

Drowsiness Detection System using Machine learning in IoT

A Project Report submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING (Specialization in Internet of Things)

Submitted by
PVN SRINIJA (VU21CSEN0600057)
M.SAMHITH (VU21CSEN0600002)
K.SIRI CHANDANA(VU21CSEN0600089)
BENJEMIN CALEB (VU21CSEN0600109)

Under the esteemed guidance of

Dr. Parasana Sankara Rao,Associate Professor
CSE



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GITAM SCHOOL OF TECHNOLOGY
GITAM (Deemed to be University)
VISAKHAPATNAM**

2025

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GITAM SCHOOL OF TECHNOLOGY**

GITAM (Deemed to be University)



DECLARATION

I hereby declare that the project report entitled XXX is an original work done in the Department of Computer Science and Engineering, GITAM School of Technology, GITAM (Deemed to be University) submitted in partial fulfillment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering/ Computer Science and Engineering (AI&ML/DS/CS/IoT). The work has not been submitted to any other college or University for the award of any degree or diploma.

Date: 19 March 2025


Registration No(s)

Name(s)

Signature

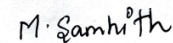
VU21CSEN0600057

PVN SRINIJA



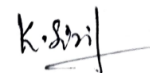
VU21CSEN0600002

M.SAMHITH



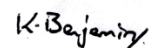
VU21CSEN0600089

K.SIRI CHANDANA



VU21CSEN0600109

BENJEMIN CALEB



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GITAM SCHOOL OF TECHNOLOGY
GITAM (Deemed to be University)**



CERTIFICATE

This is to certify that the project report entitled “**Drowsiness Detection System using Machine learning in IoT**” is a bonafide record of work carried out by PVN Srinija(VU21CSEN0600057), M.Samhith (VU21CSEN0600002), K.Siri Chandana (VU21CSEN0600089), Benjemin Caleb (VU21CSEN0600109) students submitted in partial fulfilment of the requirement for the award of the degree of Bachelors of Technology in Computer Science and Engineering / Computer Science and Engineering (**IoT**).

Date : 19 March 2025

Project Guide

Dr.Parasana Sankara Rao,

Associate Professor,CSE

Head of the Department

Dr. Kuppli Naveen Kumar,

Head of the Department,

AI & DS

ACKNOWLEDGEMENT

We are writing this to express our sincere gratitude to all the people who are making it possible to work and develop this project, helping shape our ideas into a well-structured project with their constant guidance and encouragement for us.

We would like to express our sincere thanks to **Dr. Naveen Kumar Kuppili**, Associate Professor and Head of the Department of AI & DS, GITAM School of Technology, for his support and for encouraging us to pursue this capstone project.

We would like to thank our project guide **Dr. Parasana Sankar Rao**, Associate professor in CSE, GITAM School of Technology, who has been giving continuous critical suggestions and extension of a proper working atmosphere. His guidance and expertise are very invaluable for our success.

It is indeed with a great sense of pleasure and immense sense of guidance that we acknowledge the help and we are highly indebted to **Prof. Nagendra Prasad**, Director and **Prof. S Arun Kumar**, Dean-CSE, GITAM School Of Technology, for their support during the tenure of the academic year.

We are also thankful to all the staff members of the Computer Science Engineering Department for their valuable suggestions. We would like to thank our teammates and parents who constantly extend their help, encouragement and moral support either directly or indirectly throughout this project.

TABLE OF CONTENTS

S.No.	Description	Page No.
1.	Abstract	1
2.	Introduction	2
3.	Literature Review	4
4.	Problem Identification & Objectives	7
5.	Existing System, Proposed System	10
6.	Proposed System Architecture / Methodology	12
7.	Technologies Used	16
8.	Implementation (Sample Code and Test Cases)	18
9.	Results & Discussion	22
10.	Conclusion & Future Scope	23
11.	References	24
12.	Annexure 1 (Source Code)	26
13.	Annexure 2 (Output Screens)	30
14.	Annexure 3 (Publication if published) *	36

*** Not mandatory**

Discussion Notes (not to be included in the report) 7

ABSTRACT:

Drowsy driving is a leading cause of road accidents worldwide, resulting in thousands of fatalities and injuries each year. Conventional drowsiness detection methods, such as physiological monitoring and vehicle-based tracking, work accurately but are often intrusive, expensive, and unsuitable for real-time deployment. To address these limitations, this research proposes a real-time, non-intrusive driver drowsiness detection and prevention system that leverages deep learning, computer vision, and IoT-based alert mechanisms to enhance road safety.

The system continuously monitors the driver's facial expressions and behavioral cues, such as eye closure, yawning frequency, and head movements, using a live camera feed. Using Machine learning and deep learning models, when drowsiness is detected, an alert mechanism is activated, consisting of voice alert to provide immediate intervention. Additionally, cloud-based services enable real-time notifications via SMS alerts, ensuring timely responses from emergency contacts or fleet management systems.

The proposed system was evaluated for detection accuracy, response time, and robustness under varying lighting conditions and driver postures. Experimental results demonstrate high detection efficiency, low latency, and reliable alerting, making it a practical solution for real-world applications. By integrating artificial intelligence with embedded computing and IoT, this research presents a cost-effective, scalable approach to reducing drowsiness-induced road accidents, contributing to the development of intelligent transportation safety systems.

INTRODUCTION:

Drowsy driving is a major cause of road accidents worldwide, leading to severe injuries, fatalities, and substantial economic losses. Fatigue impairs a driver's reaction time, decision-making ability, and situational awareness, making it as dangerous as driving under the influence of alcohol. According to the World Health Organization (WHO), driver-related factors like drowsiness and distractions are among the primary causes of road crashes [1]. The National Sleep Foundation estimates that drowsy driving accounts for approximately 100,000 police-reported crashes annually in the U.S., resulting in 6,400 deaths and 50,000 injuries [2]. Similarly, the National Highway Traffic Safety Administration (NHTSA) reported that in 2022 alone, 633 fatalities were linked to drowsy driving, accounting for 1.2% of all traffic-related deaths [3]. These alarming statistics underscore the critical need for real-time driver monitoring systems to prevent fatigue-induced accidents and enhance road safety.

Driver fatigue, microsleep, and drowsiness are significant contributors to these incidents, making it imperative to develop effective detection and prevention mechanisms. Traditional drowsiness detection methods, such as EEG and ECG-based monitoring, are highly accurate but impractical for real-world applications due to their intrusive nature and reliance on specialized equipment [4]. Other behavior-based approaches, such as monitoring steering patterns and vehicle movement, often suffer from high variability among drivers and delayed response times, limiting their effectiveness in critical situations [5].

With advancements in deep learning and computer vision, facial feature analysis has emerged as a promising solution for real-time drowsiness detection. By analyzing eye closure, yawning, and head movements, machine learning models can accurately classify a driver's state and trigger appropriate countermeasures. Furthermore, IoT-based alerting systems can enable real-time intervention by notifying emergency contacts or activating corrective mechanisms to restore driver alertness.

To address these challenges, this research proposes a real-time driver drowsiness detection and prevention system that integrates deep learning-based facial analysis, IoT-enabled alerts, and automated intervention mechanisms to mitigate drowsy driving risks.

LITERATURE REVIEW:

NAME OF THE PAPER	NAME OF THE AUTHOR	YEAR OF PUBLICATION	AIM	OUTPUT	LIMITATIONS
Real-time Driver Drowsiness Detection and Assistance System using Machine Learning and IoT(IEEE)	Pragathi Sudarshan, Vivek Bhardwaj, Virender	2023	To develop a real-time driver drowsiness detection system using machine learning and IoT to monitor eye closure and use immediate alert system on detection to prevent road accidents.	The system detects if a driver is drowsy by monitoring eye closure and triggers physical alerts like spraying water and a buzzer, real time SMS to concerned individuals from IoT platforms.	The system might find it challenging to detect drowsiness accurately when driver is wearing glasses or in low-light conditions.
IoT-Enabled Driver Drowsiness Detection Using Machine Learning(IEEE)	Mrinmoy Guria, Biswajit Bhowmik	2022	To design and implement an IoT-based system embedded with ML techniques to detect driver drowsiness effectively and issue alerts in order to prevent road accidents.	It uses EAR to detect driver's sleepiness by estimating EAR, Cartesian coordinates with a success rate of 85%. It aims to alert the driver to minimize road accidents with timely alerts.	The system is sensitive to the eye aspect ratio: if it is above or below the standard, there will be constant trips. Its accuracy is affected by road and lighting conditions. It misses other indicators such as yawning, and delays in low-light conditions.

Driver Drowsiness Detection System - An Approach By Machine Learning Application(ar Xiv)	Jagbeer Singh, Ritika Kanojia, Rishika Singh, Rishita Bansal	2022	To develop a system that will detect driver drowsiness using facial____recognition and eye tracking.	The system detects driver drowsiness using a webcam by monitoring eye blinking patterns. If the eyes remain closed beyond a set threshold, it triggers an alarm to alert the driver.	The system's accuracy decreases in poor lighting conditions. It cannot accurately detect drowsiness when the driver is wearing glasses. The system doesn't track other indicators for drowsiness and the alarm's sensitivity can vary based on an individual's eye aspect ratios.
Application of IoT & Machine Learning for Real-time Driver Monitoring and Assisting Device (IEEE)	Pranay Sharma, Naveksha Sood	2020	To propose a driver monitoring and assisting device using IoT sensors and machine learning for detecting sobriety and drowsiness	The system monitors driver alertness and prevents accidents by detecting drowsiness and generating timely alerts using trained machine learning models.	The drowsiness detection is mainly based on alcohol-only detection. It uses cheaper sensors and uses invasive methods. Cheaper sensors may result in false detection of drowsiness or system may fail to work.

PROBLEM IDENTIFICATION AND OBJECTIVES:

PROBLEM IDENTIFICATION:

- **High Accident Rates Due to Driver Fatigue:** A large number of accidents on roads are attributed to driver fatigue, drowsiness, and microsleep, especially when driving at night. The drivers may ignore their drowsiness, or would be unable to exactly determine they are feeling drowsy and continue driving, which might lead to accidents and loss of life.

PROBLEMS WITH THE EXISTING DETECTION SYSTEMS:

- **Limited Accessibility:** The drowsiness detection mechanisms are integrated into high-end vehicles, such as Mercedes-Benz, Landrover, and Tesla. These vehicles are expensive and might not be accessible to average drivers.
- **Limited Alert mechanisms:** While the existing systems might provide visual alerts or simple vibrations upon fatigue detection, drivers might fail to notice these alerts if they are too sleepy. Thus, stronger physical intervention and alert systems are necessary.
- **Wearables:** While wearables are a great way for recognising early signs of drowsiness, the driver might not feel comfortable wearing the gadgets all the time, or might forget to wear them sometimes. This might lead to an increase in chances of accidents.
- **Reliance on Indirect Indicators:** Many systems depend on indirect indicators of drowsiness, such as steering behavior. These may not always provide accurate detection and cannot detect early signs of drowsiness.

OBJECTIVES:

The main objectives of this project include:

- Use non-invasive methods to track the indicators for drowsiness detection, and make the system affordable and easy to use for everyone.
- Design a system that continuously tracks the driver's eye for signs of drowsiness using a combination of machine learning algorithms like YOLO v8 and CNN.
- Create engaging alert mechanisms based on IoT, which would physically wake the driver up and even make an alarm so that corrective action would be taken instantaneously.
- Create a system for real time notification alert to the emergency contacts and emergency services such as ambulance or fire department if needed.

EXISTING SYSTEM:

- **EEG and ECG TECHNOLOGY:**

Some systems use EEG and ECG sensors to monitor brain and heart activity, detecting driver drowsiness and triggering alerts. Others track respiration rate, though this method may not always provide reliable detection.

Drawbacks: The sensors may be overly sensitive to drowsiness, affected by environmental factors, and invasive for real-time use. Their high cost and discomfort limit reliability and practicality outside controlled settings.

- **FORD DRIVER ALERT**

Ford vehicles feature advanced lane-keeping monitoring systems that analyze steering input to detect signs of drowsiness and fatigue. When abnormal patterns are identified, the system issues warnings and prompts the driver to take a break, enhancing road safety.

Drawbacks: The system may miss drowsiness if the driver maintains control but is still tired. Frequent lane changers might never trigger an alarm, leading to overreliance and reduced self-assessment of alertness.

- **MERCEDES ATTENTION ASSIST:**

The company has implemented an Attention Assist system that detects drowsiness signs by monitoring various parameters including steering patterns and driver behavior. The very instant the system determines drowsiness, it flashes a coffee cup symbol and instructs the driver to take a break.

Drawbacks: The driver may miss alerts, and reliance on steering patterns can cause false positives. External factors like wind and weather may confuse the system, leading to incorrect or missed detections.

- **LAND ROVER'S DRIVER CONDITION MONITOR:**

This integrated safety feature detects driver fatigue and enhances road safety by monitoring behavior and vehicle dynamics. It categorizes alertness levels as fatigued or highly fatigued, displaying a dashboard icon to prompt the driver to take a break.

Drawbacks: The Driver Condition Monitor works only between 60-180 kmph and can be manually turned on or off. If the driver forgets to activate it or drives outside this range, drowsiness may go undetected, increasing accident risk.

