



**JOMO KENYATTA UNIVERSITY OF AGRICULTURE AND
TECHNOLOGY.**

BACHELOR OF SCIENCE IN INFORMATION TECHNOLOGY

DROWSY DRIVER ALERT SYSTEM

JOAN MULIARO

SCT2221-C004-0067/2022

THIS PROJECT HAS BEEN SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF THE BACHELOR OF SCIENCE IN INFORMATION TECHNOLOGY AT JOMO KENYATTA UNIVERSITY OF AGRICULTURE AND TECHNOLOGY FOR THE YEAR 2025

DECLARATION

I, Joan Muliaro, hereby declare that this project, titled "

DROWSY DRIVER ALERT SYSTEM," is my original work and has not been submitted in part or in full for any other degree or diploma at this or any other university.

All sources of information, including books, articles, websites, and personal communications, have been duly acknowledged and referenced in accordance with the academic standards of Bachelor of Science in Information Technology at Jomo Kenyatta University of Agriculture and Technology (JKUAT).

I further declare that I have adhered to all ethical guidelines and principles of academic integrity in the conduct of this research and the preparation of this project.

.....

Signature

.....

Date

Registration Number: SCT221-C004-0067/2022

Student's Name: Joan Muliaro

The undersigned certify that this is the final year project for the above-named student under my supervision and that it has been submitted to Jomo Kenyatta University of Agriculture and Technology (JKUAT) with my approval

Supervisor's Name: Dr. Judy Gateri

.....

.....

Signature

Date

ACKNOWLEDGMENT

The successful completion of this "DROWSY DRIVER ALERT SYSTEM" project would not have been possible without the generous support and contributions of several individuals. I am profoundly grateful to all of them.

My deepest gratitude goes to my supervisor, Dr. Judy Gateri, for their unwavering dedication, expert guidance, and continuous motivation throughout this endeavor. Their profound knowledge and patient mentorship were indispensable to the successful execution of this research.

I extend my sincere thanks to Jomo Kenyatta University of Agriculture and Technology (JKUAT) and the School of Computing and Information Technology for providing an excellent academic environment, state-of-the-art facilities, and the necessary resources that facilitated this project.

I am immensely thankful to my beloved family and friends, whose enduring encouragement, understanding, and belief in me provided the strength and solace needed during challenging times.

DEDICATION

This project is dedicated to the memory of those whose lives have been tragically cut short or irrevocably altered by road accidents, and to their families and communities who carry the burden of these losses.

It is also dedicated to the relentless efforts of advocates, researchers, and policymakers committed to enhancing road safety. Special acknowledgment is extended to the diligent drivers and transport professionals enduring long hours and challenging conditions, and to the innovative engineers and developers creating solutions to mitigate drowsy driving. It is my hope that this work contributes meaningfully to fostering a safer driving environment for everyone.

Contents

| | |
|--|----|
| DECLARATION | 2 |
| ACKNOWLEDGMENT | 4 |
| DEDICATION | 5 |
| CHAPTER 1: INTRODUCTION | 9 |
| 1.1 Background | 9 |
| 1.2 Problem Statement..... | 10 |
| 1.3 Research Questions | 10 |
| 1.4 Objectives | 11 |
| 1.4.1 General Objective | 11 |
| 1.4.2 Specific objectives | 11 |
| 1.5 Project Justification..... | 12 |
| 1.6 Project scope..... | 12 |
| 1.7 Project Constraints..... | 12 |
| CHAPTER 2: LITERATURE REVIEW | 13 |
| 2.1 Introduction to the Literature Review | 13 |
| 2.1.1 Challenges of Drowsy Driving Detection. | 13 |
| 2.2 Existing Drowsy Driving Detection Measures. | 13 |
| 2.2.1 Subjective measures (SM): | 14 |
| 2.3.2 Vehicle-based measures (VBM): | 14 |
| 2.3.3 Physiological (Biological) measures (PM): | 15 |
| 2.3.4 Behavioral measures (BM): | 15 |
| 2.3.5 Hybrid measures (HM): | 17 |
| 2.3.6 Choice of the Driver Drowsiness Detection Measure | 17 |
| 2.4 Technologies and Platforms to Be Used..... | 17 |
| 2.4.1 HAAR CASCADE | 17 |
| 2.4.2 Deep Learning (DL) Models and TinyML | 22 |
| 2.4.3 Transfer Learning | 23 |
| 2.4.5 Running TFLite Interpreter: | 29 |
| 2.4.6 Models' Conversion to C Array: | 29 |
| CHAPTER 3: METHODOLOGIES | 30 |
| 3.1 System Development Methodology..... | 30 |
| I. Business Understanding | 30 |
| II. Data Understanding | 32 |
| III. Data Preparation | 32 |
| IV. Modeling | 33 |

| | |
|---|-----------|
| V. Evaluation..... | 34 |
| VI. Deployment..... | 34 |
| Crisp-DM Methodology Justification | 34 |
| 3.2 Data Set | 36 |
| 3.2.1 Eye Dataset..... | 36 |
| 3.2.2 ESP32 Cam / Laptop Webcam..... | 36 |
| 3.3 Model Development, Deployment and Testing Steps. | 36 |
| 3.3.1 Phase 1: Data collection and Preprocessing | 36 |
| 3.3.2 Phase 2: Model Training and Evaluation..... | 36 |
| 3.3.3 Phase 3: Haar Cascade..... | 37 |
| 3.3.4 Phase 4: Transfer Learning and Tuning of Pretrained Model..... | 37 |
| 3.3.5 Phase 5: Detect face and eyes and check that, if the person’s left eye and right eye are closed or open..... | 37 |
| 3.3.6 Phase 6: Alert the driver..... | 37 |
| 3.4 Software | 37 |
| 3.5 Hardware | 37 |
| 3.6 SYSTEM ANALYSIS | 38 |
| 3.6.1 Feasibility | 38 |
| 3.6.2 Requirements Gathering..... | 39 |
| Modeling Tools And Techniques | 39 |
| 3.6.3 Requirements Analysis..... | 40 |
| 3.6.4 System Design | 41 |
| CHAPTER 4: IMPEMENTATION | 47 |
| 4.1 Approach to Create Drowsiness detection system:..... | 47 |
| 4.1.1 Download Driver Drowsiness Dataset..... | 47 |
| 4.1.2 Create a model. | 47 |
| 4.1.3 Main file ‘Drowsy driver detection system’. | 48 |
| CHAPTER 5: CONCLUSION..... | 50 |
| Recommendations | 51 |
| Appendix A: Sample Code..... | 54 |
| Appendix B: User Manual | 66 |
| Introduction..... | 66 |
| System Requirements | 66 |
| Installation | 66 |
| Setup and Running the System | 67 |
| Using the System | 67 |
| Troubleshooting | 68 |

| | |
|--------------------------------|-----------|
| Safety Precautions..... | 68 |
| Conclusion | 68 |

CHAPTER 1: INTRODUCTION

1.1 Background

Deaths from traffic accidents are a catastrophe that goes unnoticed. In addition to the terrible human cost, road safety affects every aspect of our lives every day, including commuting to and from work and school, thus action on the issue must be taken quickly. Driving while drowsy is a risky mix of being tired and driving or driving when exhausted, which can have a number of underlying causes, such as excessive drowsiness, shift work adjustments, lack of sleep, exhaustion, and prescription drugs with sedatives and drinking alcohol after a long day. These elements' combined effects have serious consequences on reaction speeds, performance, attentiveness, memory, and focus.

Data from the 2023 Economic Survey shows that 4,690 road deaths were recorded in 2022 compared to 4,579 in 2021, indicating a 2.4 percent increase and an increase of 111 fatalities. Additional data from the National Transportation and Safety Authority (NTSA) demonstrates that the nation recorded 3,609 deaths as of October 2023, an 8.9% decrease from until 2022, a year in which 3,936 deaths were recorded. This is due to the efforts being made by interested parties to reduce road violence.

Organizations have come up with technology solutions like Lane Departure Warning Systems. This technology uses sensors to monitor a vehicle's position on the road. If a driver unintentionally drifts out of their lane without signalling, the system warns them through sounds or vibrations. Example is the Audi A7 it relies on machine vision technology. An image recognition algorithm is developed specifically for identifying road lanes and changes in vehicle positioning, Advanced driver assistance. (n.d.). IIHS-HLDI Crash Testing and Highway Safety. Another technology that organizations have used to solve the problem is An Automatic brake system that automatically apply the brakes if they detect an imminent collision. Using radar and sensors, the system monitors the road ahead to avoid accidents. Example is General Motors AEB system, used in various models like the Chevrolet Malibu and Cadillac CT6 Rammohan, A., Chavhan, S., Chidambaram, R. K., Manisaran, N., & Kumar, K. V. P. (2022). Also, the Government have conducted public awareness campaigns and enforced stricter laws as measures to combat driver fatigue and drowsiness.

Drowsy drivers are more likely to becoming involved in an accident that results in injury or death, making them a crucial target group for sleep-related interventions such a detection system for drowsy drivers. The solution would include building a system capable of real-time monitoring of driver behaviour, continuously analysing facial features and issuing timely alerts when signs of drowsiness are detected. This will be done by developing and implementing algorithms for facial landmark detection and mapping to accurately track and monitor driver drowsiness indicators, such as eye closure and blink frequency. The Model to be used is Convolutional Neural Network (CNN) it is a type of deep learning algorithm that is particularly well-suited for image recognition and processing tasks. The architecture of CNNs is inspired by the visual processing in the human brain, and they are well-suited for capturing hierarchical patterns and spatial dependencies within images.

1.2 Problem Statement

Road crash fatalities remain a critical issue, claiming thousands of lives annually and impacting daily activities such as commuting to work or school. Driving while drowsy is a risky mix of being tired and driving or driving when exhausted, a significant contributor to these accidents is drowsy driving, a condition resulting from factors such as sleep deprivation, shift work adjustments, sedative medications or sleep pills, excessive drowsiness, exhaustion and drinking alcohol after a long day. These factors impair performance, alertness, and reaction times, leading to increased accident risks. Despite a reported decrease in road fatalities in 2023, the challenge of effectively managing driver fatigue persists, particularly in the commercial transportation sector. Existing regulations on driving hours and rest periods fall short in enforcement and efficacy. In 2023 Economic Survey shows that 4,690 road deaths were recorded in 2022 compared to 4,579 in 2021, indicating a 2.4 percent increase and an increase of 111 fatalities. Additional data from the National Transportation and Safety Authority (NTSA) demonstrates that the nation recorded 3,609 deaths as of October 2023, an 8.9% decrease from until 2022, a year in which 3,936 deaths were recorded. Road safety is hindered by the lack of efficient systems for identifying and addressing driver drowsiness in real time. The effectiveness and execution of current measures to combat driver fatigue and drowsiness are limited. Examples of these measures include laws governing driving hours and rest intervals. Thus, a system that can alert the drivers when they seem drowsy needs to be developed.

