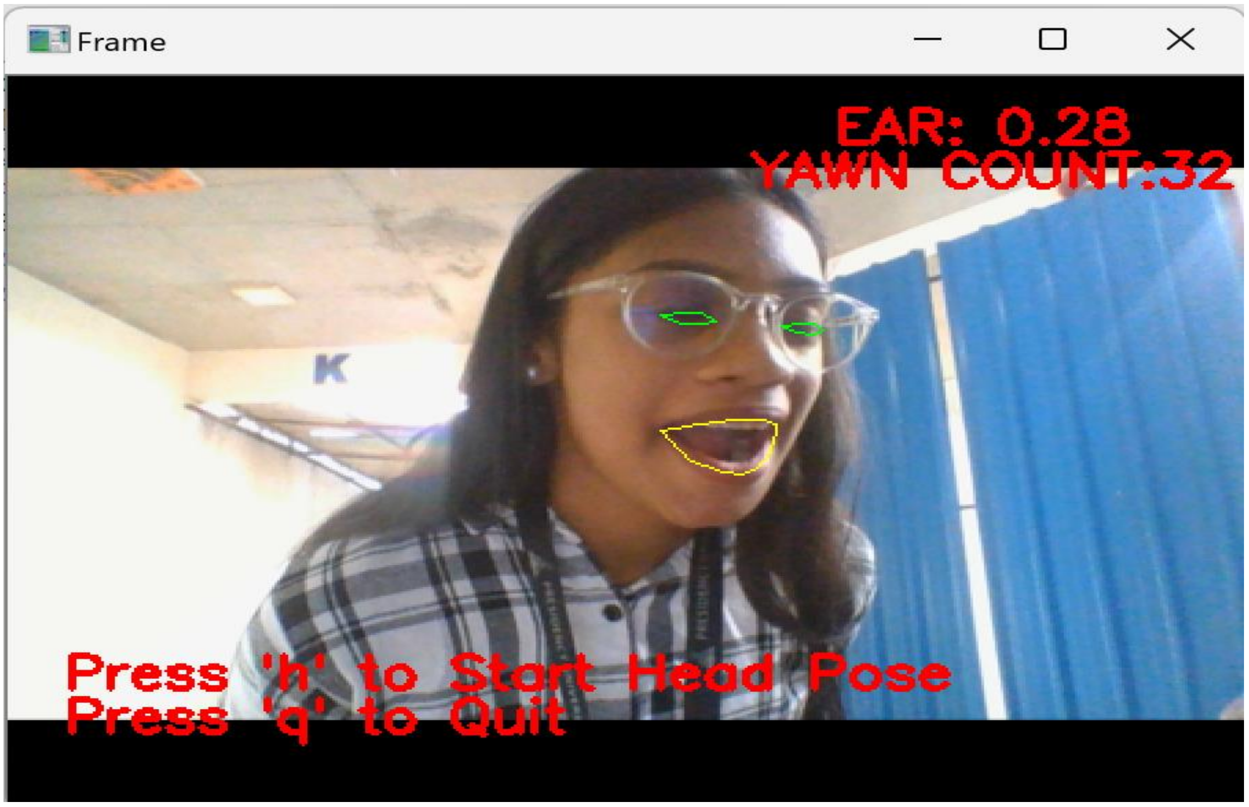


Fig 5 EAR and YAWN

REFERENCES

[1].Dakshnakumar G S, J. Anitha, "Investigation on Driver Drowsiness Detection using Deep Learning Approaches", in *Proceedings of the 2023 IEEE International Conference on Communication and Power Technologies (ICCPCT)*, 2023.
https://doi.org/10.1109/iccpct58313.2023.10245868

[2].Eryl Nanda Pratama, Wikky Fawwaz Al Maki, "Drowsiness Detection System for Masked Face Based on Deep Neural Network and Haar Cascade", in *IEEE Access*, 2022.
https://ieeexplore.ieee.org/document/10029948

[3].Kanwarpartap Singh Gill, Vatsala Anand, Rahul Chauhan, Siddhant Thapliyal, Rupesh Gupta", A Convolutional Neural Network-Based Method for Real-Time Eye State Identification in Driver Drowsiness Detection," in *IEEE Access*, 2023.
https://ieeexplore.ieee.org/document/10442238

[4].Mandapati Ankitha, Sandeep Vemuri, Suraneni Lowkya Gayathri, Jarabala Sindhu Bhargavi, "Enhanced Driver’s Drowsiness Detection System using CNN model", *IEEE Xplore*, 2022.
https://ieeexplore.ieee.org/document/9793253

[5].R Syed Ali Fathima, Kovi Venkata Keerthi, Kovuri Naga Bhuvanesh, Kota Naga Jyothi, Kotikalapudi Sravya, Patnana Vijay Kumar, "Driver Drowsiness Detection System with OpenCV and Keras", *IEEE Xplore*, 2024.
https://ieeexplore.ieee.org/document/10823233

[6].S Spandana, M. Srividhya, Yedavalli Venkata Raghava Rao, P. Devika, V. Srikanth, S Parvathi, "Vision based Driver Drowsiness Detection using Deep Learning", *IEEE*, 2024.
https://doi.org/10.1109/assic60049.2024.10507905