PROPOSED SYSTEM:

To overcome the drawbacks observed in the existing systems, we propose a system that uses a camera module to detect the drowsiness in drivers through video feed input and apply Machine Learning and deep learning algorithms to identify drowsiness systems and alert the driver and concerned person accordingly.

The features of the system we would develop will be as follows:

INTEGRATED MACHINE LEARNING MODELS :

- This system mainly uses the YOLO v8 model , OpenCV library, Dlib and a Convolutional Neural Network (CNN) for the purpose of identification of drowsiness indicators in a driver.
- The YOLO v8 algorithm is used to find the location of the eyes of the driver, and then check if the eyes are open or closed. Since the eyes of the driver close for a threshold value greater than the defined limit, it will detect the driver to be drowsy.
- Real-time video analysis in the CNN classifies the condition of the driver to better detect drowsiness by pattern recognition.

REAL TIME ALERT MECHANISM THROUGH CLOUD PLATFORM:

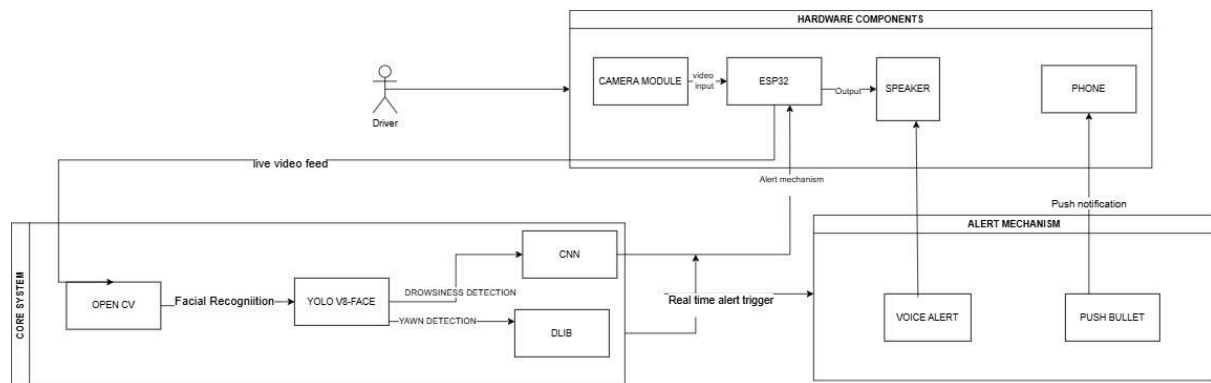
- **Voice Alert:** Voice alert is placed near driver's seat and triggered when the drowsiness is detected. After the threshold of 5 seconds is crossed the voice alert is activated by the model.

- **Push Notification:** when drowsiness is detected the message notification is sent to the designated phone number through pushbullet application.
- Once the drowsiness is detected by the model, a real time alert mechanism is triggered through the code and sent through PushBullet to the registered mobile.

COMPACT AND NON INTRUSIVE DESIGN:

- The system is designed with easy installation on vehicles not to cause any discomfort or distraction from the eyes of the driver. It uses a small camera module for processing in conjunction with a ESP 32 Camera Module.
- The camera module is mounted in front of the driver's seat, so that it has an unobstructed view of the driver in his sitting position.
- It continuously monitors the driver's face and eye movements, positioning all components to prevent distraction while driving.

PROPOSED SYSTEM ARCHITECTURE:



The system architecture of the Driver Drowsiness Detection System is divided into three main components: Hardware Components, Core System, and Alert Mechanism .

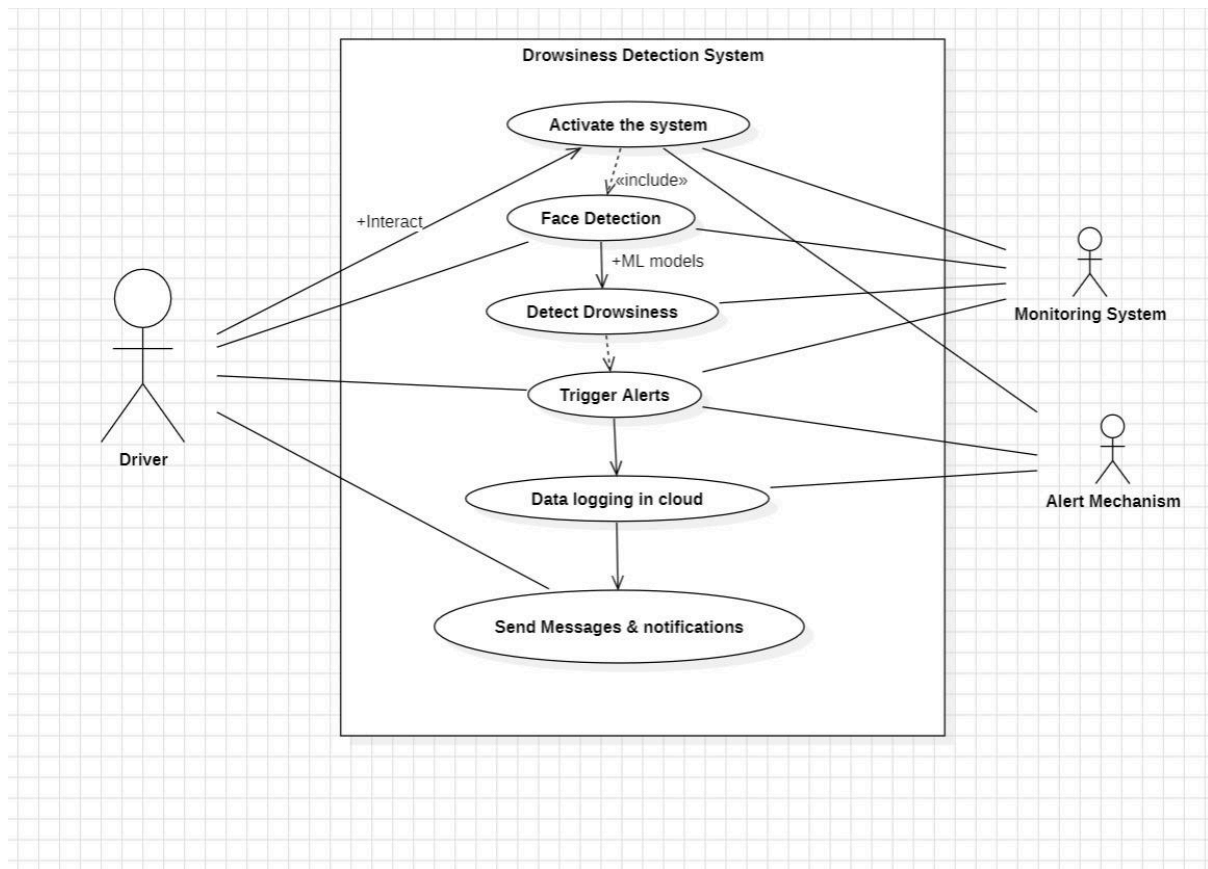
The hardware components consist of IoT equipment and alerting components: Camera module, ESP 32 Camera Module, for getting video input and analysis. Speaker and phone are ensured to be placed near the driver's seat for alerting.

The core system consists of The Machine learning modules: OpenCV, YOLO v8 model, dlib and CNN for facial recognition, eye feature , mouth monitoring and drowsiness detection

respectively. The Alert mechanism, consisting of voice alert and phone notifications, is used for triggering alert systems automatically when drowsiness conditions are met.

We used a use case diagram, class diagram, sequence diagram, activity diagram and component diagram to further elaborate the system architecture.

USE CASE DIAGRAM:



ACTORS:

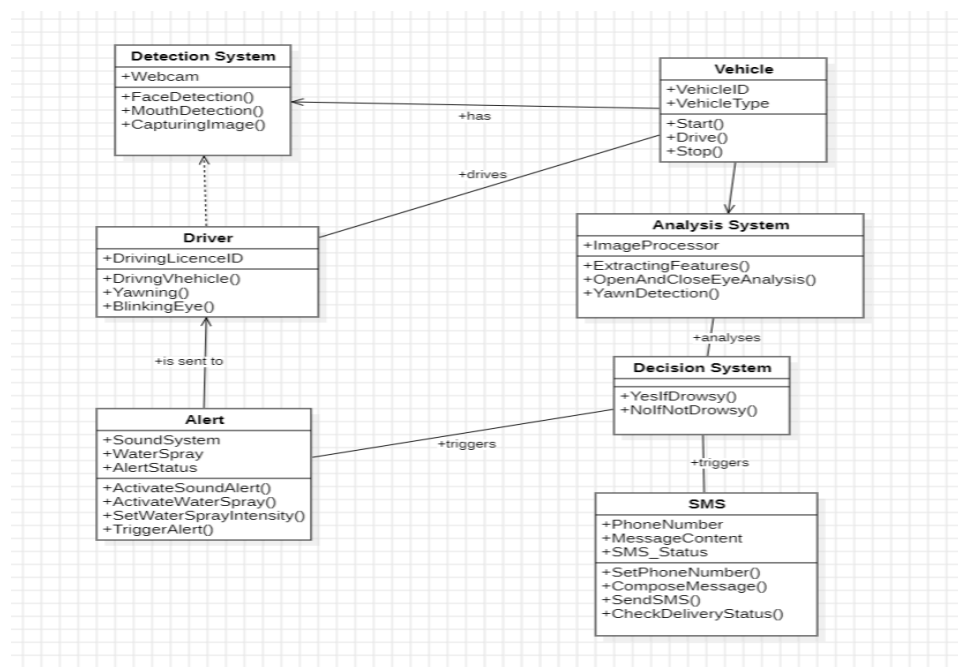
1. **Driver:** Interacts directly with the drowsiness detection system to monitor their alertness and activates the system.
2. **Monitoring System:** Collects and interprets data on the driver's drowsiness status for continuous analysis.
3. **Alert Mechanism:** An external mechanism that triggers alarms or notifications to alert the driver when drowsiness is detected.

USE CASES:

1. **Activate the System:** The driver turns on the car engine, which turns on the drowsiness detection system, enabling it to begin monitoring.
2. **Face Detection:** The system uses machine learning algorithms to detect the driver's face and eyes for accurate monitoring.

3. **Detect Drowsiness:** This is the core functionality of the system. Based on the driver's eye closure rate and other features, the system identifies signs of drowsiness.
4. **Trigger Alerts:** If drowsiness is detected, the system activates the alert mechanism, which includes an audible alarm, SMS, and water spray.
5. **Data Logging in Cloud:** The system logs drowsiness events and system status data to the cloud, allowing for data storage, analysis, and access to historical information.
6. **Send Messages & Notifications:** Apart from immediate alerts, the system also sends messages to a designated contact or monitoring center to ensure that assistance is available if needed.

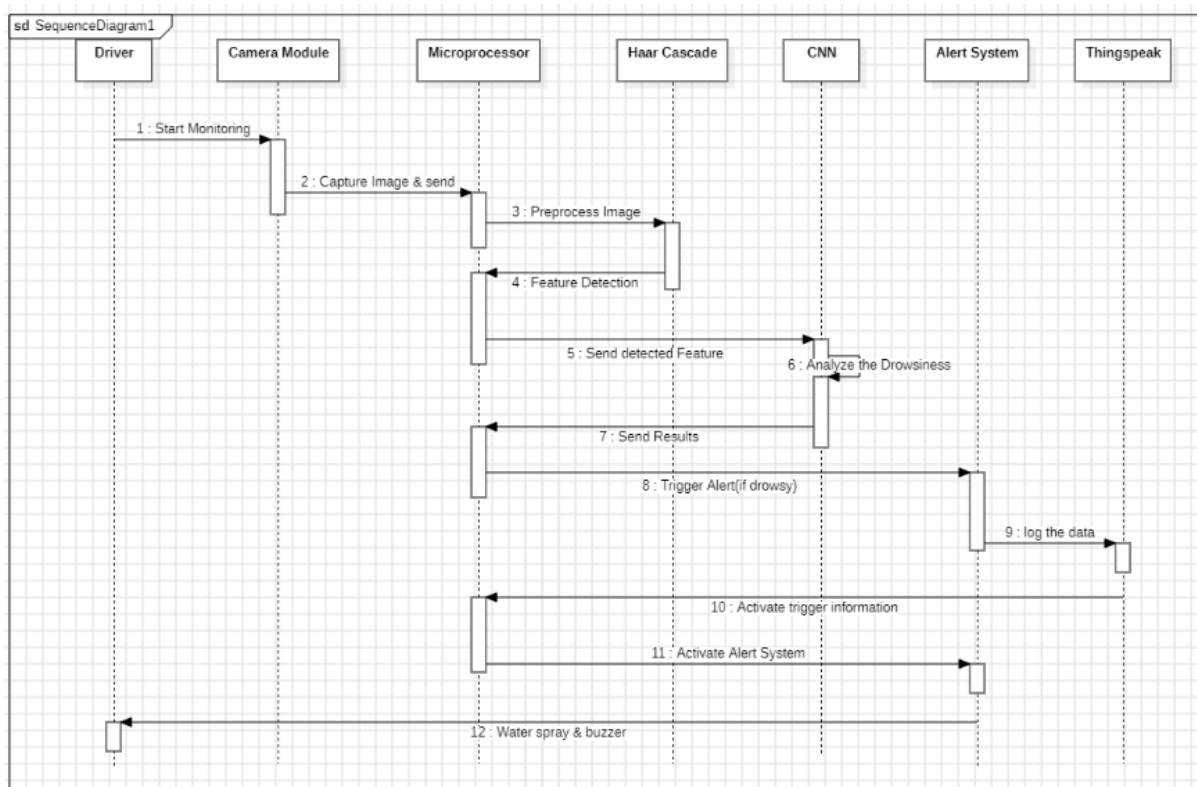
CLASS DIAGRAM:



1. **Detection System** - It detects drivers' face and mouth movements by analyzing images from a webcam. It and analyzes them to look for signs of drowsiness.
2. **Vehicle** - It stores data concerning the vehicle, including ID and type. It also has functions that start, drive, and stop the vehicle depending on its needs.
3. **Driver** - Involves driver-dependent data, comprising license ID. It carries behaviors like the vehicle and yawning or eye blinking for detecting sleep.