1.3 Research Questions

- i. How significant is drowsy driving as a factor in road accidents in Kenya compared to other causes?
- ii. What are the limitations or challenges of existing drowsiness detection technologies in Kenya's transportation environment?
- iii. What features would drivers and transport authorities in Kenya prioritize in a drowsy driving detection system?
- iv. How feasible is the integration of facial recognition and other alarm-based technologies for detecting drowsiness in Kenyan road conditions?

1.4 Objectives

1.4.1 General Objective

To develop a real-time drowsy driver detection system capable of monitoring and analyzing drivers' facial features, such as eye closure and blink frequency, to promptly detect signs of drowsiness and issue alerts, thereby enhancing road safety.

1.4.2 Specific objectives

- i. Design User-friendly Interface: Develop an intuitive and user-friendly interface for the Drowsy Driver detection System that is accessible to drivers of varying technical expertise and preferences.
- ii. Implement Facial Recognition Algorithms: Develop and implement algorithms for facial landmark detection and mapping to accurately track and monitor driver drowsiness indicators, such as eye closure and blink frequency.
- iii. Create Real-time Monitoring System: Build a system capable of real-time monitoring of driver behaviour, continuously analysing facial features and issuing timely alerts when signs of drowsiness are detected.
- iv. Test and Validate the System: Conduct thorough testing and validation of the Drowsy driver detection System.
- v. To study drowsiness as a contributor to road accidents by conducting comprehensive research on road safety, particularly within Kenyan.
- vi. Evaluate existing technologies and methodologies used for detecting and preventing driver drowsiness to identify gaps for improvement.

vii. Investigate User Requirements: Investigate the needs and preferences of potential users, including drivers and transportation authorities, to inform the design and development of an effective drowsy driving detection System tailored to local requirements.

viii. Explore the feasibility of integrating various technologies like facial recognition and alarm systems, to develop a reliable and efficient solution for detecting and preventing driver drowsiness.

1.5 Project Justification

The drowsy driver detection System addresses a critical issue of road safety by mitigating the risks associated with driver drowsiness, which is a significant contributor to road accidents. By implementing this system, lives can be saved, injuries can be prevented, and economic losses resulting from road accidents can be minimized. The system aligns with the government's commitment to improving road safety and enhancing transportation infrastructure.

1.6 Project scope

The project involves the development and implementation of a drowsy driver detection System capable of detecting signs of drowsiness in real-time using facial landmark detection. It encompasses the design and deployment of software algorithms for facial landmark detection, mapping, alarm triggering, and notification. The system will be designed to be easily installable in vehicles, with minimal technical requirements and user-friendly interfaces. Initial testing and validation will be conducted to ensure the system's effectiveness and reliability in detecting driver drowsiness.

1.7 Project Constraints

Environmental factors such as varying lighting conditions, weather, and road vibrations may affect the system's performance and reliability, posing constraints during real-world deployment. The success of the project depends on user acceptance and adoption, necessitating consideration of user preferences, usability, and feedback throughout the development process. The system's functionality may be constrained by the capabilities of available hardware and software tools, as well as limitations in data processing speed and accuracy.

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction to the Literature Review

Drowsy driving, is the dangerous combination of sleepiness and driving or driving while fatigued, and can result from several underlying causes, including excessive sleepiness, changes due to shift work, sleep deprivation, fatigue, medications with sedatives and consumption of alcohol when tired. The cumulative effects of these factors have severe effects on performance, alertness, memory, concentration and reaction times, Audi A7 relies on machine vision technology. (2025). According to data from the 2023 economic survey, 4,690 deaths were reported on our roads in 2022 as compared 4,579 in 2021, presenting an increase of 111 fatalities and an increase of 2.4 percent. Further Statistics from the National Transport and Safety Authority (NTSA) show that as of October 2023, the country reported 3,609 deaths marking an 8.9 per cent drop compared to 2022 where 3,936 fatalities were reported in the same period. This can be attributed to efforts being made

10

by stakeholders to tame road carnage, General Motors AEB system. (2025). There is therefore need to come up with a way of detecting Drowsy driving to prevent accidents.

2.1.1 Challenges of Drowsy Driving Detection.

Accurate and timely detection of drowsiness using non-intrusive methods, especially in real- time driving scenarios has been a challenge due to non- availability of smart small devices that can be installed in vehicles. Secondly, the difference in Symptoms where drowsiness manifests differently among individuals, makes it challenging to develop universal detection methods. Thirdly, environmental Factors such as lighting, road conditions, and vehicle vibrations can affect the accuracy of detection systems.

2.2 Existing Drowsy Driving Detection Measures.

There are two approaches, intrusive and non-intrusive, to identify drowsy driving. In an intrusive approach, the drowsy condition of a person is identified using physiological parameters, but in a non-intrusive approach, this is identified by installing devices and sensors on the vehicle. Based on intrusive and non-intrusive approaches, five different measures are utilized to identify driver

drowsiness at an early stage Bajaj, J. S., Kumar, N., Kaushal, R. K., Gururaj, H. L., Flammini, F., & Natarajan, R. (2023, January 23). These measures are as follows:

2.2.1 Subjective measures (SM):

The Stanford Sleeping Scale (SSS) and Karolinska Sleeping Scale (KSS) approaches help to gather the driver's observations while operating the vehicle to assess the driver's state. In SSS, seven levels of the Likert scale (1-feeling active to 7-extremely sleepy), and in KSS, nine levels of the Likert scale (1-extremely alert to 9-extremely sleepy) help to evaluate the different levels of drowsiness at a particular time. The drawback of a subjective measure is that it is impractical and produces biased results, making it impossible to utilize in real driving conditions Bajaj, J. S., Kumar, N., Kaushal, R. K., Gururaj, H. L., Flammini, F., & Natarajan, R. (2023, January 23).

2.3.2 Vehicle-based measures (VBM):

The two most common vehicle-based measures for driver drowsiness detection are Standard Deviation of Lane Positioning (SDLP) and Steering Wheel Movement (SWM). In SDLP, a camera is mounted on the front of the vehicle to track the lane position, which helps to identify the alert or drowsy state. Its biggest drawback is needing to rely on external variables like road markings, lighting and weather conditions. In SWM, various sensors positioned on the vehicle's steering wheel gather information to aid in the detection of driver drowsiness. The primary issue associated with SWM is that it is expensive and has a high false positive detection rate, making it ineffective in real driving conditions. In comparison, three significant challenges face vehicle-based DDD systems. First, weather conditions. If a driver is driving in harsh

weather, with strong wind or rain, the car will naturally deviate out of the lane, leading the system to generate false results. Another challenge is the geometric conditions of roads. In some cases, the driver may drive on a steep and bumpy road, causing the car to vibrate, divert, and the steering wheel to shake, causing the collected data to become unreliable. Furthermore, some vehicle-based systems do not extract drowsiness signs precisely, leading to inaccurate detection results. Thus, many systems use an additional measure with vehicle measures, in order to increase the system's accuracy Bajaj, J. S., Kumar, N., Kaushal, R. K., Gururaj, H. L., Flammini, F., & Natarajan, R. (2023, January 23).

2.3.3 Physiological (Biological) measures (PM):

A number of devices are directly attached to the driver to capture the relevant physiological parameters, such as electroencephalograms (EEG), electrocardiograms (ECG), electromyograms (EMG) and electrooculograms (EOG), Galvanic Skin Response (GSR) sensor. Although physiological measures have a high level of accuracy, they are very intrusive. Using these highly intrusive devices in real driving conditions is challenging. However, the main issue with such systems is the equipment and sensors associated with them. Such equipment types are not comfortable, in some cases, as they must be attached to the driver's body during the journey. Additionally, the used biological sensors are vulnerable to slight movement, which may produce some noise in the extracted signals, reducing the accuracy. Another challenge that may specifically affect physiological systems is the hardware complexity and limitations. Where, because of the instability of the used dry electrodes the hardware may not collect as much data as required. There could also be issues of Electromagnetic interference (EMI) which affects the reliability and accuracy of such a measure. More complex sensors need to be employed for better system performance, in order to provide multimodal information, i.e., additional features that can enhance accurate sleepiness detection, Bajaj, J. S., Kumar, N., Kaushal, R. K., Gururaj, H. L., Flammini, F., & Natarajan, R. (2023, January 23).

2.3.4 Behavioral measures (BM):

Behavioral measures are based on the driver's features, such as eyes, mouth and head inclination. To identify drowsy driving, the researcher primarily focuses on eye blink rate and percentage of eye closure (PERCLOS), which are further examined by machine learning (ML) and deep learning (DL) algorithms Albadawi, Y., Takruri, M., & Awad, M. (2022). Other signs that can aid in detecting drowsiness in a driver include yawning and head movement. Due to their non-intrusive characteristics, these behavioral measurement techniques are frequently used in simulated and real driving conditions. The present state of the art reveals that behavioral measures are more accurate

than vehicle-based measures.

2.3.5 Hybrid measures (HM):

Hybrid measures are a combination of two or more measures.

Figure 1 depicts the various measures for driver drowsiness detection and their respective techniques.