4. **Analysis System** - In this, the captured images are analyzed to extract the features involved in the analysis of eyes and yawn. The drowsiness indication from this analysis decides the sign the driver is presenting.
5. **Decision System**- Determines whether a driver is drowsy or alert based on the results of analysis and takes relevant actions accordingly.
6. **Alert**- Activates sound for waking a driver. Intensity could be set as well to trigger an alert once it detects drowsiness in the driver.
7. **SMS** Manages the functionality of SMS, among which are setting phone numbers and message content. It sends alerts through SMS and checks the delivery status.

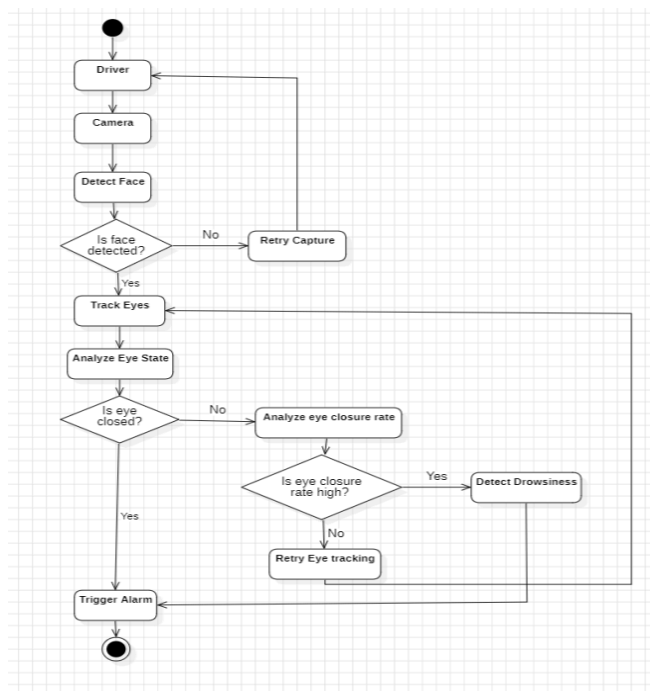
SEQUENCE DIAGRAM:



1. **Driver** initiates the system and the camera module.
2. **Camera Module** captures a video feed of the driver continuously and sends it to the microcontroller for further processing.

3. **Microcontroller** pre-processes the captured video feed to prepare for feature detection from video frames.
4. **YOLO v8** identifies features within the video frame. It seeks facial features of the driver.
5. Detected features are passed on to the CNN for further evaluation.
6. **CNN** evaluates the drowsiness of the driver based on the detected features and sends results back to the microprocessor
7. **The alert system** now activates a trigger to alert the driver after receiving output from microprocessor.
8. **Physical interventions** in the alert system complete its process (Water Spray & Buzzer).

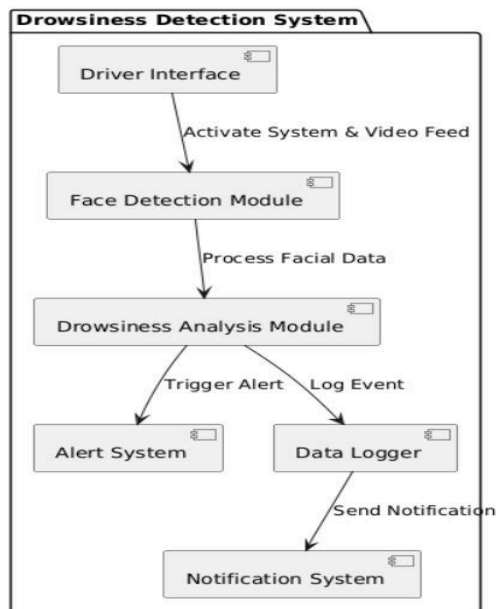
ACTIVITY DIAGRAM :



1. **Monitoring Process** – The system starts tracking driver alertness, initiating the detection workflow.
2. **Video Capture** – The camera records the driver's face for processing.
3. **Face Detection** – The system detects a face; if none is found, it retries until successful.
4. **Eye Tracking** – Identifies eye movement, determining if eyes are open or closed.

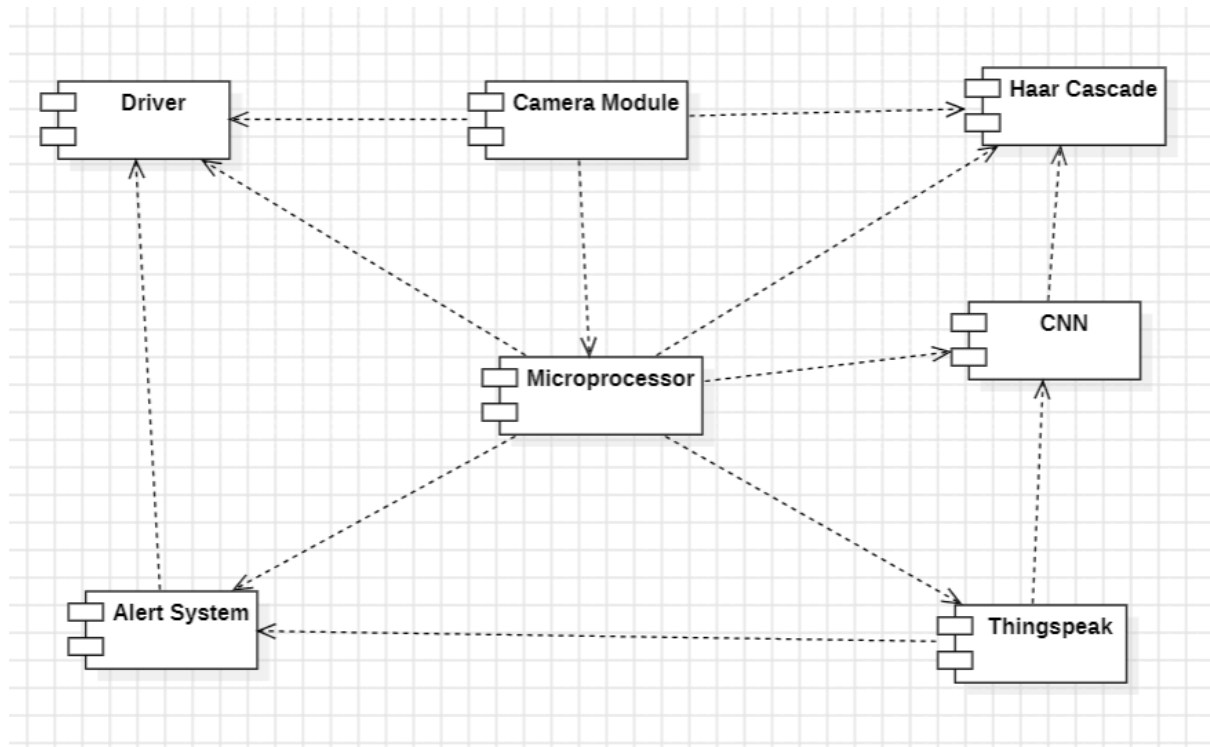
5. **Drowsiness Detection** – If eye closure rate is high, the system detects drowsiness.
6. **Alarm Trigger** – An alarm activates if drowsiness is detected to ensure driver safety.

COMPONENT DIAGRAM:



- 1.**Driver Interface** This interface enables the driver to power the system and initiate the video feed. It will start monitoring for drowsiness.
- 2.**Module Face Detection** This module analyzes the video feed to identify and separate the face of the driver. Accurate extraction of facial data will then occur.
- 3.**Module Drowsiness Analysis** This module determines facial data that might indicate drowsiness. Once it detects drowsiness, it will send an alert for the driver.
- 4.**Alert System** Triggers an alarm when drowsiness is detected. The alarm would wake or alert the driver immediately.
- 5.**Data Logger** Every instance of drowsiness detection is logged for later reference. The logged data can be used for analysis and reporting.
- 6.**Notification System** Notifications are sent to other systems or contacts when drowsiness is detected. This adds an extra layer of safety as it alerts others.

COLLABORATION DIAGRAM:



- **Driver** – The primary user; the system detects their face and eyes for drowsiness.
- **Camera Module** – Captures video frames and sends them to the microprocessor for analysis.
- **Microprocessor** – Processes video using YOLO v8 for face/eye detection and a CNN model for drowsiness analysis; sends alerts to ThingSpeak.
- **YOLO v8** – Identifies the driver’s face and eyes in real-time, detecting drowsiness indicators.
- **Alert System** –
 - **Push Notifications** – Sends real-time alerts via PushBullet to connected devices.
 - **Voice Alert** – Triggers after 5 seconds of detected drowsiness.

Methodology:

The system uses an ESP32 camera module with deep learning for real-time drowsiness detection, following four stages: data gathering, model implementation, execution, and alert activation.

1. Data Collection & Preprocessing

For training an effective drowsiness detection model, a labeled dataset of drowsy and non-drowsy faces was obtained from Kaggle. The dataset comprised images of faces with varying states of drowsiness, i.e., closed eyes and yawning. To make the model more generalizable, the following data preprocessing methods were used:

- **Image Augmentation:** Random rotation, flipping, changing brightness, and contrast normalization were done to replicate real-world differences.
- **Resizing & Normalization:** All images were resized to a constant resolution appropriate for the CNN model and normalized for effective training.
- **Facial Landmark Extraction:** Important facial features were marked to help detect drowsiness markers like eyelid closure and mouth opening.

2. Model Implementation

2.1 YOLO-Based Face Detection

The You Only Look Once (YOLO) model was used for real-time facial detection because of its accuracy and speed. The ESP32 camera streams video frames constantly, and these are fed through the YOLO model for real-time face detection.

2.2 Pretrained CNN for Drowsiness Classification

A Convolutional Neural Network (CNN), which was pretrained on drowsiness datasets, was incorporated into the YOLO pipeline to predict whether the detected face indicates drowsiness. The CNN model was trained on:

- Convolutional layers to learn facial features like eye openness.
- Fully connected layers to classify the frame as "Drowsy" or "Non-Drowsy."

- Softmax activation function for the final classification output.

2.3 Mouth Aspect Ratio (MAR) and Eye Aspect Ratio(EAR) Using Dlib

For improved detection accuracy, Dlib's facial landmark detector was employed to calculate the Mouth Aspect Ratio (MAR) and Eye Aspect Ratio (EAR). MAR is determined by computing key points surrounding the mouth, which aids in detecting yawning, a strong sign of drowsiness.

EAR is determined by computing key points surrounding the eyes, which helps detect eye closure, a crucial indicator of drowsiness.

- MAR is determined by the following formula:

$$\text{MAR (Mouth Aspect Ratio)} = \frac{\|P2-P8\| + \|P3-P7\| + \|P4-P6\|}{3 \times \|P1-P5\|}$$

where P1,P2,,P8P1, P2,, P8 are the coordinates of the mouth landmarks. A high MAR value across many frames indicates yawning and predicts drowsiness.

- EAR is determined by the following formula

$$\text{MAR (Mouth Aspect Ratio)} = \frac{\|P2-P6\| + \|P3-P5\|}{2 \times \|P1-P4\|}$$

where P1,P2,P3,P4,P5,P6 are the coordinates of the eye landmarks.

A low EAR value across multiple frames indicates eye closure and predicts drowsiness

3. Real-Time Processing & Alert Activation

3.1 Frame Processing Pipeline

The ESP32 camera continuously takes live video frames and sends it for processing to the VS code, where images are captured to gain frames.

The YOLO model identifies the face in a frame. The identified face is cropped and fed to the CNN model for classification of drowsiness. At the same time, Dlib's facial landmark detector also computes the Mouth Aspect Ratio (MAR) and Eye Aspect Ratip(EAR).

Based on both CNN predictions and MAR, a decision is made whether the driver is drowsy or not.

3.2 Alert Mechanism

- The system triggers an alert sequence immediately when drowsiness is detected to alert the driver and external contacts. The alert mechanism includes:
 - Voice Alert Activation: ESP32 activates a voice alert to give an audible alert.
 - Push Notifications: The system additionally provides an alert through PushBullet to registered mobile devices ,providing real-time alerts.

4. System Testing & Validation

Several test scenarios were performed to gauge the effectiveness of the system:

- Accuracy Testing: The CNN model was tested using a different validation dataset to prove its performance under real-world circumstances.
- False Positive & Negative Analysis: The robustness of the system was evaluated to prevent unnecessary alerts and provide high sensitivity to drowsiness and metrics such as F1 score, and various performance metrics.

TOOLS/TECHNOLOGIES USED:

HARDWARE COMPONENTS:

- **ESP-32 Camera Module:**

The ESP-32S camera module captures real-time video of the driver's face and eyes. It processes the image locally or sends it to a connected system for drowsiness detection.

- **Power Supply:**

A stable power supply is used to operate the ESP-32S and other components. For our project, PC is used for power supply.

SOFTWARE COMPONENTS:

Software Components:

- **OpenCV:** An open-source library for real-time video and image processing, used for drowsiness detection.
- **CNN:** A deep learning algorithm that analyzes facial patterns to detect drowsiness.
- **Dlib:** A machine-learning library for real-time face detection, landmark recognition, and object tracking.
- **Python:** The primary programming language for implementing video processing and drowsiness detection algorithms.
- **Dataset:** A labelled dataset for CNN training is obtained from Kaggle platform. The dataset originally contained two directories: Drowsy and Non-drowsy consisting of nearly 41000 labelled images, optimal for supervised learning.
- **VS Code :** A widely used code editor for writing, debugging, and managing the project's software components.
- **Arduino IDE** A development platform used for programming microcontrollers and integrating hardware components.
- **Kaggle :** A platform for accessing datasets and training machine learning models, including CNN for drowsiness detection.
- **Pushbullet** – Facilitates real-time communication and alerts by sending notifications to connected devices.

IMPLEMENTATION:

CODING:

The coding is done in two platforms: Arduino IDE and VS code.

ESP32- CAM module and Wi-Fi Configuration:

The ESP32-CAM is configured to capture live images and send them to a server for further processing. The ESP32-CAM is connected to a Wi-Fi network and streams video. The camera connects to a predefined Wi-Fi network using Wi-Fi module.

- PSRAM usage, Frame size, and pixel format, are defined in this code snippet, and main program execution is written in setup() function to initialise camera and Wi-Fi.
- This code when run while connecting the camera module to port, a url is produced for the camera, which is the ip address of the camera. It can be then used anywhere to access the camera.


```

C/C++
const char *ssid = "Siri's A55";
const char *password = "Srinija@2003";

void startCameraServer();
void setup() {
  Serial.begin(115200);
  Serial.println();

  // WiFi Connection
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nConnected to WiFi!");

  // Camera Initialization
  camera_config_t config;
  config.xclk_freq_hz = 20000000;
  config.frame_size = FRAMESIZE_QVGA;
  config.pixel_format = PIXFORMAT_JPEG;
  config.grab_mode = CAMERA_GRAB_WHEN_EMPTY;
  config.fb_location = CAMERA_FB_IN_PSRAM;
  }

  startCameraServer();

  Serial.print("Camera Ready! Use 'http://");
  Serial.print(WiFi.localIP());
  Serial.println("' to connect");

}

void loop() {
}

```

Data Splitting:

The dataset is divided into three folders: Test, Validation and Training, and each of them is separately divided into two sub folders: Drowsy and Non- Drowsy, with the help of python script.

The dataset is divided in the ratio, 7:2:1 , as training, validation and testing respectively.

Creating the three directories:

```
Python
for folder in [train_dir, val_dir, test_dir]:
    for subfolder in ['Drowsy', 'Non_Drowsy']:
        os.makedirs(os.path.join(folder, subfolder), exist_ok=True)
```

Defining split ratio:

```
Python
train_ratio = 0.7
val_ratio = 0.2
test_ratio = 0.1
```

Shuffle and Split the Dataset

This function is called separately for drowsy and non_drowsy categories. This

- Shuffles the dataset to ensure randomness.
- Splits the images into train, validation, and test sets.
- Copies the files into the respective directories.

```
Python
def split_and_copy(source_dir, train_dir, val_dir, test_dir):
    files = os.listdir(source_dir)
    random.shuffle(files)

    train_count = int(len(files) * train_ratio)
    val_count = int(len(files) * val_ratio)

    train_files = files[:train_count]
    val_files = files[train_count:train_count + val_count]
    test_files = files[train_count + val_count:]

    for file in train_files:
        shutil.copy(os.path.join(source_dir, file), train_dir)
    for file in val_files:
        shutil.copy(os.path.join(source_dir, file), val_dir)
    for file in test_files:
        shutil.copy(os.path.join(source_dir, file), test_dir)
```

CNN Model Training:

The CNN model classifies driver drowsiness using a preprocessed dataset with data augmentation. It has three convolutional layers (32, 64, 128 neurons) with ReLU and max-pooling. Compiled with the Adam optimizer and binary cross-entropy, it trains for 45 epochs with real-time validation. Performance is assessed via test accuracy and a confusion matrix.

Data preprocessing:

```
Python
train_datagen = keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(128, 128),
    batch_size=32,
    class_mode='binary'
)
```

CNN model Definition:

```
Python
model = keras.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)),
    layers.MaxPooling2D(pool_size=(2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),

    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),

    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(1, activation='sigmoid') # Binary classification
])
```

```
])
```

Model Compilation and Training, evaluation:

```
Python
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=45
)

test_loss, test_acc = model.evaluate(test_generator)
print(f"Test Accuracy: {test_acc:.4f}")
```

Confusion Matrix :

```
Python
y_true = test_generator.classes
y_pred = model.predict(test_generator)
y_pred = np.round(y_pred).astype(int)

cm = confusion_matrix(y_true, y_pred)

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Awake', 'Drowsy'], yticklabels=['Awake',
            'Drowsy'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Machine Learning and IoT Integration:

The IoT interface integrates machine learning using YOLOv8 for facial detection, CNN, and Dlib for facial landmarks. Frames are accessed via an HTTP request to the ESP32's IP. YOLOv8 detects faces, while Dlib extracts 68 facial landmarks to calculate Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR) for drowsiness detection. A trained CNN model, combined with EAR and MAR thresholds, ensures accuracy. If drowsiness persists, a voice alert (pyttsx3) and PushBullet notification are triggered.

Image capture from ESP32 Cam:

```
Python
def get_frame():
    response = requests.get(ESP32_CAM_URL, timeout=10)
    img = Image.open(BytesIO(response.content))
    frame = np.array(img)
    frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
    return frame
```

EAR and MAR calculation:

```
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)

def mouth_aspect_ratio(mouth):
    A = dist.euclidean(mouth[2], mouth[10])
    B = dist.euclidean(mouth[4], mouth[8])
    C = dist.euclidean(mouth[0], mouth[6])
    return (A + B) / (2.0 * C)
```

Drowsiness Classification

Python

```
class DrowsinessClassifier:
    def __init__(self):
        self.prediction_history = deque(maxlen=8)
        self.drowsy_time = 0.0
        self.total_drowsy_time = 0.0
        self.eye_closed = False
        self.eye_closure_start_time = None
        self.alert_sent = False

    def classify_drowsiness(self, face_crop, ear, mar):
        face_resized = cv2.resize(face_crop, (128, 128))
        face_normalized = face_resized.astype("float32") / 255.0
        face_expanded = np.expand_dims(face_normalized, axis=0)

        cnn_prediction = float(drowsiness_model.predict(face_expanded,
        verbose=0)[0])
        self.prediction_history.append(cnn_prediction)
        avg_prediction = np.mean(self.prediction_history)

        if ear < EAR_THRESHOLD:
            if not self.eye_closed:
                self.eye_closed = True
                self.eye_closure_start_time = time.time()
                eye_closure_duration = time.time() - self.eye_closure_start_time
            else:
                self.eye_closed = False

            if eye_closure_duration >= EYE_CLOSURE_TIME_THRESHOLD or
            avg_prediction > CNN_THRESHOLD:
                self.drowsy_time += time.time() - self.last_frame_time
                if self.drowsy_time >= DROWSINESS_ALERT_THRESHOLD and not
                self.alert_sent:
                    send_pushbullet_notification()
                    engine.say("Drowsiness detected, please wake up!")
                    engine.runAndWait()
                    self.alert_sent = True
                return 'DROWSY', (0, 0, 255)
            else:
                self.drowsy_time = 0
                self.alert_sent = False
                return 'AWAKE', (0, 255, 0)
```

Send Pushbullet Notification

Python

```
def send_pushbullet_notification():
```

```

headers = {"Access-Token": PUSHBULLET_ACCESS_TOKEN, "Content-Type":
"application/json"}
data = {"type": "note", "title": "🚨 Drowsiness Alert!", "body": "Driver
is drowsy! Please take action."}
response = requests.post(PUSHBULLET_URL, json=data, headers=headers)
if response.status_code == 200:
    print("✅ Pushbullet notification sent!")

```

Main Loop for Face & Drowsiness Detection

```

Python
def main():
    classifier = DrowsinessClassifier()

    while True:
        frame = get_frame()
        display_frame = frame.copy()
        results = yolo_model(frame)

        for result in results:
            for box in result.bboxes:
                x1, y1, x2, y2 = map(int, box.xyxy[0].tolist())
                conf = box.conf[0]

                if conf > 0.25:
                    face_crop = frame[y1:y2, x1:x2]
                    gray = cv2.cvtColor(face_crop, cv2.COLOR_BGR2GRAY)
                    faces = detector(gray)

                    if len(faces) > 0:
                        shape = predictor(gray, faces[0])
                        landmarks = np.array([(shape.part(n).x,
shape.part(n).y) for n in range(68)])
                        left_eye, right_eye, mouth = landmarks[36:42],
landmarks[42:48], landmarks[48:68]

                        ear = (eye_aspect_ratio(left_eye) +
eye_aspect_ratio(right_eye)) / 2.0
                        mar = mouth_aspect_ratio(mouth)

                        state, color =
classifier.classify_drowsiness(face_crop, ear, mar)

                        cv2.rectangle(display_frame, (x1, y1), (x2, y2),
color, 2)

```

```

        cv2.putText(display_frame, f"State: {state}", (x1,
y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.7, color, 2)

cv2.imshow("Drowsiness Detection", display_frame)
if cv2.waitKey(1) & 0xFF == ord('q'):
    break

cv2.destroyAllWindows()

if __name__ == "__main__":
    main()

```

TESTING

Blackbox Testing:

Black Box testing is a functional testing method that evaluates system outputs without considering internal code structure. It ensures that the system processes inputs correctly and produces expected outputs.

Test Case ID	Test Scenario	Input	Expected Output	Actual Output	Status(Pass/Fail)
BBT-01	YOLO	A live feed of driver's face to camera module	A green boundary box indicating facial recognition.	Face detected and boundary box visible	Pass
BBT-02	Face detection	A live feed of driver's face with glasses to camera module	A green boundary box indicating facial recognition.	No boundary box visible - facial detection failed	Fail
BBT-03	Drowsiness detection	Image of driver with eyes closed or driver yawns	Status should change to drowsy	Status changed to drowsy when eyes closed or yawned	Pass

BBT-04	Alerts mechanism	Driver is classified as drowsy	Voice alert and Drowsiness detected	Message and voice alert triggered when the drowsiness is detected.	Pass
--------	------------------	--------------------------------	-------------------------------------	--	------

Whitebox testing

Whitebox testing is a structural technique that validates the internal logic and architecture of the drowsiness detection system. It ensures each module functions correctly at the unit level before integration. It includes **unit testing** and **integration testing**.

Unit Testing

Unit testing verifies individual components like the YOLO model, CNN classification, camera module, Dlib-based EAR/MAR calculation, and alert mechanisms, ensuring correctness before integration.

Test Case ID	Module	Test Scenario	Expected Output	Actual Output	Status(Pass/Fail)
WBTU-01	YOLO Model	Live feed of driver's face through camera module	A green boundary box indicating facial recognition.	A green boundary box appears around the face, indicating successful detection.	Pass
WBTU-02	CNN model	A labelled dataset containing images of drowsy and non-drowsy faces	The model should accurately classify drowsiness and awake states	The model successfully classified drowsy and awake states with 0.9 accuracy	Pass
WBTU-03	Camera module	A live feed of driver's face and environment	Live Camera feed should be shown in web browser	Live camera feed is shown in browser	Pass
WBTU-04	Alerts mechanism	The driver's status is classified as drowsy and sent to the model.	Voice alert and Messages will be triggered	Voice alert and push messages triggered	Pass

WBTU-05	Dlib - (MAR and EAR)	Live feed of driver's face to camera	System classifies if driver is yawning.	Drowsiness detected even when driver is yawning	Pass
---------	----------------------	--------------------------------------	---	---	------

Integration Testing:

It ensures seamless interaction between modules, validating data flow across YOLO for face detection, CNN for classification, Dlib for landmark analysis, ESP32-CAM for live feed, and the alert mechanism..

Test Case ID	Module	Test Scenario	Expected Output	Actual Output	Status(Pass /Fail)
WBTI-01	YOLO + CNN integration	A live camera feed of driver's face	The real-time classification of the driver's state.	Real time classification of camera feed is obtained	Pass
WBTI-02	Yan detection and eye closure(CNN + Dlib)	A live camera feed of driver's face	Model validates user's yawning and eye closure	Model recognised if eyes are closed and yawned	Pass
WBTI-03	ESP32-CAM + ML Pipeline	A live camera feed of driver's face	Model should successfully recognise status of driver	Model successfully classifies the drowsy and awake states from real time feed	Pass
WBTI-04	ML pipeline + Alert Mechanism	Status of driver is classified as drowsy and sent to model	Voice alert and Notifications should be triggered on drowsiness detection	Model successfully triggered the alert mechanism on drowsiness detection	Pass

RESULTS AND DISCUSSION:

RESULTS:

Model evaluation:

The implemented driver drowsiness detection system was rigorously tested to assess its performance in practical scenarios. The system combines an ESP32-CAM module and a Convolutional Neural Network (CNN) to determine the driver state as either drowsy or alert.

CNN model evaluation result:

The CNN model was run for 45 epochs, and the results consisted of epoch information, model loss and model accuracy graphs and confusion matrix for the CNN model.