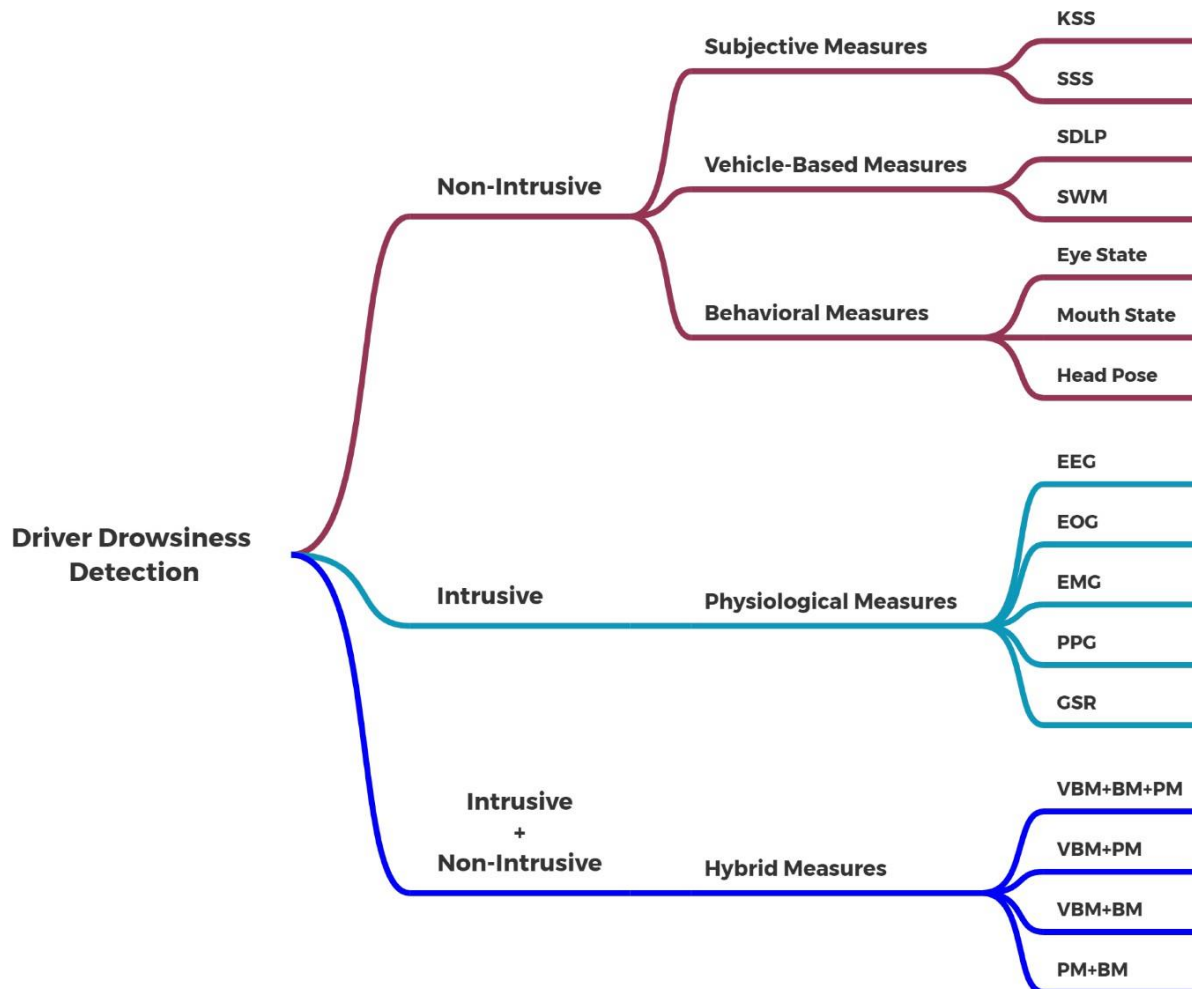


Fig 1. Driver drowsiness detection measures

2.3.6 Choice of the Driver Drowsiness Detection Measure

In this research project the Behavioral Measure (BM) will be used because of its non-intrusive nature.

2.4 Technologies and Platforms to Be Used

2.4.1 HAAR CASCADE

Haar Cascade Classifier is one of the few object detection methods with the ability to detect faces. It offers high-speed computation depending on the number of pixels inside the rectangle feature and not depending on each pixel value of the image. This method has four steps for

detecting an object namely as Haar-like feature, integral image, AdaBoost learning and Cascade Classifier. For the detection of the face, Haar features are the main part of the Haar Cascade Classifier. The Haar features are used to detect the presence of a feature in given image. Each feature results in a single value, which is calculated by the sum of pixels under black rectangle. The Haar-like feature is a rectangular feature providing specific indication to an image for rapid face detection. Figure 2 shows the examples of common variety of Haar- like features.

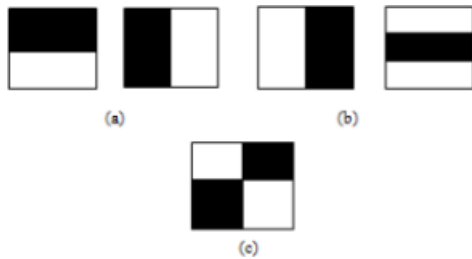


Fig2. (a) Edge feature, (b) Line feature and (c) Four-Triangle feature

In obtaining object detection value, Haar-like feature value was calculated using integral image. It starts scanning the image for the detection of the face from the top left corner and ends the face detection process at the right bottom of image in order to detect the face from an image as shown in Figure 3 (a). The integral image could calculate values accurately and relatively quick by creating new presentation of image by using value of region previously scanned by specific Haar-like feature as depicted in Figure 3. The value at any point (x,y) is the summed area table of the sum off all the pixels above and to the left of (x,y), inclusive as shown in Equation (2).

$$I(x, y) = \sum_{x^i < x} \sum_{y^i < y} i(x^i, y^i) \dots \dots \text{Equation 2.}$$

Where $i(x,y)$ is the value of pixel at (x,y) whereas $I(x,y)$ is the sum of integral of pixel values. The value of integral image, $I(x,y)$ is obtained by sum value previous index, starting from the left top until right bottom. Moreover, the summed-area table can be computed efficiently in a single pass

over the image, as the value in the summed-area table at (x,y) in Equation (2)

| | | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| 1 | 3 | 7 | 7 | 3 |
| 2 | 1 | 3 | 3 | 1 |
| 3 | 5 | 9 | 9 | 5 |
| 4 | 3 | 6 | 6 | 3 |

(a)

| | | | | |
|---|----|----|----|----|
| | 1 | 2 | 3 | 4 |
| 1 | 3 | 10 | 17 | 20 |
| 2 | 4 | 14 | 24 | 28 |
| 3 | 9 | 28 | 47 | 56 |
| 4 | 12 | 37 | 62 | 74 |

(b)

Fig 3. (a) Input image and (b) Integral image

Once the summed-area table had been computed, evaluating the sum of intensities over any rectangular area requires exactly four array references regardless of the area size. Equation (3) shows the sum of $i(x,y)$ over the rectangle spanned by A, B, C and D.

$$\sum_{x_0 < x < x_1} \sum_{y_0 < y < y_1} i(x,y) = I(D) + I(A) - I(B) - I(C) \dots \text{Equation 3}$$

The notation in the Figure 4 shows the example of the description of computing a sum in the summed-area table data structure having A= (x0, y0), B= (x1, y0), C= (x0, y1) and D= (x1, y1).

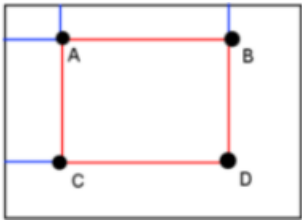


Fig 4. Description of computing a sum in the summed-area table data.

Value which has been calculated by using integral image would then be compared with threshold value of specific features provided by AdaBoost. This should be completed to find potential features because not all features were relevant to use for specific object detection. AdaBoost combines potential features called weak classifier to become strong classifier. The Cascade classifier can be divided into two which are strong classifier and weak classifier. Weak classifier means less accurate or also irrelevant prediction and strong classifier means more accurate or relevant prediction. Strong classifier made by AdaBoost can detect object level by level on a cascade Zhao Z et al (2020).

2.4.2 Deep Learning (DL) Models and TinyML

The images output from the haar-Perclos are used as inputs for deep learning models to detect the driver's status. Recent research indicates that there is effort being made towards integrating DL with Edge devices. However, DL learning models have high computational costs and require a lot of training time. Furthermore, they required a large quantity of data for quality predictions.

The approximate range of the DL model size in the literature on driver drowsiness detection is from 10 MB to 54 MB. This poses a challenge to deployment on Edge devices that have resource constraints, such as smartphones or Raspberry Pi or ESP32 Cam which are recommended for use in detecting drivers' drowsiness, as these devices have limited battery energy capacity, small memory size, and limited processor capacity. On the other hand, some studies have used devices that have resource constraints but deployed the DL models to a cloud- based platform, which also poses challenges in terms of latency.

To tackle these challenges, a new and emerging technology called Tiny Machine Learning (TinyML) has paved the way to meet the challenges of integrating DL with Edge devices. TinyML can be defined as an integration of two concepts: machine learning or deep learning and Edge devices. TinyML technology enables the deployment of DL models on resource- constrained devices powered by microcontrollers.

TinyML can be applied to a driver drowsiness detection case in order to overcome the challenge of integrating DL with Edge devices by producing lightweight DL models with a few kilobytes and, thus, enable their deployment on Edge devices that have resource constraints, such as microcontrollers Kamarudi N et al (2019).

At present, we can define TinyML as a subfield of machine learning that applies machine learning and deep learning models to embedded systems running on microcontrollers, digital signal processors, or other ultra-low-power specialized processors. Technically, these embedded systems must have a power consumption lower than 1 mW so that they can run for weeks, months, or even years without recharging or battery replacement. The following features characterize a TinyML application:

1. It runs on a microcontroller, digital signal processor, low-power microcomputer hosting embedded Linux, or a mobile platform with explicitly limited RAM, flash memory, and battery power to implement a machine learning model.
2. It runs on ultra-low power devices with power consumption in mW or μ W while deriving inferences from the 24×7 running machine learning model.
3. It must be implementing machine learning at the edge of the network, therefore not requiring to transfer or exchange data with a cloud or server, i.e., the machine learning model must be executed within the edge device without any need of data communication over a network. The network used by the device must be used only to communicate results of inferences from the machine learning model to a server/cloud/controller if required.
4. The TinyML machine learning model must have a minimal footprint, typically a few tens of kilobytes, to be run on microcontrollers and microcomputers.

At present, TensorFlow Lite is synonymous with TinyML as there is no other machine-learning framework for microcontrollers. TensorFlow Lite has been developed to run on Android, iOS, Embedded Linux, and microcontrollers Alajlan N.N and Ibrahim D. M (2023).

2.4.3 Transfer Learning

Transfer learning is the reuse of a pre-trained model on a new problem. It's currently very popular in deep learning because it can train deep neural networks with comparatively little data. A pre-trained model is a saved network that was previously trained on a large dataset, typically on a large-scale image-classification task. You either use the pretrained model as is or use transfer learning to customize the model to a given task.

The intuition behind transfer learning for image classification is that if a model is trained on a large and general enough dataset, this model will effectively serve as a generic model of the

visual world. You can then take advantage of these learned feature maps without having to start from scratch by training a large model on a large dataset.

There are two ways to customize a pretrained model

1. Feature Extraction: Use the representations learned by a previous network to extract meaningful features from new samples. You simply add a new classifier, which will be trained from scratch, on top of the pretrained model so that you can repurpose the feature maps learned previously for the dataset.

You do not need to (re)train the entire model. The base convolutional network already contains features that are generically useful for classifying pictures. However, the final, classification part of the pretrained model is specific to the original classification task, and subsequently specific to the set of classes on which the model was trained.

2. Fine-Tuning: Unfreeze a few of the top layers of a frozen model base and jointly train both the newly-added classifier layers and the last layers of the base model. This allows us to "fine-tune" the higher-order feature representations in the base model in order to make them more relevant for the specific task Agnihotri, N. (2023, May 17).