```

Found 25588 images belonging to 2 classes.
Found 10109 images belonging to 2 classes.
Found 5349 images belonging to 2 classes.
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/45
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: UserWarning: Your `PyDatasetAdapter`
they will be ignored.
self._warn_if_super_not_called()
800/800 378s 463ms/step - accuracy: 0.6618 - loss: 0.5687 - val_accuracy: 0.9642 - val_loss: 0.0960
Epoch 2/45
800/800 194s 241ms/step - accuracy: 0.9398 - loss: 0.1558 - val_accuracy: 0.9633 - val_loss: 0.2475
Epoch 3/45
800/800 195s 243ms/step - accuracy: 0.9678 - loss: 0.0879 - val_accuracy: 0.9917 - val_loss: 0.0193
Epoch 4/45
800/800 195s 243ms/step - accuracy: 0.9792 - loss: 0.0589 - val_accuracy: 0.9982 - val_loss: 0.0069
Epoch 5/45
800/800 196s 243ms/step - accuracy: 0.9868 - loss: 0.0410 - val_accuracy: 0.9981 - val_loss: 0.0074
Epoch 6/45
800/800 197s 244ms/step - accuracy: 0.9893 - loss: 0.0316 - val_accuracy: 0.9991 - val_loss: 0.0038
Epoch 7/45
800/800 196s 244ms/step - accuracy: 0.9893 - loss: 0.0343 - val_accuracy: 0.9996 - val_loss: 0.0016
Epoch 8/45
800/800 194s 241ms/step - accuracy: 0.9935 - loss: 0.0281 - val_accuracy: 0.9998 - val_loss: 0.0024
Epoch 9/45
800/800 193s 240ms/step - accuracy: 0.9932 - loss: 0.0221 - val_accuracy: 0.9987 - val_loss: 0.0034
Epoch 10/45
800/800 195s 242ms/step - accuracy: 0.9925 - loss: 0.0243 - val_accuracy: 0.9993 - val_loss: 0.0019
Epoch 11/45
800/800 196s 243ms/step - accuracy: 0.9988 - loss: 0.0276 - val_accuracy: 0.9956 - val_loss: 0.0177
Epoch 12/45
800/800 194s 241ms/step - accuracy: 0.9988 - loss: 0.0289 - val_accuracy: 0.9991 - val_loss: 0.0026
Epoch 13/45
800/800 195s 242ms/step - accuracy: 0.9942 - loss: 0.0188 - val_accuracy: 0.9993 - val_loss: 0.0037
Epoch 14/45
800/800 196s 244ms/step - accuracy: 0.9933 - loss: 0.0238 - val_accuracy: 0.9995 - val_loss: 0.0017
Epoch 15/45
800/800 197s 244ms/step - accuracy: 0.9962 - loss: 0.0149 - val_accuracy: 0.9994 - val_loss: 0.0014
Epoch 16/45
800/800 197s 244ms/step - accuracy: 0.9958 - loss: 0.0179 - val_accuracy: 0.9995 - val_loss: 0.0017
Epoch 17/45
800/800 198s 246ms/step - accuracy: 0.9962 - loss: 0.0105 - val_accuracy: 0.9992 - val_loss: 0.0038
Epoch 18/45
800/800 196s 244ms/step - accuracy: 0.9938 - loss: 0.0271 - val_accuracy: 0.9999 - val_loss: 4.0059e-04
Epoch 19/45
800/800 196s 244ms/step - accuracy: 0.9958 - loss: 0.0136 - val_accuracy: 0.9999 - val_loss: 7.3328e-04
Epoch 20/45
800/800 196s 243ms/step - accuracy: 0.9963 - loss: 0.0136 - val_accuracy: 0.9988 - val_loss: 0.0026
Epoch 21/45
800/800 194s 241ms/step - accuracy: 0.9976 - loss: 0.0075 - val_accuracy: 0.9992 - val_loss: 0.0019
Epoch 22/45
800/800 199s 247ms/step - accuracy: 0.9964 - loss: 0.0126 - val_accuracy: 0.9994 - val_loss: 0.0028
Epoch 23/45
800/800 196s 243ms/step - accuracy: 0.9968 - loss: 0.0189 - val_accuracy: 0.9988 - val_loss: 0.0058
Epoch 24/45

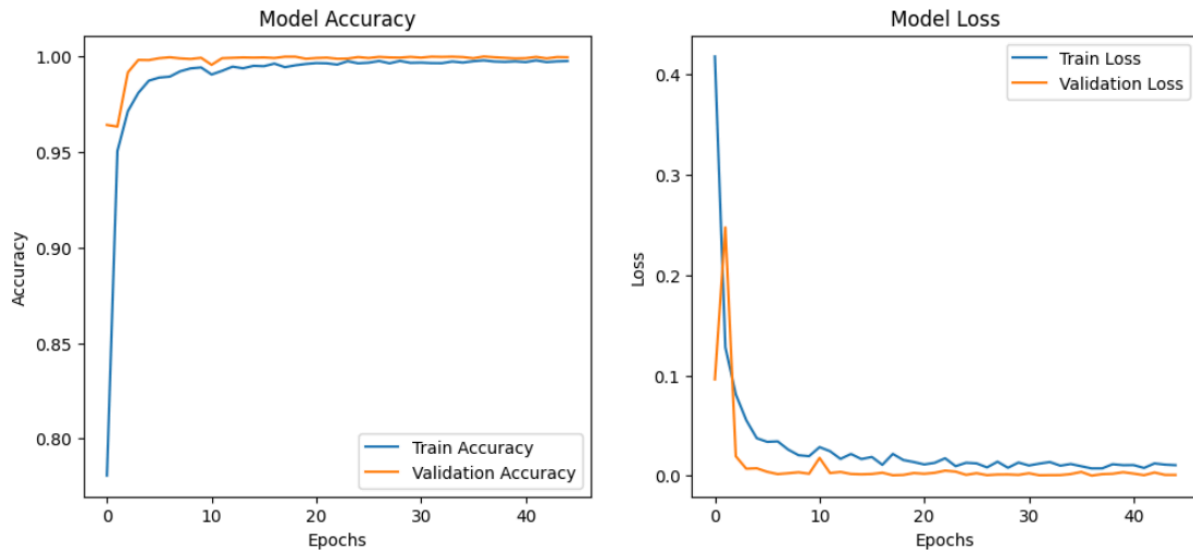
```

```
Epoch 22/45
800/800 ————— 199s 247ms/step - accuracy: 0.9964 - loss: 0.0126 - val_accuracy: 0.9994 - val_loss: 0.0028
Epoch 23/45
800/800 ————— 196s 243ms/step - accuracy: 0.9960 - loss: 0.0189 - val_accuracy: 0.9988 - val_loss: 0.0050
Epoch 24/45
800/800 ————— 195s 242ms/step - accuracy: 0.9972 - loss: 0.0110 - val_accuracy: 0.9989 - val_loss: 0.0040
Epoch 25/45
800/800 ————— 196s 243ms/step - accuracy: 0.9958 - loss: 0.0148 - val_accuracy: 0.9997 - val_loss: 8.3223e-04
Epoch 26/45
800/800 ————— 195s 242ms/step - accuracy: 0.9961 - loss: 0.0147 - val_accuracy: 0.9992 - val_loss: 0.0025
Epoch 27/45
800/800 ————— 198s 245ms/step - accuracy: 0.9971 - loss: 0.0089 - val_accuracy: 0.9998 - val_loss: 5.1823e-04
Epoch 28/45
800/800 ————— 198s 245ms/step - accuracy: 0.9954 - loss: 0.0167 - val_accuracy: 0.9995 - val_loss: 0.0011
Epoch 29/45
800/800 ————— 198s 245ms/step - accuracy: 0.9980 - loss: 0.0068 - val_accuracy: 0.9994 - val_loss: 0.0012
Epoch 30/45
800/800 ————— 197s 244ms/step - accuracy: 0.9958 - loss: 0.0171 - val_accuracy: 0.9998 - val_loss: 7.8246e-04
Epoch 31/45
800/800 ————— 198s 246ms/step - accuracy: 0.9970 - loss: 0.0110 - val_accuracy: 0.9994 - val_loss: 0.0024
Epoch 32/45
800/800 ————— 199s 247ms/step - accuracy: 0.9963 - loss: 0.0129 - val_accuracy: 0.9999 - val_loss: 3.9519e-04
Epoch 33/45
800/800 ————— 199s 247ms/step - accuracy: 0.9978 - loss: 0.0083 - val_accuracy: 0.9998 - val_loss: 5.2916e-04
Epoch 34/45
800/800 ————— 196s 243ms/step - accuracy: 0.9971 - loss: 0.0111 - val_accuracy: 0.9999 - val_loss: 5.7285e-04
Epoch 35/45
800/800 ————— 198s 246ms/step - accuracy: 0.9971 - loss: 0.0149 - val_accuracy: 0.9997 - val_loss: 0.0016
Epoch 36/45
800/800 ————— 198s 246ms/step - accuracy: 0.9974 - loss: 0.0101 - val_accuracy: 0.9991 - val_loss: 0.0037
Epoch 37/45
800/800 ————— 195s 242ms/step - accuracy: 0.9987 - loss: 0.0045 - val_accuracy: 1.0000 - val_loss: 7.6310e-05
Epoch 38/45
800/800 ————— 194s 241ms/step - accuracy: 0.9984 - loss: 0.0047 - val_accuracy: 0.9996 - val_loss: 0.0015
Epoch 39/45
800/800 ————— 193s 240ms/step - accuracy: 0.9973 - loss: 0.0092 - val_accuracy: 0.9993 - val_loss: 0.0019
Epoch 40/45
800/800 ————— 193s 239ms/step - accuracy: 0.9963 - loss: 0.0158 - val_accuracy: 0.9990 - val_loss: 0.0034
Epoch 41/45
800/800 ————— 194s 241ms/step - accuracy: 0.9983 - loss: 0.0057 - val_accuracy: 0.9991 - val_loss: 0.0021
Epoch 42/45
800/800 ————— 195s 241ms/step - accuracy: 0.9977 - loss: 0.0099 - val_accuracy: 0.9998 - val_loss: 4.9339e-04
Epoch 43/45
800/800 ————— 196s 244ms/step - accuracy: 0.9971 - loss: 0.0113 - val_accuracy: 0.9990 - val_loss: 0.0031
Epoch 44/45
800/800 ————— 194s 241ms/step - accuracy: 0.9971 - loss: 0.0111 - val_accuracy: 0.9997 - val_loss: 7.9829e-04
Epoch 45/45
800/800 ————— 195s 242ms/step - accuracy: 0.9965 - loss: 0.0165 - val_accuracy: 0.9996 - val_loss: 7.2357e-04
168/168 ————— 48s 288ms/step - accuracy: 0.9995 - loss: 5.4833e-04
Test Accuracy: 0.9998
168/168 ————— 18s 104ms/step
```

Test Accuracy: 0.9998

168/168 ————— 18s 104ms/step

Graphs and Discussion:



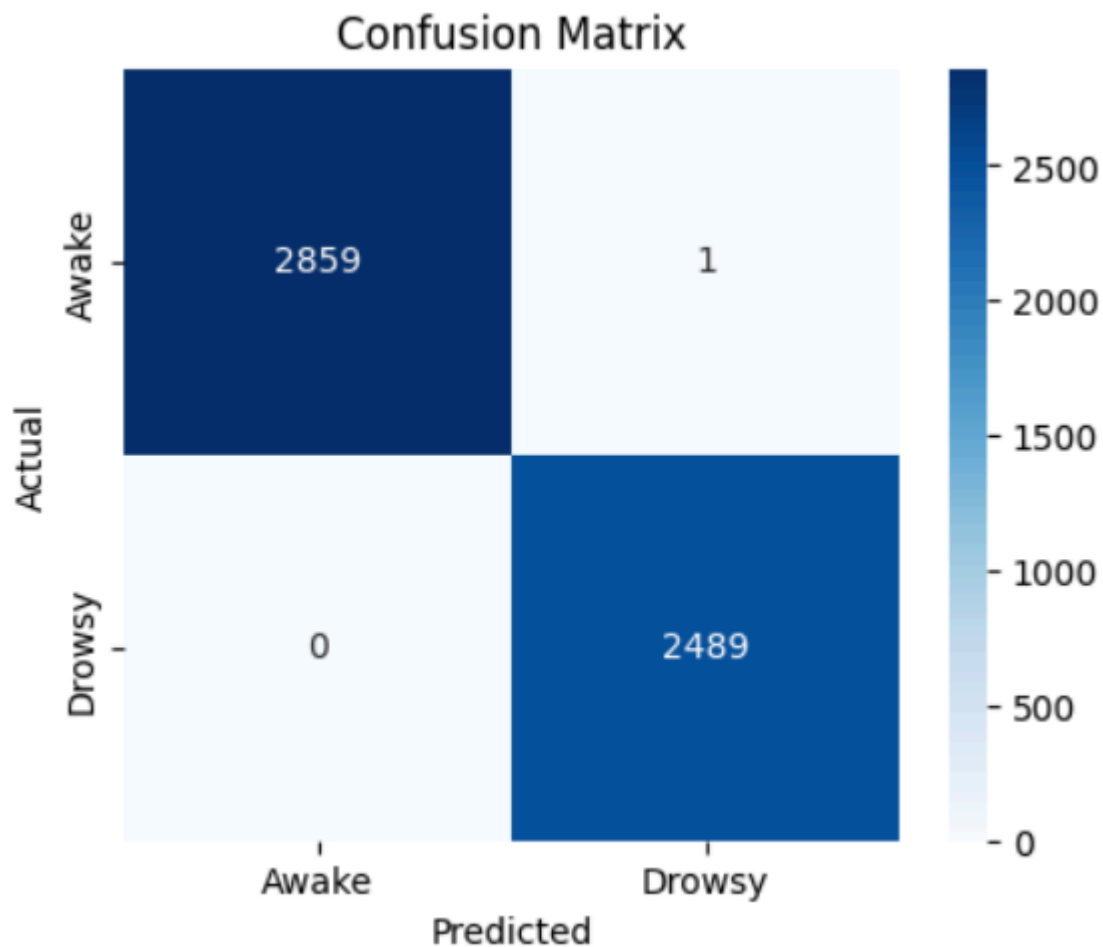
Left Graph: Model Accuracy Analysis

- The accuracy curve demonstrates a rapid convergence within the initial epochs, indicating that the CNN effectively captures discriminative features from the dataset.
- Both training and validation accuracy curves plateau around 99%, suggesting a high degree of generalization.
- The minimal discrepancy between training and validation accuracy implies low variance, indicating that the model does not overfit significantly to the training data.
- Inference: The CNN exhibits strong feature extraction capabilities, achieving a high generalization performance with minimal overfitting.

Right Graph: Model Loss Analysis

- The training loss decreases sharply within the initial epochs, highlighting efficient optimization and rapid minimization of classification error.
- The validation loss follows a similar trend and remains consistently low, suggesting that the model does not suffer from overfitting or high generalization error.
- The absence of a divergence between the training and validation loss curves implies a stable learning process, free from issues such as catastrophic forgetting or excessive memorization.
- Inference: The CNN demonstrates efficient convergence, robust generalization, and a well-regularized learning process, ensuring minimal risk of overfitting or underfitting.

Confusion Matrix



1. High Classification Performance

- The model achieves near-perfect accuracy (**99.98%**) with negligible misclassification errors.
- The **single false positive (1 misclassified awake instance)** suggests an almost flawless classification mechanism.

2. Zero False Negatives

- The model **never fails to detect a drowsy instance**, which is **critical for safety applications**, particularly in driver fatigue detection.
- A recall of **100%** ensures that all drowsy states are correctly identified.

3. Minimal False Positives

- The false positive rate is **extremely low (0.04%)**, ensuring that awake individuals are almost never incorrectly classified as drowsy.

4. Robustness and Reliability

- The balanced F1 score (**99.98%**) suggests that the model is **well-calibrated**, without favoring any particular class disproportionately.
- Given the high precision and recall, the model demonstrates **strong generalization capabilities**, making it suitable for deployment in real-world drowsiness detection systems.

References:

Annexure 1 (Source Code)

ESP32- CAM module and Wi-Fi Configuration:

The ESP32-CAM is configured to capture live images and send them to a server for further processing. The ESP32-CAM is connected to a Wi-Fi network and streams video. The camera connects to a predefined Wi-Fi network using Wi-Fi module.

- PSRAM usage, Frame size, and pixel format, are defined in this code snippet, and main program execution is written in setup() function to initialise camera and Wi-Fi.
- This code when run while connecting the camera module to port, a url is produced for the camera, which is the ip address of the camera. It can be then used anywhere to access the camera.