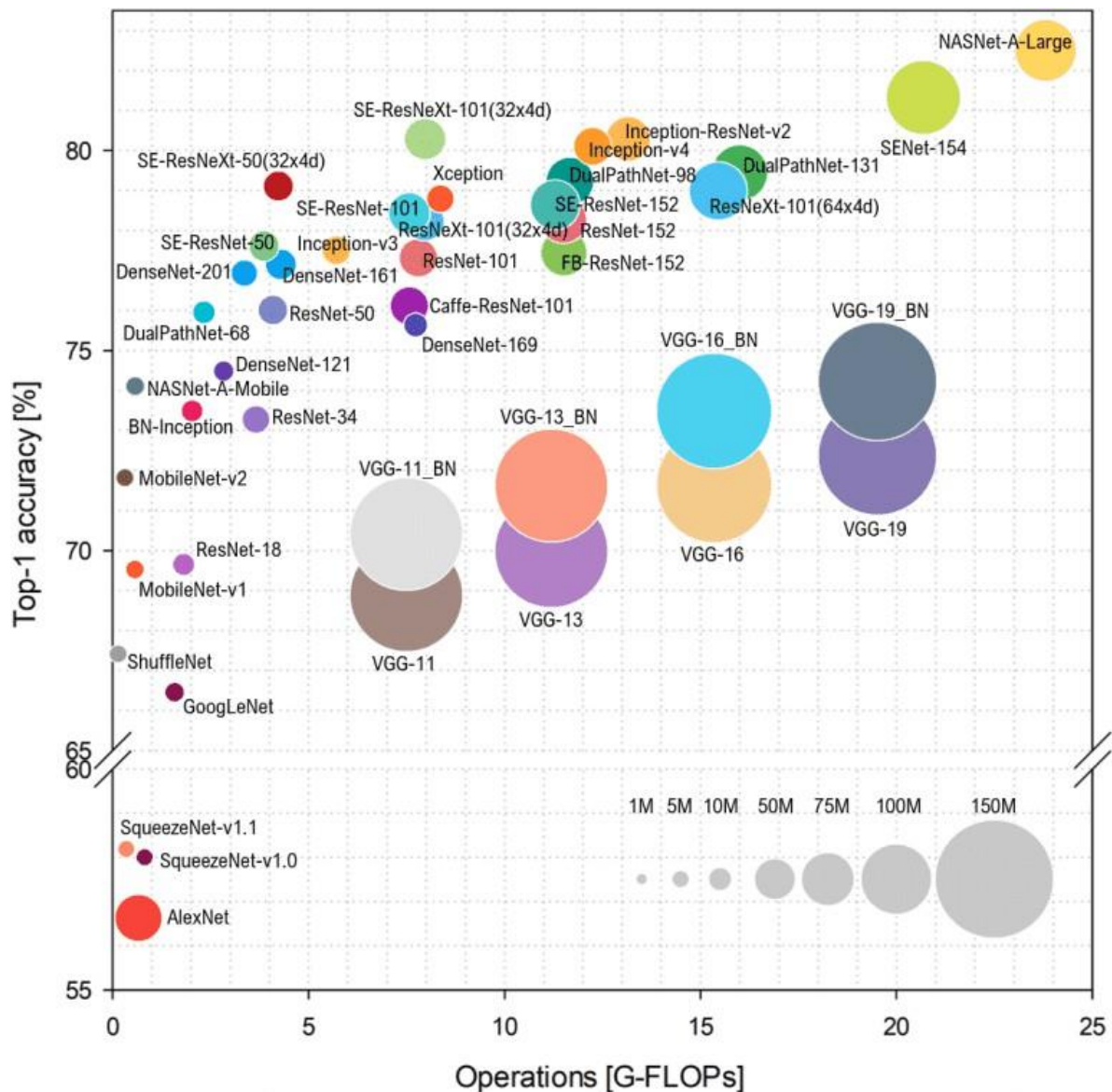


Fig 5. Overview of architectures until 2018, Donges, N. (2024, August 15).

Fig 5 above shows an overview of pretrained deep learning models. The pretrained models that have lower Floating-point Operations Per second (G-FLOPS) and higher accuracy would be candidates for deployment and Edge devices. Note that, the Floating-point Operations Per second (FLOPs) indicate the complexity of the model, while on the vertical axis we have the Imagenet accuracy. The radius of the circle indicates the number of parameters

Some pre-trained light weight models that can be used for driver drowsy detection are SqueezeNet, AlexNet, and CNN, MobileNet-V2 and MobileNet-V3, with their pre-training conducted on the

ImageNet dataset

A Convolutional Neural Network (CNN) is a type of deep learning algorithm that is particularly well-suited for image recognition and processing tasks. It is made up of multiple layers, including convolutional layers, pooling layers, and fully connected layers. The architecture of CNNs is inspired by the visual processing in the human brain, and they are well-suited for capturing hierarchical patterns and spatial dependencies within images.

Convolutional Layers: These layers apply convolutional operations to input images, using filters (also known as kernels) to detect features such as edges, textures, and more complex patterns. Convolutional operations help preserve the spatial relationships between pixels.

Pooling Layers: Pooling layers down sample the spatial dimensions of the input, reducing the computational complexity and the number of parameters in the network. Max pooling is a common pooling operation, selecting the maximum value from a group of neighboring pixels.

Activation Functions: Non-linear activation functions, such as Rectified Linear Unit (ReLU), introduce non-linearity to the model, allowing it to learn more complex relationships in the data.

Fully Connected Layers: These layers are responsible for making predictions based on the high-level features learned by the previous layers. They connect every neuron in one layer to every neuron in the next layer.

CNNs are trained using a large dataset of labeled images, where the network learns to recognize patterns and features that are associated with specific objects or classes. Proven to be highly effective in image-related tasks, achieving state-of-the-art performance in various computer vision applications. Their ability to automatically learn hierarchical representations of features makes them well-suited for tasks where the spatial relationships and patterns in the data are crucial for accurate predictions. CNNs are widely used in areas such as image classification, object detection, facial recognition, and medical image analysis.

The convolutional layers are the key component of a CNN, where filters are applied to the input image to extract features such as edges, textures, and shapes.

The output of the convolutional layers is then passed through pooling layers, which are used to down-sample the feature maps, reducing the spatial dimensions while retaining the most

important information. The output of the pooling layers is then passed through one or more fully connected layers, which are used to make a prediction or classify the image, Costa, F. (2022, January 5).

2.4.4 Model Optimization and Conversion:

This phase aims to optimize the saved deep learning models by reducing the size of models, after which they are converted to Tensor Flow Lite (TFLite) format. Quantization aims to convert the weights of models or activation or both from 32-bit floating-point numbers to 8-bit integer format, which provides a significant reduction in model size, thus leading to a decrease in the device's memory footprint in devices. Secondly, the deep learning models are converted to TFLite format (.tflite) to enable inference of these models.

In order to shrink a regular Deep Learning model to a tiny model to fit in an embedded device, there are several frameworks available:

1. AIMET from Qualcomm
2. TensorFlow Lite from Google
3. CoreML from Apple
4. PyTorch Mobile from Facebook

Tensorflow developed a package called tensorflow model optimization. This toolkit is a python package with a suite of techniques for developers to optimize their models for either latency, size, or a balance of the previous two. Each technique has its own API built on top of Keras which makes it incredibly straightforward to use.

Shrinking Techniques:

When we talk about model optimization or size reduction, we're mainly talking about:

1. Reducing the number of weights
2. Reducing the number of bits per weight, Deep Convolutional Neural Networks (AlexNet) — Dive into Deep Learning 1.0.3 documentation. (n.d.).

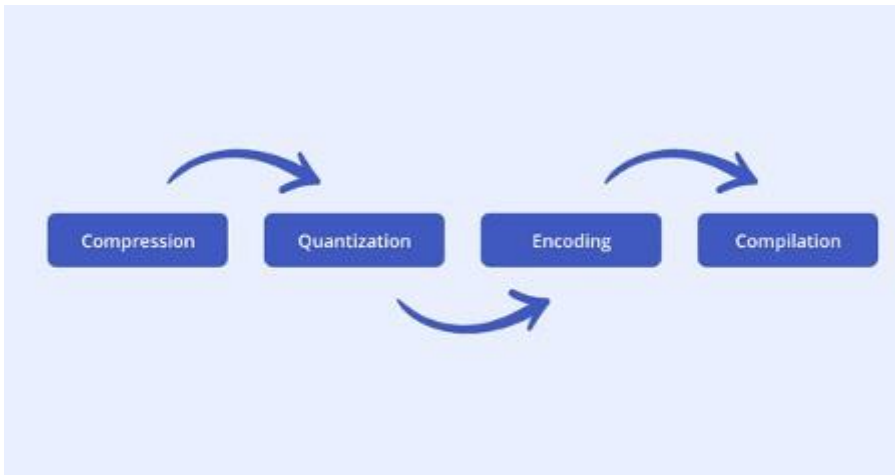


Fig 7 General steps towards optimization of the model

2.4.5 Running TFLite Interpreter:

In this phase, the TensorFlow Lite (TFLite) Interpreter is run to evaluate the TFLite deep learning models on the host computer. This is conducted by loading the TFLite model to the Interpreter and using the testing dataset for evaluation.

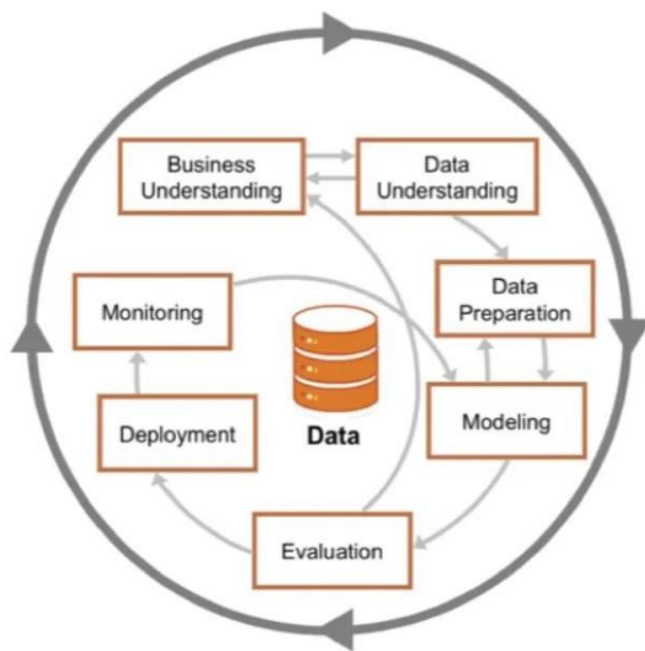
2.4.6 Models' Conversion to C Array:

In this phase, the TFLite models are converted using TensorFlow Lite (TFLite) Micro tools that convert deep learning models to C byte array. This is conducted by using `XXD` to generate a C source file for the TFLite models as a char array. The models are then deployed into an independent platform using C++ language in Arduino software and compiled with IoT devices as microcontrollers

CHAPTER 3: METHODOLOGIES

3.1 System Development Methodology

The Project employs Crisp-DM Methodology. CRISP-DM or Cross Industry Standard Process for Data Mining is a process model with six phases that naturally describes the data science life cycle.



I. Business Understanding

The Business Understanding phase focuses on understanding the objectives and requirements of the project.

1. Determine business objectives: You should first “thoroughly understand, from a business perspective, what the customer really wants to accomplish.” (CRISP-DM Guide) and then define business success criteria.
2. Assess situation: Determine resources availability, project requirements, assess risks and contingencies, and conduct a cost-benefit analysis.

3. Determine project goals: In addition to defining the business objectives, you should also define what success looks like from a technical project perspective.
4. Produce project plan: Select technologies and tools and define detailed plans for each project phase.

Set my project objectives, goals and even made my project schedule to follow.

II. Data Understanding

Drives the focus to identify, collect, and analyze the data sets that can help you accomplish the project goals. This phase also has four tasks:

1. Collect initial data: Acquire the necessary data and load it into your analysis tool.
2. Describe data: Examine the data and document its surface properties like data format, number of records, or field identities.
3. Explore data: Dig deeper into the data. Query it, visualize it, and identify relationships among the data.
4. Verify data quality: How clean/dirty is the data? Document any quality issues.

Needed dataset to train the model. The dataset used was yawn-eye-dataset, downloaded from the link: [yawn_eye_dataset_new \(kaggle.com\)](https://www.kaggle.com/yawn-eye-dataset).

In this dataset you will see the train and test folder in which there are subfolders (closed, open, yawn, no_yawn) which consist of 617 images of each closed eyes, open eyes, etc. I only needed the open and closed folders.

III. Data Preparation

This phase prepares the final data set(s) for modeling. It has five tasks:

1. Select data: Determine which data sets will be used and document reasons for inclusion/exclusion.
2. Clean data: A common practice during this task is to correct, impute, or remove erroneous values.
3. Construct data: Derive new attributes that will be helpful.
4. Integrate data: Create new data sets by combining data from multiple sources.
5. Format data: Re-format data as necessary. For example, convert string values that store numbers to numeric values so that you can perform mathematical operations.

To create a model first set a path to the dataset downloaded. The dataset had four subfolders inside, train and test folders. From those four folders only needed open and closed folders to train the model on. After setting the path, preprocessed the images and performed certain

operations so that it was ready for training.

IV. Modeling

Build and assess model. This phase has four tasks:

1. Select modeling techniques: Determine which algorithms to use e.g. regression, neural net).
2. Generate test design: Pending your modeling approach, you might need to split the data into training, test, and validation sets.
3. Build model.
4. Assess model.

Used keras to build a model using Convolutional Neural Networks (CNN). CNN is a type of deep neural network algorithm which performs very accurately and efficiently for image classification.

V. Evaluation

Evaluation phase looks more broadly at which model best meets the project and what to do next.

This phase has three tasks:

1. Evaluate results: Does the model meet the project goals
2. Review process: Review the work accomplished. Was anything overlooked? Were all steps properly executed? Summarize findings and correct anything if needed.
3. Determine next steps: Based on the previous three tasks, determine whether to proceed to deployment, iterate further, or initiate new projects.

VI. Deployment

Depending on the requirements, the deployment phase can be as simple as generating a report or as complex as implementing a repeatable project.

Implement a drowsy driver detection system.

Crisp-DM Methodology Justification

Why Use CRISP-DM?

1. Versatility: It's industry-agnostic, meaning it can be applied across various sectors and business problems.
2. Structured Approach: It provides a clear roadmap for data mining projects, ensuring that crucial steps aren't overlooked.

3. Iterative: Given its cyclical nature, it encourages continuous improvement. If a problem arises in one phase, you can loop back to an earlier phase, Hotz, N. (2024, December 9).

3.2 Data Set

3.2.1 Eye Dataset

This dataset [yawn_eye_dataset_new \(kaggle.com\)](https://www.kaggle.com/yawn-eye-dataset-new) was used to Fine-tune the pretrained model.

3.2.2 ESP32 Cam / Laptop Webcam.

The camera took videos of the drivers faces to feed into the systems.

3.3 Model Development, Deployment and Testing Steps.

3.3.1 Phase 1: Data collection and Preprocessing

The dataset used was yawn-eye-dataset, which was downloaded from the link: [yawn_eye_dataset_new \(kaggle.com\)](https://www.kaggle.com/yawn-eye-dataset-new). So, to create a model first set a path to the dataset. When you open the dataset, you see there are four subfolders inside train and test folders. From those four folders only needed open and closed folders to train the model on. After setting the path, preprocessed the images and performed certain operations so that it would be ready for training.

3.3.2 Phase 2: Model Training and Evaluation

Used keras to build models using Convolutional Neural Networks (CNN). After preprocessing of images, created a model which will classify whether a person's eyes are open or closed. After training the model, saved the final weights and model architecture file as "models/my_model.keras". Convolutional Neural Network consists of input layer, output layer, and hidden layers. So, this operation is performed on these hidden layers using a filter that performs 2D matrix multiplication on the layers.

The CNN model architecture created consist of following layers:

Convolutional layer; 64 nodes, kernel size 3

Convolutional layer; 64 nodes, kernel size 3

Convolutional layer; 64 nodes, kernel size 3

Fully connected layer; 128 nodes

The output layer is also a fully connected layer with 2 nodes. In all the layers, we use the activation function Relu except the output layer. In the output layer, we have used SoftMax as an activation function.

3.3.3 Phase 3: Haar Cascade

Set a path of haar cascade files to detect face, detect left eye, and detect right eye.

3.3.4 Phase 4: Transfer Learning and Tuning of Pretrained Model

load the pre-trained model, and use OpenCV to access the webcam that will capture each frame.

3.3.5 Phase 5: Detect face and eyes and check that, if the person's left eye and right eye are closed or open.

Predict if eyes are open or closed. The images taken from the frames are by the model to perform prediction to know the state of the eye.

3.3.6 Phase 6: Alert the driver.

If the person's left eye and right eye are closed, time will increase and if time increases more than 4 the alert sound will start. If both eyes are open the time will be 0 and the alarm will stop ringing.

3.4 Software

- a. OpenCV - Haar cascade
- b. TENSOR FLOW

3.5 Hardware

- a. ESP32
- b. LAPTOP

3.6 SYSTEM ANALYSIS

This section involves analyzing the feasibility, requirements, and constraints associated with the Drowsy driver detection system.

3.6.1 Feasibility

3.6.1.1 Feasibility Study

Since drowsy driving is a major cause of serious traffic accidents, there is a growing requirement for drowsiness prevention technologies. This study proposes a drowsy driving prediction method based on eye opening time. One issue of using eye opening time is predicting strong drowsiness before the driver actually feels sleepy. Because overlooking potential hazards is one of the causes of traffic accidents and is closely related to driver cognition and drowsiness, this study focuses on eye opening movements during driving. First, this report describes hypotheses concerning drowsiness and eye opening time based on the results of previous studies. It is assumed that the standard deviation of eye opening time (SDEOP) indicates driver drowsiness and the following two transitions are considered: increasing and decreasing SDEOP. To confirm the hypotheses, the relationship between drowsiness and SDEOP was investigated. The two transitions were observed in preliminary experiments on a test course (number of drivers: 7, speed: 80 km/h). A drowsy driving prediction method was then developed based on the hypotheses. The proposed method has upper and lower thresholds, and predicts drowsiness when SDEOP crosses one of the thresholds. The thresholds are determined by an adaptation session to address individual differences in SDEOP. Finally, experiments on the test course (number of drivers: 10, speed: 80 km/h) confirmed that this method has the potential to predict strong drowsiness 5 to 25 minutes in advance, Albadawi, Y., Takruri, M., & Awad, M. (2022).

3.6.1.2 Technical Feasibility

Computer vision technology is a non-intrusive method to monitor drowsiness. It uses one or multiple cameras to monitor driver's face and eye. Eye-tracking can be seen as a special case of computer vision based drowsiness detection which focuses on drivers' eye movements, especially eye blink and percentage of eye closure. Eye blink is directly associated with drowsiness. Eye blink is often detected using advanced computer vision together with devoted and specially-designed camera. For example, an IR camera to detect eye blinks by tracking pupil. The combined computer vision algorithm with devoted camera approach suffers from many limitations. For example, the camera system cannot reliably detect face, eye, and eye blinks at nighttime, for unevenly lighted faces, and for dark skin colored users. Users are not

very willing to purchase expensive devoted system to monitor drowsiness. Thus, in this article, we proposed a new wearable proximity-sensor approach to detect eye blinks and drowsiness, in a hope to address the limitations of the camera-based blink/drowsiness detection system .

3.6.1.3 Economic Feasibility

Economic feasibility includes output performance, information, and information reliability. Financial, rather than creative, examines the projects in an economic context. It will find out whether the overall investment in the system is recoverable. With the hardware and software ready to go, no monetary investment is needed; the only expense is for system implementation. This system has no cost over the long term .

3.6.1.4 Schedule Feasibility:

Determining whether the project can be completed within the specified timeframe, considering resource availability and project dependencies.

3.6.2 Requirements Gathering

This stage involves gathering and analyzing the requirements for the Drowsy driver detection System, including functional and non-functional requirements.

Modeling Tools And Techniques

The chosen design method for this project is the Unified Modeling Language (UML), which provides a standardized and widely accepted approach to system modeling. UML is a visual modeling language that allows for the representation of different aspects of a system, including its structure, behaviour, and interactions. It provides a set of graphical notations and techniques that enable the creation of diagrams to depict various system components and their relationships.

3.6.3 Requirements Analysis

These are the primary functions to be accomplished by the system. They are the following;

3.6.3.1 Functional Requirements

1. Take image input from the camera
2. Detect face and eyes in the image.
3. Create a Region of Interest (ROI), for both detected face and eyes.
4. Feed this to our classifier(model), which will categorize whether eyes are open or closed.
5. At last, calculate the time to check if the person is drowsy or not.