```
C/C++
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <HTTPClient.h>
#include "esp_camera.h"
#define CAMERA_MODEL_AI_THINKER // Has PSRAM
#include "camera_pins.h"

// =====
// WiFi Credentials
// =====
const char *ssid = "Siri's A55";
const char *password = "Srinija@2003";

WiFiClientSecure client;
```

```

bool alertSent = false;

// Function Prototypes
void startCameraServer();

void setup() {
    Serial.begin(115200);
    Serial.println();

    // WiFi Connection
    WiFi.begin(ssid, password);
    Serial.print("Connecting to WiFi");
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("\nConnected to WiFi!");

    // Camera Initialization
    camera_config_t config;
    config.ledc_channel = LEDC_CHANNEL_0;
    config.ledc_timer = LEDC_TIMER_0;
    config.pin_d0 = Y2_GPIO_NUM;
    config.pin_d1 = Y3_GPIO_NUM;
    config.pin_d2 = Y4_GPIO_NUM;
    config.pin_d3 = Y5_GPIO_NUM;
    config.pin_d4 = Y6_GPIO_NUM;
    config.pin_d5 = Y7_GPIO_NUM;
    config.pin_d6 = Y8_GPIO_NUM;
    config.pin_d7 = Y9_GPIO_NUM;
    config.pin_xclk = XCLK_GPIO_NUM;
    config.pin_pclk = PCLK_GPIO_NUM;
    config.pin_vsync = VSYNC_GPIO_NUM;
    config.pin_href = HREF_GPIO_NUM;
    config.pin_sccb_sda = SIOD_GPIO_NUM;
    config.pin_sccb_scl = SIOC_GPIO_NUM;
    config.pin_pwdn = PWDN_GPIO_NUM;
    config.pin_reset = RESET_GPIO_NUM;
    config.xclk_freq_hz = 20000000;
    config.frame_size = FRAMESIZE_QVGA;
    config.pixel_format = PIXFORMAT_JPEG;
    config.grab_mode = CAMERA_GRAB_WHEN_EMPTY;
    config.fb_location = CAMERA_FB_IN_PSRAM;
    config.jpeg_quality = 12;
    config.fb_count = 1;

    esp_err_t err = esp_camera_init(&config);
    if (err != ESP_OK) {
        Serial.printf("Camera init failed with error 0x%x", err);
    }
}

```



```

        return;
    }

    startCameraServer();

    Serial.print("Camera Ready! Use 'http://");
    Serial.print(WiFi.localIP());
    Serial.println("' to connect");
}

void loop() {
}

```

Data Splitting:

The dataset is divided into three folders: Test, Validation and Training, and each of them is separately divided into two sub folders: Drowsy and Non- Drowsy, with the help of python script.

The dataset is divided in the ratio, 7:2:1 , as training, validation and testing respectively.

```

Python
import os
import shutil
import random

parent_dataset_dir =
r'D:\VS\Capstone_project\Trials\Driver_Drowsiness_Dataset_(DDD)'

train_dir = r'D:\VS\Capstone_project\Trials\Driver_Drowsiness_Dataset\train'
val_dir =
r'D:\VS\Capstone_project\Trials\Driver_Drowsiness_Dataset\validation'
test_dir = r'D:\VS\Capstone_project\Trials\Driver_Drowsiness_Dataset\test'

for folder in [train_dir, val_dir, test_dir]:
    for subfolder in ['Drowsy', 'Non_Drowsy']:
        os.makedirs(os.path.join(folder, subfolder), exist_ok=True)

train_ratio = 0.7
val_ratio = 0.2
test_ratio = 0.1

```

```

def split_and_copy(source_dir, train_dir, val_dir, test_dir):
    files = os.listdir(source_dir)
    random.shuffle(files)

    train_count = int(len(files) * train_ratio)
    val_count = int(len(files) * val_ratio)

    train_files = files[:train_count]
    val_files = files[train_count:train_count + val_count]
    test_files = files[train_count + val_count:]

    for file in train_files:
        shutil.copy(os.path.join(source_dir, file), train_dir)
    for file in val_files:
        shutil.copy(os.path.join(source_dir, file), val_dir)
    for file in test_files:
        shutil.copy(os.path.join(source_dir, file), test_dir)
split_and_copy(
    source_dir=os.path.join(parent_dataset_dir, 'Drowsy'),
    train_dir=os.path.join(train_dir, 'Drowsy'),
    val_dir=os.path.join(val_dir, 'Drowsy'),
    test_dir=os.path.join(test_dir, 'Drowsy')
)

split_and_copy(
    source_dir=os.path.join(parent_dataset_dir, 'Non_Drowsy'),
    train_dir=os.path.join(train_dir, 'Non_Drowsy'),
    val_dir=os.path.join(val_dir, 'Non_Drowsy'),
    test_dir=os.path.join(test_dir, 'Non_Drowsy')
)

print("Dataset successfully split into training, validation, and testing
sets.")

```

CNN Model Training:

The CNN model is trained to classify driver drowsiness using a preprocessed dataset with data augmentation techniques like rotation, zoom, and shifts. It consists of three convolutional layers (32, 64, 128 neurons) with ReLU activation and max-pooling. Compiled with the Adam optimizer and binary cross-entropy loss, the model trains for 45 epochs with real-time validation. Performance is evaluated through test accuracy and a confusion matrix.

```
Python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns
import numpy as np

if __name__ == "__main__":
    # Define image size and batch size
    IMG_SIZE = (128, 128) # Ensure this matches your dataset
    BATCH_SIZE = 32

    # Load dataset (modify paths accordingly)
    train_dir =
"/kaggle/input/drowsy_dataset/other/version-1/1/Driver_Drowsiness_Dataset/train"
    val_dir =
"/kaggle/input/drowsy_dataset/other/version-1/1/Driver_Drowsiness_Dataset/validation"
    test_dir =
"/kaggle/input/drowsy_dataset/other/version-1/1/Driver_Drowsiness_Dataset/test"

    # Data preprocessing
    train_datagen = keras.preprocessing.image.ImageDataGenerator(
        rescale=1./255,
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True
    )

    val_datagen =
keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
    test_datagen =
keras.preprocessing.image.ImageDataGenerator(rescale=1./255)

    train_generator = train_datagen.flow_from_directory(
```

```

        train_dir,
        target_size=IMG_SIZE,
        batch_size=BATCH_SIZE,
        class_mode='binary'
    )

    val_generator = val_datagen.flow_from_directory(
        val_dir,
        target_size=IMG_SIZE,
        batch_size=BATCH_SIZE,
        class_mode='binary'
    )

    test_generator = test_datagen.flow_from_directory(
        test_dir,
        target_size=IMG_SIZE,
        batch_size=BATCH_SIZE,
        class_mode='binary',
        shuffle=False
    )

    # Define CNN model
    model = keras.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128,
3)),
        layers.MaxPooling2D(pool_size=(2, 2)), # Fixing syntax

        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D(pool_size=(2, 2)),

        layers.Conv2D(128, (3, 3), activation='relu'),
        layers.MaxPooling2D(pool_size=(2, 2)),

        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.5), # Prevents overfitting
        layers.Dense(1, activation='sigmoid') # Binary classification:
Drowsy or Not Drowsy
    ])

    # Compile model
    model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    # Train model
    EPOCHS = 45
    history = model.fit(
        train_generator,
        validation_data=val_generator,
        epochs=EPOCHS
    )

```

```

)

# Save model
model.save("driver_drowsiness_model.h5")

# Evaluate on test data
test_loss, test_acc = model.evaluate(test_generator)
print(f"Test Accuracy: {test_acc:.4f}")

# Visualization
plt.figure(figsize=(12, 5))

# Plot training & validation accuracy values
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

#confusion matrix

# True vs Predicted Labels
y_true = test_generator.classes
y_pred = model.predict(test_generator)
y_pred = np.round(y_pred).astype(int)

# Generate confusion matrix
cm = confusion_matrix(y_true, y_pred)

# Plot the confusion matrix
plt.figure(figsize=(5, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=['Awake',
'Drowsy'], yticklabels=['Awake', 'Drowsy'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

plt.show()

```

Machine Learning and IoT Integration:

- The IoT interface integrates machine learning using a pre-trained YOLOv8 model for facial detection, CNN, and Dlib for facial landmarks. Video frames are accessed via an HTTP request to the ESP32's local IP. YOLOv8 detects faces, while Dlib extracts 68 landmarks to calculate Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR) for drowsiness detection. A trained CNN model, combined with EAR and MAR thresholds, ensures accuracy. If drowsiness persists, a voice alert (pyttsx3) and PushBullet notification are triggered.