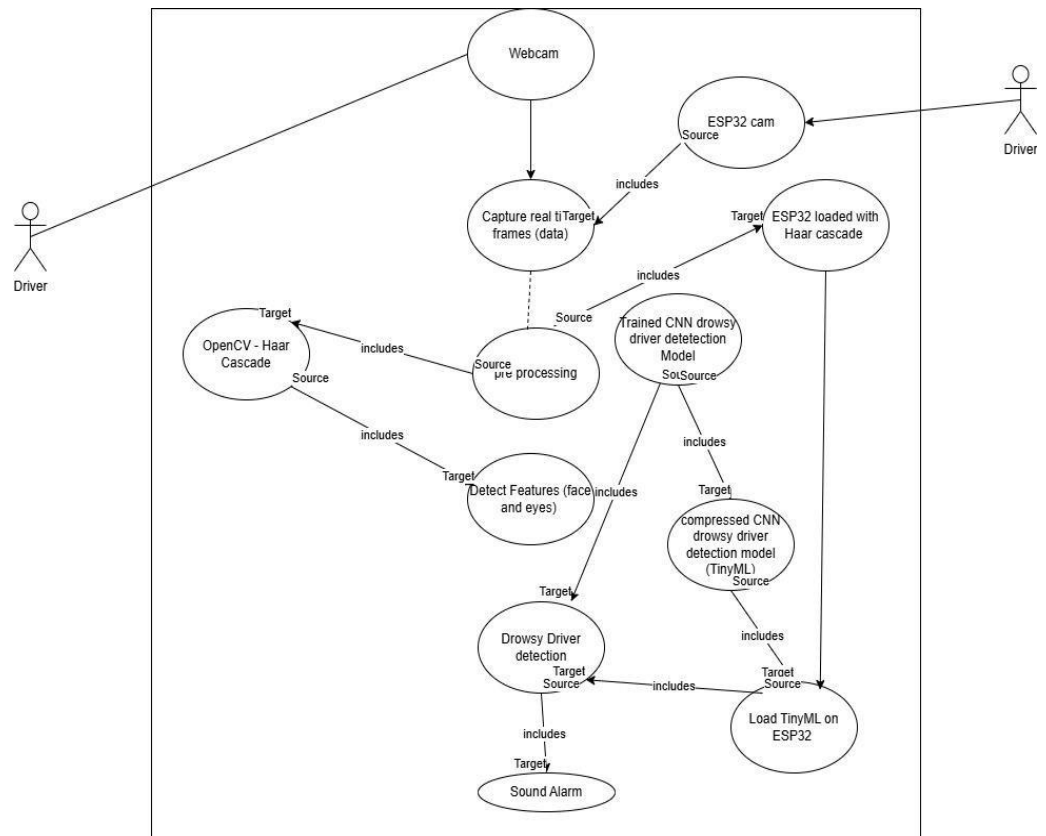
3.6.3.2 Non-functional Requirements

- a. Algorithmically simple and easy to implement- Rely on off-the-shelf solutions for most stages, based on the popularity and success of the associated algorithms (e.g., Support Vector Machine classifier)
- b. Easily portable to different platforms – The system must run on a ESP32 camera mounted on the vehicle's dashboard. Ideally, it should be easily portable to other mobile devices of comparable size and computational capabilities.
- c. Computationally non-intensive- Since (near) real-time performance is required, algorithms must be optimized to ensure continuous monitoring of driver's state without excessive burdening of the device's main processor. As a side benefit, battery consumption is reduced as well
- d. Accuracy – One of the main challenges of designing such a system is related to the fact that false positives will annoy the driver and reduce their willingness to use the system due to excessive false alarms, whereas false negatives can have literally catastrophic consequences and defeat the purpose of the entire system.
- e. Robustness- The system must be tolerant to modest amounts of lighting variations, relative camera motion (e.g. due to poor road conditions), changes to the driver's visual appearance (even in the course of a session, e.g., by wearing/removing a hat or sunglasses), camera resolution and frame rates, and different computational capabilities of the device's processors, Chumbar, S. (2023, September 24).

3.6.4 System Design

Overview of the Drowsy Driving Detection System/System Architecture

3.6.4.1 Use Case Diagram



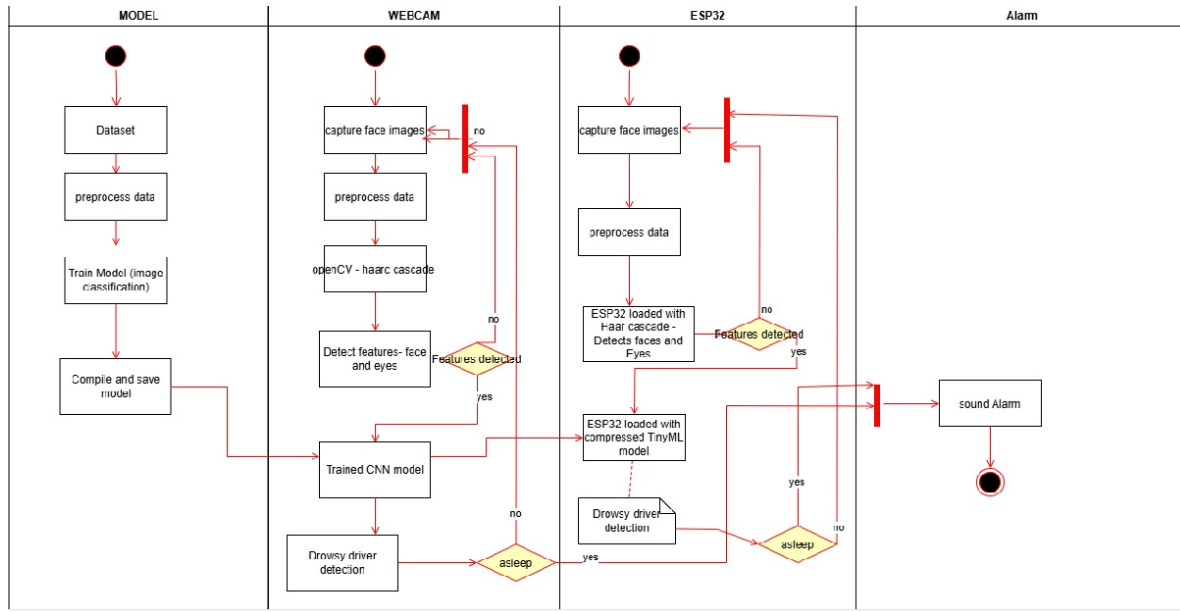
This use case diagram has two processes taking place.

The first one is where the driver used a webcam to capture the frames(images)that are used for detecting faces and eyes. Using haar cascade and a trained model the system detects and predicts whether the driver is sleeping or not. When the Driver sleeps the alarm rings.

The second one is where the driver used an ESP32 microcontroller witch has a camera that is mounted in a vehicle to capture frames. The ESP32 is loaded with haar cascade that is used for detecting faces and eyes. It is also loaded with an alarm and a compressed CNN model using TinyML witch will then determine whether the driver is sleeping or not causing the alarm to ring.

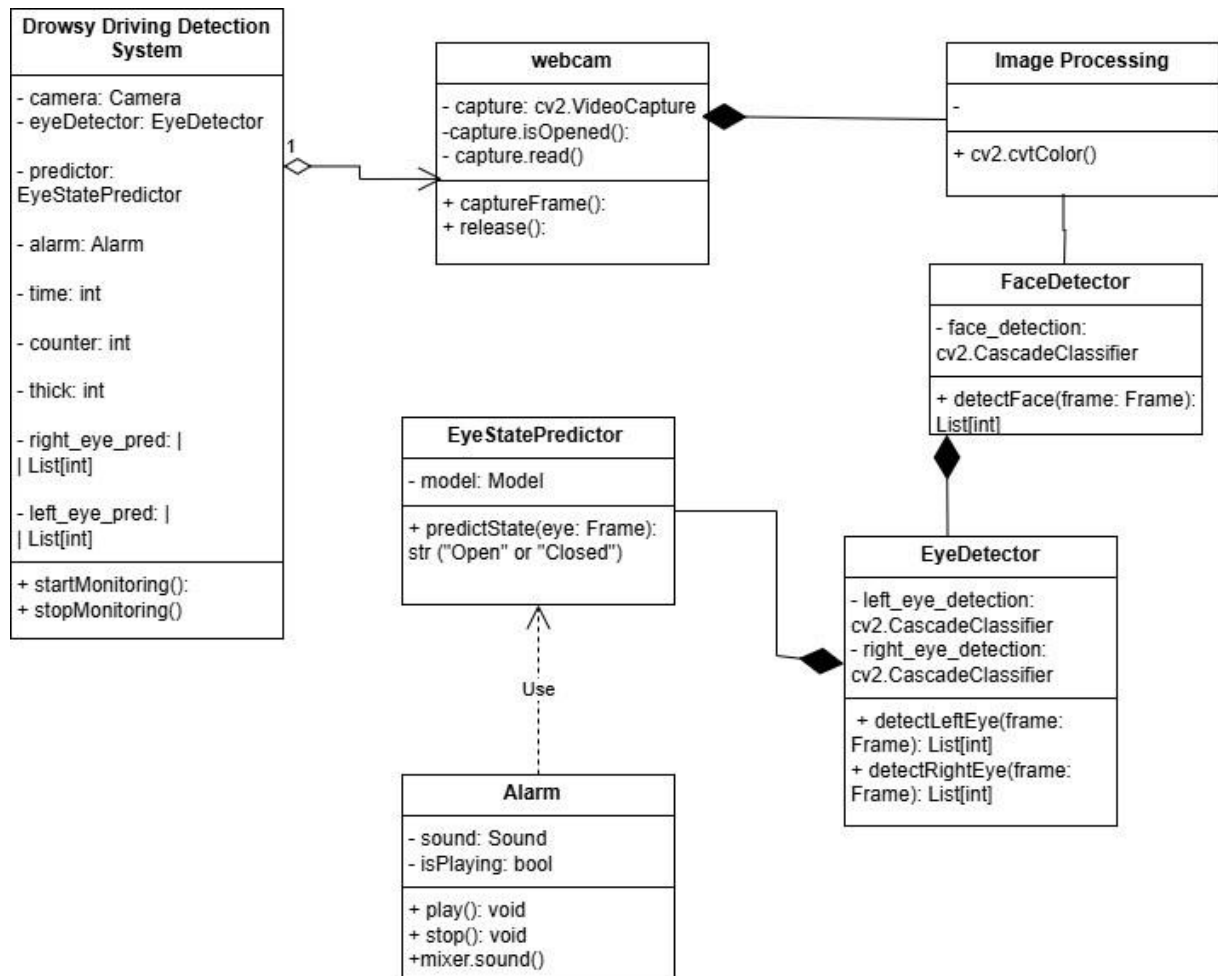
3.6.4.2 Activity Diagram

Figure below depicts an activity diagram illustrating the activities involved in the Drowsy driver detection system:



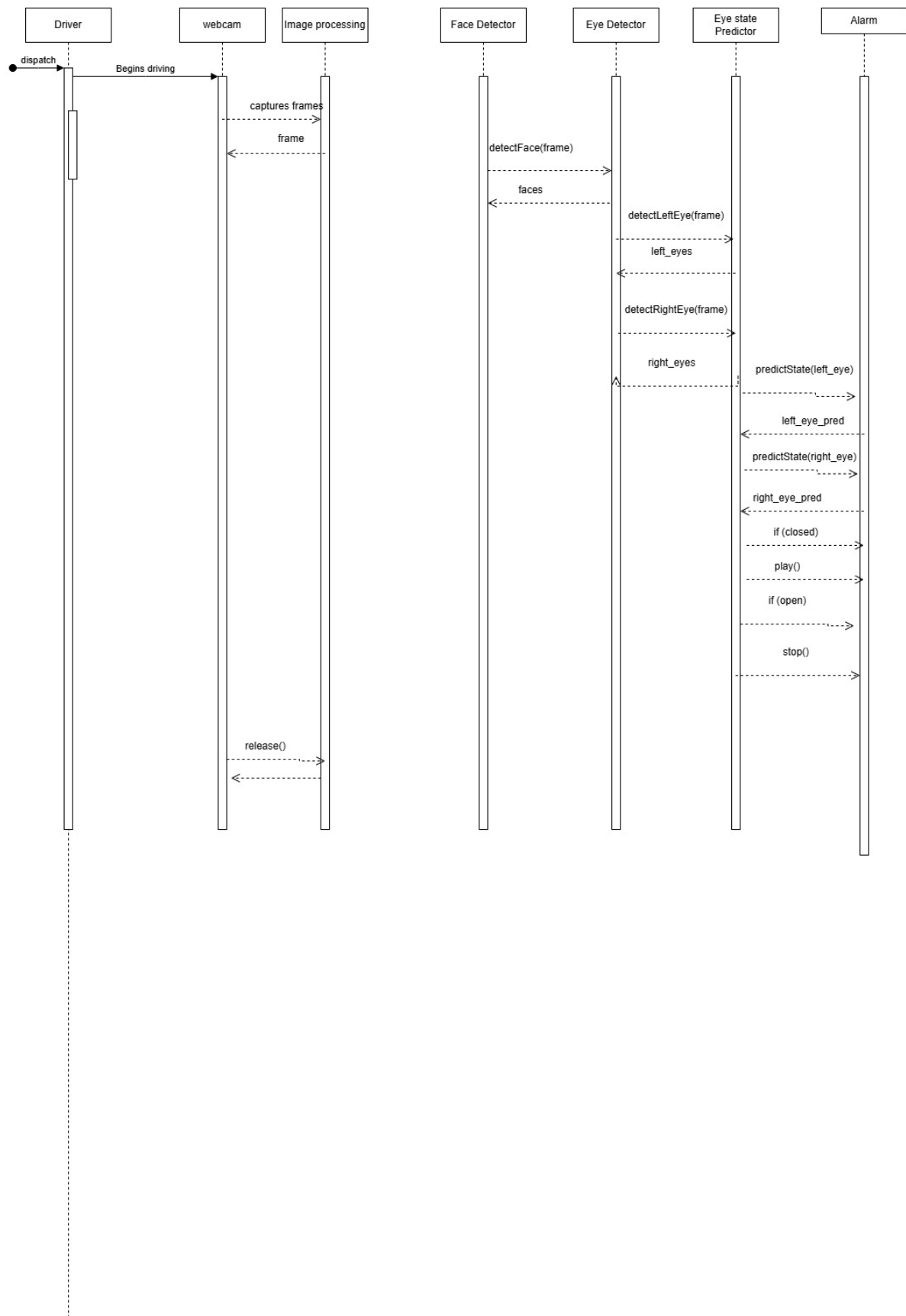
3.6.4.3 Class Diagram

Below is a class diagram that annotates how and in what order the various objects work together.



3.6.4.4 Sequence Diagram

Below is a sequence diagram that annotates how and in what order the various objects work together.



CHAPTER 4: IMPEMENTATION

4.1 Approach to Create Drowsiness detection system:

In this project, for collecting images from webcam we will be using OpenCV and feed these images to our Deep learning model which will classify that the person's eyes is 'Open' or 'Closed'. So following steps were taken:

- a) Take image input from the camera
- b) Detect face and eyes in the image.
- c) Create a Region of Interest (ROI), for both detected face and eyes.
- d) Feed this to our classifier(model), which will categorize whether eyes are open or closed.
- e) At last, calculate the time to check if the person is drowsy or not.

4.1.1 Download Driver Drowsiness Dataset

Download the dataset and set a path to it.

4.1.2 Create a model.

- a) After Setting the path, preprocess the images and perform operations so that it will be ready for training.
- b) After preprocessing of images, create the model which will classify whether a person's eyes are open or closed. 37
- c. Train the model and save the final weights and model architecture file as "models/my_model.keras".

Create The model.

Used keras to build models using Convolutional Neural Networks (CNN). CNN is a type of deep neural network which performs very accurately and efficiently for image classification. Convolutional Neural Network consists of input layer, output layer, and hidden layers. So, this operation is performed on these hidden layers using a filter that performs 2D matrix multiplication on the layers.

The CNN model architecture created consist of following layers:

Convolutional layer; 64 nodes, kernel size 3

Convolutional layer; 64 nodes, kernel size 3

Convolutional layer; 64 nodes, kernel size 3

Fully connected layer; 128 nodes

The output layer is also a fully connected layer with 2 nodes. In all the layers, use the activation function Relu except the output layer. In the output layer, used Softmax as an activation function.

Compile and save Model.

Check if model is loaded.

4.1.3 Main file 'Drowsy driver detection system'.

Project Prerequisites

You should have a webcam on your system through which images will be captured. You need to have installed the latest version of python or version (3.6 and above recommended). Now to run this project you are required to install:

a. OpenCV: to install run following command on cmd: pip install OpenCV-python (for webcam and for detection of face and eyes)

b. TensorFlow - pip install tensorflow (keras uses TensorFlow as backend).

38

c. Keras - pip install keras (to build our model)

d. Pygame - pip install pygame (to play alert sound).

1. Importing all the needed libraries:

2. Setting an alarm sound file, and setting a path of haar cascade files to detect face, detect left eye, and detect right eye.

3. Load model, and using OpenCV access the webcam that will capture each frame.

4. Initializing variables

5. This is the main logic of code.

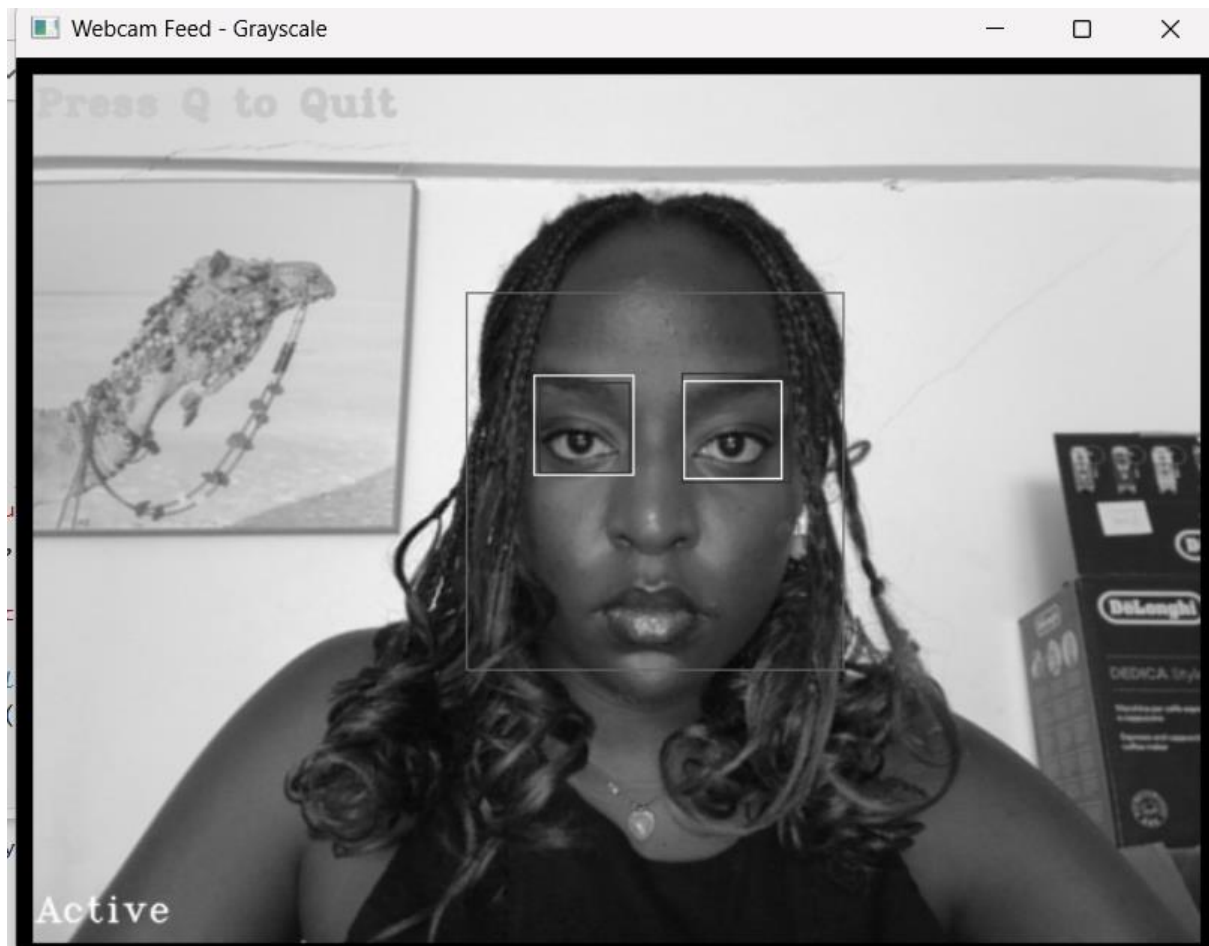
a. Perform detection (this will return x,y coordinates , height , width of the boundary boxes object)

b. Iterating over faces and drawing boundary boxes for each face.