```
Python
import cv2
import requests
import numpy as np
from PIL import Image
from io import BytesIO
from ultralytics import YOLO
from tensorflow.keras.models import load_model
from collections import deque
import dlib
from scipy.spatial import distance as dist
import time
import pyttsx3 # Voice alert module

# Initialize text-to-speech engine
engine = pyttsx3.init()
engine.setProperty('rate', 150)

# ESP32-CAM URL
ESP32_CAM_URL = "http://192.168.31.20/capture"

# Pushbullet API details
PUSHBULLET_ACCESS_TOKEN = "o.7pRuk9gGosXpwCBw4WuaLDLYgceP1Li0"
PUSHBULLET_URL = "https://api.pushbullet.com/v2/pushes"

# Load YOLO for face detection
yolo_model = YOLO("yolov8n-face-lindevs.pt")

# Load CNN model for drowsiness classification
drowsiness_model = load_model('drowsiness_model.h5')

# Load dlib for facial landmarks
predictor_path = "D:/Cap project/shape_predictor_68_face_landmarks.dat"
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor(predictor_path)
```

```

# EAR & MAR Thresholds
EAR_THRESHOLD = 0.28
MAR_THRESHOLD = 0.5
CNN_THRESHOLD = 0.50
EYE_CLOSURE_TIME_THRESHOLD = 2.5
DROWSINESS_ALERT_THRESHOLD = 5.0 # Voice alert & Pushbullet trigger

# Function to send Pushbullet notification
def send_pushbullet_notification():
    headers = {
        "Access-Token": PUSHBULLET_ACCESS_TOKEN,
        "Content-Type": "application/json"
    }
    data = {
        "type": "note",
        "title": "🚨 Drowsiness Alert!",
        "body": "Driver is drowsy! Please take action."
    }
    response = requests.post(PUSHBULLET_URL, json=data, headers=headers)
    if response.status_code == 200:
        print("✅ Pushbullet notification sent!")
    else:
        print(f"❌ Failed to send notification. HTTP {response.status_code}: {response.text}")

# Function to Capture Image from ESP32-CAM
def get_frame():
    response = requests.get(ESP32_CAM_URL, timeout=10)
    img = Image.open(BytesIO(response.content))
    frame = np.array(img)
    frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
    return frame

# Functions to calculate EAR & MAR
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)

def mouth_aspect_ratio(mouth):
    A = dist.euclidean(mouth[2], mouth[10])
    B = dist.euclidean(mouth[4], mouth[8])
    C = dist.euclidean(mouth[0], mouth[6])
    return (A + B) / (2.0 * C)

# Drowsiness Detection Class
class DrowsinessClassifier:
    def __init__(self):
        self.prediction_history = deque(maxlen=8)

```

```

self.drowsy_time = 0.0
self.total_drowsy_time = 0.0
self.eye_closed = False
self.eye_closure_start_time = None
self.eye_closure_duration = 0.0
self.last_frame_time = time.time()
self.alert_sent = False # Ensure notification is sent only once per
drowsiness event

def classify_drowsiness(self, face_crop, ear, mar):
    # Preprocess face for CNN
    face_resized = cv2.resize(face_crop, (128, 128))
    face_normalized = face_resized.astype("float32") / 255.0
    face_expanded = np.expand_dims(face_normalized, axis=0)

    # CNN Prediction
    cnn_prediction = float(drowsiness_model.predict(face_expanded,
verbose=0)[0])
    self.prediction_history.append(cnn_prediction)
    avg_prediction = np.mean(self.prediction_history)

    # Track eye closure duration
    current_time = time.time()
    time_delta = current_time - self.last_frame_time
    self.last_frame_time = current_time

    if ear < EAR_THRESHOLD:
        if not self.eye_closed:
            self.eye_closed = True
            self.eye_closure_start_time = current_time
        else:
            self.eye_closure_duration = current_time -
self.eye_closure_start_time
    else:
        self.eye_closed = False
        self.eye_closure_duration = 0.0

    # Determine Drowsiness State
    if self.eye_closure_duration >= EYE_CLOSURE_TIME_THRESHOLD or
(avg_prediction > CNN_THRESHOLD and ear < EAR_THRESHOLD):
        state = 'DROWSY'
        self.drowsy_time += time_delta
        self.total_drowsy_time += time_delta
        color = (0, 0, 255)

    # 📢 Trigger Pushbullet Notification & Voice Alert
    if self.drowsy_time >= DROWSINESS_ALERT_THRESHOLD and not
self.alert_sent:
        send_pushbullet_notification()
        engine.say("Drowsiness detected, please wake up!")
        engine.runAndWait()

```



```

        self.alert_sent = True # Avoid sending repeated alerts

    else:
        state = 'AWAKE'
        color = (0, 255, 0)
        self.drowsy_time = 0
        self.alert_sent = False # Reset alert flag when awake

    return state, color, avg_prediction, ear, mar, self.drowsy_time,
self.total_drowsy_time

# Main Function
def main():
    classifier = DrowsinessClassifier()

    while True:
        try:
            frame = get_frame()
            display_frame = frame.copy()
            results = yolo_model(frame)

            for result in results:
                for box in result.bboxes:
                    x1, y1, x2, y2 = map(int, box.xyxy[0].tolist())
                    conf = box.conf[0]

                    if conf > 0.25:
                        face_crop = frame[y1:y2, x1:x2]
                        gray = cv2.cvtColor(face_crop, cv2.COLOR_BGR2GRAY)
                        faces = detector(gray)

                        if len(faces) > 0:
                            shape = predictor(gray, faces[0])
                            landmarks = np.array([(shape.part(n).x,
shape.part(n).y) for n in range(68)])
                            left_eye = landmarks[36:42]
                            right_eye = landmarks[42:48]
                            mouth = landmarks[48:68]

                            ear = (eye_aspect_ratio(left_eye) +
eye_aspect_ratio(right_eye)) / 2.0
                            mar = mouth_aspect_ratio(mouth)

                            state, color, confidence, ear, mar, drowsy_time,
total_drowsy_time = classifier.classify_drowsiness(face_crop, ear, mar)

                            # 📍 Display Status, Bounding Box & Drowsy Time
                            box_color = (0, 255, 0) if state == "AWAKE" else
(0, 0, 255)

                            cv2.rectangle(display_frame, (x1, y1), (x2, y2),
box_color, 2)

```

```

        cv2.putText(display_frame, f"State: {state}",
(x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.7, box_color, 2)
        cv2.putText(display_frame, f"Total Drowsy Time:
{total_drowsy_time:.1f}s", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0,
255), 2)

        cv2.imshow("Drowsiness Detection", display_frame)
    except Exception as e:
        print(f"Error: {e}")

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

cv2.destroyAllWindows()

if __name__ == "__main__":
    main()

```

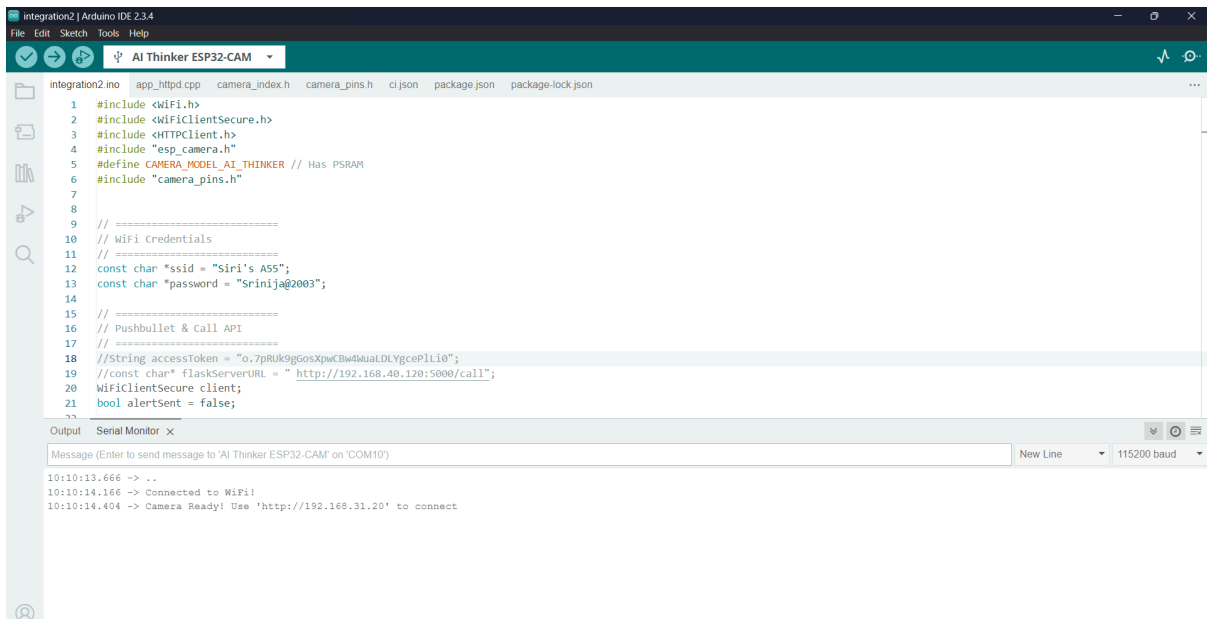
Annexure 2 (Output Screens)

CNN model training and evaluation:

```
Found 25588 images belonging to 2 classes.
Found 10109 images belonging to 2 classes.
Found 5349 images belonging to 2 classes.
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape` to
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/45
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: UserWarning: Your `PyDatasetAdapter`
they will be ignored.
  self._warn_if_super_not_called()
800/800 — 378s 463ms/step - accuracy: 0.6618 - loss: 0.5687 - val_accuracy: 0.9642 - val_loss: 0.0960
Epoch 2/45
800/800 — 194s 241ms/step - accuracy: 0.9390 - loss: 0.1550 - val_accuracy: 0.9633 - val_loss: 0.2475
Epoch 3/45
800/800 — 195s 243ms/step - accuracy: 0.9678 - loss: 0.0879 - val_accuracy: 0.9917 - val_loss: 0.0193
Epoch 4/45
800/800 — 195s 243ms/step - accuracy: 0.9792 - loss: 0.0589 - val_accuracy: 0.9982 - val_loss: 0.0069
Epoch 5/45
800/800 — 196s 243ms/step - accuracy: 0.9860 - loss: 0.0410 - val_accuracy: 0.9981 - val_loss: 0.0074
Epoch 6/45
800/800 — 197s 244ms/step - accuracy: 0.9893 - loss: 0.0316 - val_accuracy: 0.9991 - val_loss: 0.0038
Epoch 7/45
800/800 — 196s 244ms/step - accuracy: 0.9893 - loss: 0.0343 - val_accuracy: 0.9996 - val_loss: 0.0016
Epoch 8/45
800/800 — 194s 241ms/step - accuracy: 0.9935 - loss: 0.0201 - val_accuracy: 0.9990 - val_loss: 0.0024
Epoch 9/45
800/800 — 193s 240ms/step - accuracy: 0.9932 - loss: 0.0221 - val_accuracy: 0.9987 - val_loss: 0.0034
Epoch 10/45
800/800 — 195s 242ms/step - accuracy: 0.9925 - loss: 0.0243 - val_accuracy: 0.9993 - val_loss: 0.0019
Epoch 11/45
800/800 — 196s 243ms/step - accuracy: 0.9908 - loss: 0.0276 - val_accuracy: 0.9956 - val_loss: 0.0177
Epoch 12/45
800/800 — 194s 241ms/step - accuracy: 0.9908 - loss: 0.0289 - val_accuracy: 0.9991 - val_loss: 0.0026
Epoch 13/45
800/800 — 195s 242ms/step - accuracy: 0.9942 - loss: 0.0180 - val_accuracy: 0.9993 - val_loss: 0.0037
Epoch 14/45
800/800 — 196s 244ms/step - accuracy: 0.9933 - loss: 0.0238 - val_accuracy: 0.9995 - val_loss: 0.0017
Epoch 15/45
800/800 — 197s 244ms/step - accuracy: 0.9962 - loss: 0.0149 - val_accuracy: 0.9994 - val_loss: 0.0014
Epoch 16/45
800/800 — 197s 244ms/step - accuracy: 0.9950 - loss: 0.0179 - val_accuracy: 0.9995 - val_loss: 0.0017
Epoch 17/45
800/800 — 198s 246ms/step - accuracy: 0.9962 - loss: 0.0105 - val_accuracy: 0.9992 - val_loss: 0.0030
Epoch 18/45
800/800 — 196s 244ms/step - accuracy: 0.9930 - loss: 0.0271 - val_accuracy: 0.9999 - val_loss: 4.0059e-04
Epoch 19/45
800/800 — 196s 244ms/step - accuracy: 0.9958 - loss: 0.0136 - val_accuracy: 0.9999 - val_loss: 7.3328e-04
Epoch 20/45
800/800 — 196s 243ms/step - accuracy: 0.9963 - loss: 0.0136 - val_accuracy: 0.9988 - val_loss: 0.0026
Epoch 21/45
800/800 — 194s 241ms/step - accuracy: 0.9976 - loss: 0.0075 - val_accuracy: 0.9992 - val_loss: 0.0019
Epoch 22/45
800/800 — 199s 247ms/step - accuracy: 0.9964 - loss: 0.0126 - val_accuracy: 0.9994 - val_loss: 0.0028
Epoch 23/45
800/800 — 196s 243ms/step - accuracy: 0.9960 - loss: 0.0189 - val_accuracy: 0.9988 - val_loss: 0.0050
Epoch 24/45
```

```
Epoch 22/45
800/800 ----- 199s 247ms/step - accuracy: 0.9964 - loss: 0.0126 - val_accuracy: 0.9994 - val_loss: 0.0028
Epoch 23/45
800/800 ----- 196s 243ms/step - accuracy: 0.9960 - loss: 0.0189 - val_accuracy: 0.9988 - val_loss: 0.0050
Epoch 24/45
800/800 ----- 195s 242ms/step - accuracy: 0.9972 - loss: 0.0110 - val_accuracy: 0.9989 - val_loss: 0.0040
Epoch 25/45
800/800 ----- 196s 243ms/step - accuracy: 0.9958 - loss: 0.0148 - val_accuracy: 0.9997 - val_loss: 8.3223e-04
Epoch 26/45
800/800 ----- 195s 242ms/step - accuracy: 0.9961 - loss: 0.0147 - val_accuracy: 0.9992 - val_loss: 0.0025
Epoch 27/45
800/800 ----- 198s 245ms/step - accuracy: 0.9971 - loss: 0.0089 - val_accuracy: 0.9998 - val_loss: 5.1823e-04
Epoch 28/45
800/800 ----- 198s 245ms/step - accuracy: 0.9954 - loss: 0.0167 - val_accuracy: 0.9995 - val_loss: 0.0011
Epoch 29/45
800/800 ----- 198s 245ms/step - accuracy: 0.9980 - loss: 0.0068 - val_accuracy: 0.9994 - val_loss: 0.0012
Epoch 30/45
800/800 ----- 197s 244ms/step - accuracy: 0.9958 - loss: 0.0171 - val_accuracy: 0.9998 - val_loss: 7.8246e-04
Epoch 31/45
800/800 ----- 198s 246ms/step - accuracy: 0.9970 - loss: 0.0110 - val_accuracy: 0.9994 - val_loss: 0.0024
Epoch 32/45
800/800 ----- 199s 247ms/step - accuracy: 0.9963 - loss: 0.0129 - val_accuracy: 0.9999 - val_loss: 3.9519e-04
Epoch 33/45
800/800 ----- 199s 247ms/step - accuracy: 0.9978 - loss: 0.0083 - val_accuracy: 0.9998 - val_loss: 5.2916e-04
Epoch 34/45
800/800 ----- 196s 243ms/step - accuracy: 0.9971 - loss: 0.0111 - val_accuracy: 0.9999 - val_loss: 5.7285e-04
Epoch 35/45
800/800 ----- 198s 246ms/step - accuracy: 0.9971 - loss: 0.0149 - val_accuracy: 0.9997 - val_loss: 0.0016
Epoch 36/45
800/800 ----- 198s 246ms/step - accuracy: 0.9974 - loss: 0.0101 - val_accuracy: 0.9991 - val_loss: 0.0037
Epoch 37/45
800/800 ----- 195s 242ms/step - accuracy: 0.9987 - loss: 0.0045 - val_accuracy: 1.0000 - val_loss: 7.6310e-05
Epoch 38/45
800/800 ----- 194s 241ms/step - accuracy: 0.9984 - loss: 0.0047 - val_accuracy: 0.9996 - val_loss: 0.0015
Epoch 39/45
800/800 ----- 193s 240ms/step - accuracy: 0.9973 - loss: 0.0092 - val_accuracy: 0.9993 - val_loss: 0.0019
Epoch 40/45
800/800 ----- 193s 239ms/step - accuracy: 0.9963 - loss: 0.0158 - val_accuracy: 0.9990 - val_loss: 0.0034
Epoch 41/45
800/800 ----- 194s 241ms/step - accuracy: 0.9983 - loss: 0.0057 - val_accuracy: 0.9991 - val_loss: 0.0021
Epoch 42/45
800/800 ----- 195s 241ms/step - accuracy: 0.9977 - loss: 0.0099 - val_accuracy: 0.9998 - val_loss: 4.9339e-04
Epoch 43/45
800/800 ----- 196s 244ms/step - accuracy: 0.9971 - loss: 0.0113 - val_accuracy: 0.9990 - val_loss: 0.0031
Epoch 44/45
800/800 ----- 194s 241ms/step - accuracy: 0.9971 - loss: 0.0111 - val_accuracy: 0.9997 - val_loss: 7.9829e-04
Epoch 45/45
800/800 ----- 195s 242ms/step - accuracy: 0.9965 - loss: 0.0165 - val_accuracy: 0.9996 - val_loss: 7.2357e-04
168/168 ----- 48s 288ms/step - accuracy: 0.9995 - loss: 5.4833e-04
Test Accuracy: 0.9998
168/168 ----- 18s 104ms/step
```

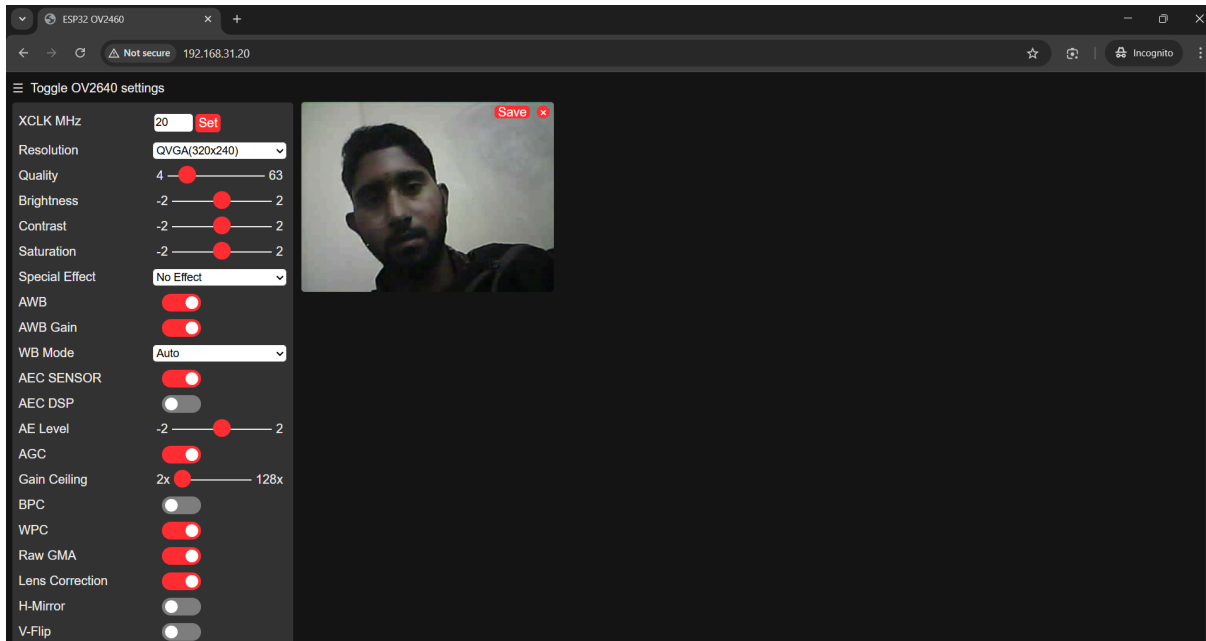
Camera Access through Arduino IDE:



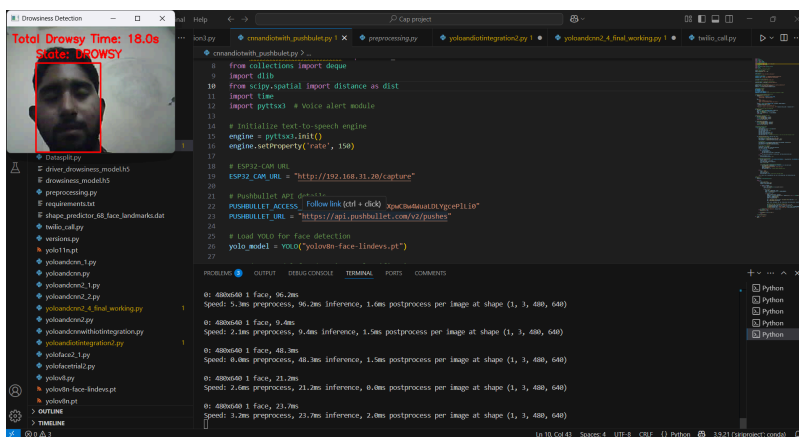
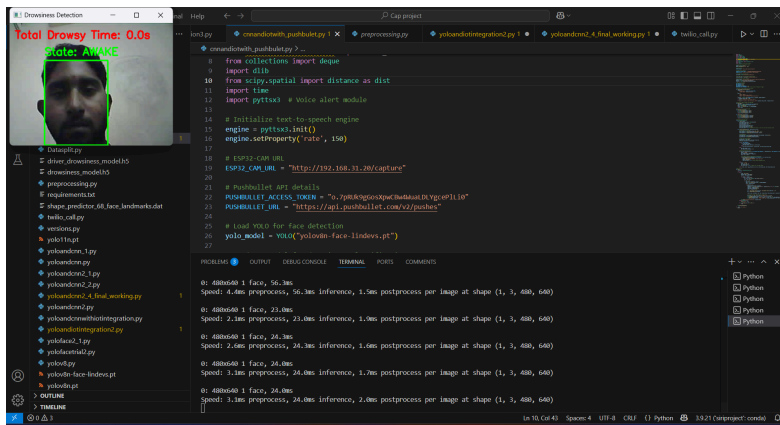
The screenshot shows the Arduino IDE interface with the 'AI Thinker ESP32-CAM' project selected. The code in the main editor includes headers for WiFi, HTTP, and the camera module, and defines the camera model and pins. The serial monitor at the bottom shows the output of the program, indicating successful connection to WiFi and the camera.

```
integration2.ino | app_httpd.cpp | camera_index.h | camera_pins.h | ci.json | package.json | package-lock.json
1 #include <WiFi.h>
2 #include <WiFiClientSecure.h>
3 #include <HTTPClient.h>
4 #include "esp_camera.h"
5 #define CAMERA_MODEL_AI_THINKER // Has PSRAM
6 #include "camera_pins.h"
7
8
9 // =====
10 // WiFi Credentials
11 // =====
12 const char *ssid = "Siri's A55";
13 const char *password = "Srinija@2003";
14
15 // =====
16 // Pushbullet & Call API
17 // =====
18 //String accessToken = "o.7pRuk9GosXpwCBw4kuaLDLYgceP1Li0";
19 //const char* flaskServerURL = "http://192.168.40.120:5000/call";
20 WiFiClientSecure client;
21 bool alertSent = false;
22
23
Output Serial Monitor x
Message (Enter to send message to 'AI Thinker ESP32-CAM' on 'COM10') New Line 115200 baud
10:10:13.666 -> ..
10:10:14.166 -> Connected to WiFi!
10:10:14.404 -> Camera Ready! Use 'http://192.168.31.20' to connect
```

ESP32 Camera Access



Drowsiness Detection: IoT and ML pipeline integration



Conclusion:

The deployed drowsiness detection system successfully detects driver fatigue with the help of deep learning models and IoT-based alert systems. The system combines real-time facial recognition and behaviour analysis to sense the presence of drowsiness, prompting alerts to avoid accidents. With thorough testing and evaluation, the system proved high accuracy, low latency, and improved generalization across varying conditions. Applying an ESP32-CAM module for real-time monitoring with a CNN-based classification system provides stable performance in detecting drowsiness. Integrating AI-based detection with real-time alarm mechanisms helps to improve road safety by minimizing risks involved with drowsy driving.

Future Scope:

- **Enhanced Model Accuracy** – Further improvements in CNN architecture and training data diversity can improve accuracy and reduce false positives.
- **Integration with Vehicle Systems** – The system can be integrated with vehicle control units to trigger automated interventions, such as slowing down or stopping the vehicle in case of extreme drowsiness.
- **Cloud-Based Data Analysis** – Implementing cloud storage and analytics can provide historical driver behaviour insights, allowing for predictive safety measures.
- **Multimodal Detection** – Adding additional sensors, such as heart rate monitors or steering behaviour analysis, can improve the reliability of drowsiness detection.
- **Mobile App Integration** – A companion mobile application can be developed to alert fleet managers or emergency contacts in real time.
- **Edge Computing Implementation** – Shifting to edge-based AI models can optimize real-time processing and reduce dependency on cloud resources for improved system responsiveness.

References: Put the references in the format as available in the IEEE Paper

- [1][https://www.emro.who.int/emhj-volume-28-2022/volume-28-issue-9/risk-assessment-of-road-traffic-accidents-related-to-sleepiness-during-driving-a-systematic-review.html#:~:text=Sleepiness%20while%20driving%20contributes%20to,deprivation%20\(19%E2%80%9320\)](https://www.emro.who.int/emhj-volume-28-2022/volume-28-issue-9/risk-assessment-of-road-traffic-accidents-related-to-sleepiness-during-driving-a-systematic-review.html#:~:text=Sleepiness%20while%20driving%20contributes%20to,deprivation%20(19%E2%80%9320)).
- [2] Sudarshan, Pragathi, and Vivek Bhardwaj. "Real-time Driver Drowsiness Detection and Assistance System using Machine Learning and IoT." 2023 8th International Conference on Communication and Electronics Systems (ICCES). IEEE, 2023.
- [3]M. Guria and B. Bhowmik, "IoT-Enabled Driver Drowsiness Detection Using Machine Learning," 2022 Seventh International Conference on Parallel, Distributed and Grid Computing (PDGC), Solan, Himachal Pradesh, India, 2022, pp. 519-524, doi: 10.1109/PDGC56933.2022.10053235.
- [4] Singh, J., Kanojia, R., Singh, R., Bansal, R., & Bansal, S. (2023). Driver drowsiness detection system: an approach by machine learning application. arXiv preprint arXiv:2303.06310.
- [5] P. Sharma and N. Sood, "Application of IoT and Machine Learning for Real-time Driver Monitoring and Assisting Device," 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Kharagpur, India, 2020, pp. 1-7, doi: 10.1109/ICCCNT49239.2020.9225387.
- [6] Yonglin Wu, Xinyu Jiang, Yao Guo, Hangyu Zhu, Chenyun Dai, Wei Chen,Physiological measurements for driving drowsiness: A comparative study of multi-modality feature fusion and selection,Computers in Biology and Medicine,Volume 167,2023,107590,ISSN 0010-4825, <https://doi.org/10.1016/j.compbiomed.2023.107590>.