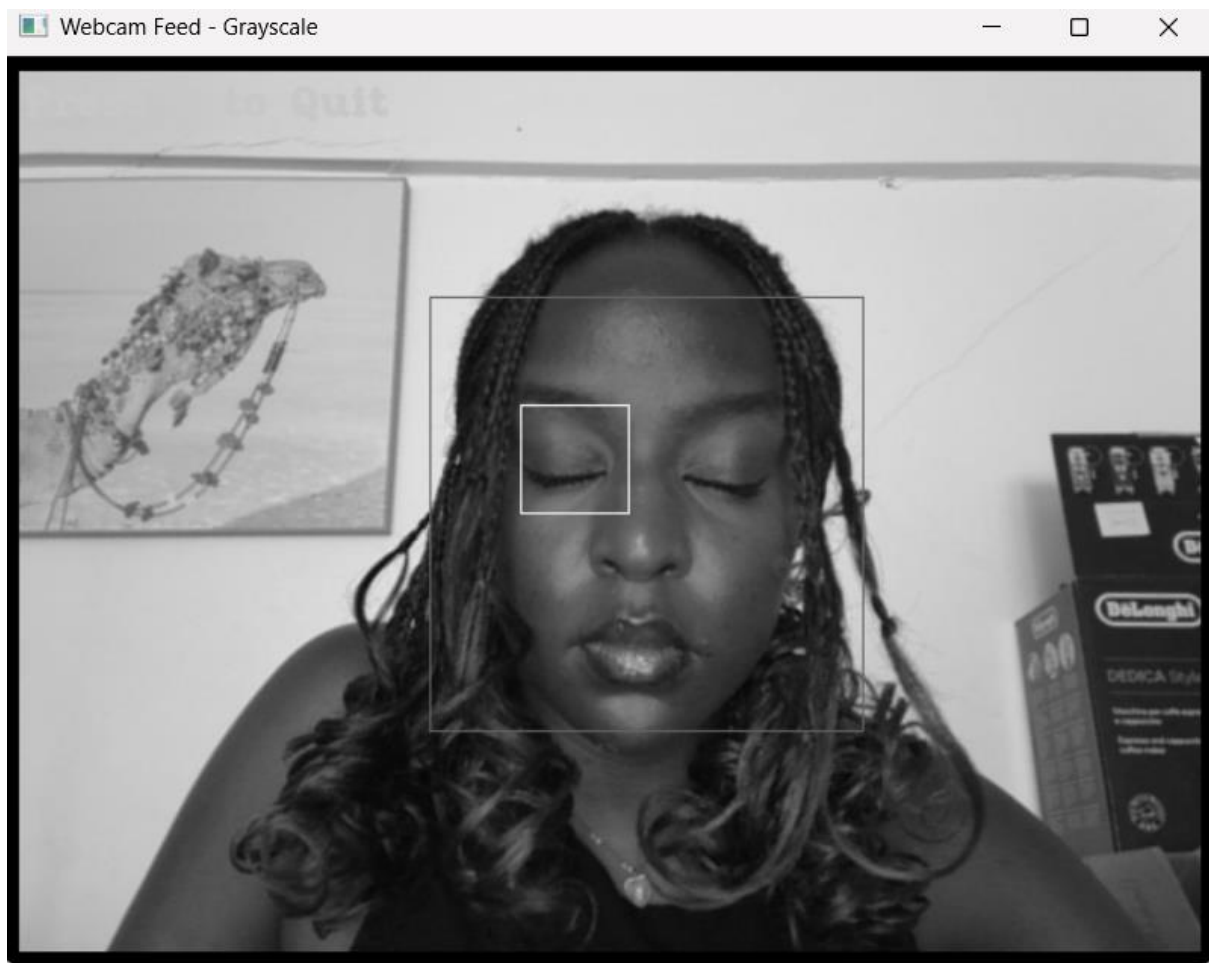
c. Iterate over left and right eye.

6. In this code check that, if the person's left eye and right eye are closed, time will increase and if time increases more than 4 the alert sound will start, and if both eyes are open the time decreases and sound stops after some time.

7. Output.



When eyes are open:



When eyes are closed (causing alarm to ring):

CHAPTER 5: CONCLUSION

The Drowsy Driver Detection System is an innovative solution designed to enhance road safety by monitoring a driver's eye state and providing timely alerts in case of drowsiness. By leveraging computer vision and machine learning, this system can effectively detect when a

driver's eyes are closed and sound an alarm to prevent potential accidents caused by drowsiness. It would have been even better to apply the TinyML and use ESP32 camera. This can be even better for the drivers as they would have been able to install the camera into their vehicles.

Recommendations

1. Regular Updates and Maintenance: Ensure the software and pre-trained models are regularly updated to incorporate the latest advancements in machine learning and computer vision. This will help improve the accuracy and reliability of the system.
2. Optimized Hardware: Use a high-quality webcam and a computer with sufficient processing power to ensure real-time video processing. This will enhance the system's performance and response time.
3. Improved Algorithms: Continuously improve the eye state detection algorithms to minimize false positives and negatives. Implementing more advanced neural networks or ensemble methods could enhance detection accuracy.
4. User Training: Educate users on the proper use of the system and its limitations. Drivers should be aware that this system is an aid and not a replacement for attentive driving.
5. Environmental Conditions: Test the system under various lighting conditions and environmental scenarios to ensure robustness. Implement adaptive techniques to handle changes in lighting and driver positioning.
6. Integration with Vehicle Systems: Explore the possibility of integrating the drowsy driver detection system with the vehicle's onboard systems. This could enable automatic responses, such as reducing speed or alerting emergency services if the driver does not respond to the alarm.
7. User Feedback: Collect feedback from users to understand their experiences and

identify areas for improvement. This feedback can be invaluable in refining the system to better meet user needs.

8. Legal and Ethical Considerations: Ensure that the implementation of the system adheres to all relevant legal and ethical standards, including data privacy and security.

Clear communication about data usage and obtaining necessary consents is essential.

By following these recommendations, the Drowsy Driver Detection System can be further enhanced to provide even greater safety benefits, making roads safer for everyone.

References

Advanced driver assistance. (n.d.). IIHS-HLDI Crash Testing and Highway Safety. <https://www.iihs.org/topics/advanced-driver-assistance>

Rammohan, A., Chavhan, S., Chidambaram, R. K., Manisaran, N., & Kumar, K. V. P. (2022). Automotive Collision Avoidance System: a review. *Studies in Systems, Decision and Control*, 1–19. https://doi.org/10.1007/978-3-030-94102-4_1

Bajaj, J. S., Kumar, N., Kaushal, R. K., Gururaj, H. L., Flammini, F., & Natarajan, R. (2023, January 23). System and method for driver drowsiness detection using behavioral and sensor-based physiological measures. *MDPI*. <https://www.mdpi.com/1424-8220/23/3/1292>

Albadawi, Y., Takturi, M., & Awad, M. (2022). A review of recent developments in driver drowsiness detection systems. *Sensors*, 22(5), 2069. <https://doi.org/10.3390/s22052069>

Zhao Z et al (2020) Driver Fatigue Detection Based on Convolutional Neural Networks Using EM-CNN Computational Intelligence and Neuroscience vol 1 pp 1 11.

Kamarudi N et al (2019) Implementation of Haar Cascade Classifier and Eye Aspect Ratio For Driver Drowsiness Detection Using Raspberry Pi Universal Journal of Electrical and Electronic Engineering 6(5B) pp 67- 75.

Alajlan N.N and Ibrahim D. M (2023) DDD TinyML: A TinyML-Based Driver Drowsiness Detection Model Using Deep Learning Sensors Volume 23 Issue 12.

Agnihotri, N. (2023, May 17). What is TinyML? Engineers Garage. <https://www.engineersgarage.com/what-is-tinyml/>

Transfer learning and fine-tuning. (n.d.). TensorFlow. https://www.tensorflow.org/tutorials/images/transfer_learning#train_the_model

Donges, N. (2024, August 15). What is transfer learning? Exploring the popular deep learning approach. Built In. <https://builtin.com/data-science/transfer-learning>

Adaloglou, N. (2021, January 21). Best deep CNN architectures and their principles: from AlexNet to EfficientNet | AI Summer. AI Summer. <https://theaisummer.com/cnn-architectures/>

8.1. Deep Convolutional Neural Networks (AlexNet) — Dive into Deep Learning 1.0.3 documentation. (n.d.). https://d2l.ai/chapter_convolutional-modern/alexnet.html

Kamarudi N et al (2019) Implementation of Haar Cascade Classifier and Eye Aspect Ratio For Driver Drowsiness Detection Using Raspberry Pi Universal Journal of Electrical and Electronic Engineering 6(5B) pp 67- 75.

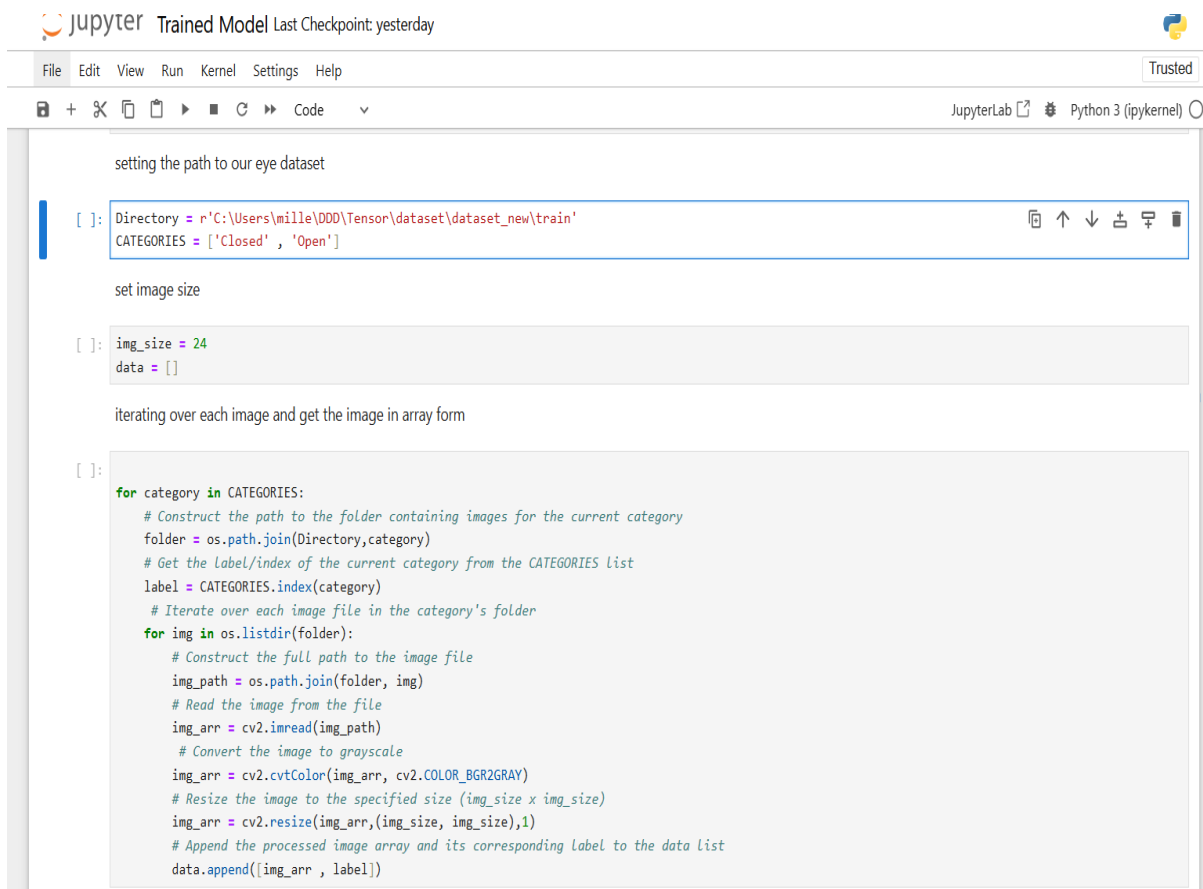
Costa, F. (2022, January 5). TinyML models behind the scenes explained | Marionete. Medium. <https://medium.com/marionete/tinyml-models-whats-happening-behind-the-scenes-5e61d1555be917>. [Convolutional Neural Network \(CNN\) in Machine Learning - GeeksforGeeks](#)

Hotz, N. (2024, December 9). What is CRISP DM? Data Science PM. <https://www.datascience-pm.com/crisp-dm-2/>

Chumbar, S. (2023, September 24). The CRISP-DM Process: A Comprehensive guide - Shawn Chumbar - medium. Medium. <https://medium.com/@shawn.chumbar/the-crisp-dm-process-a-comprehensive-guide-4d893aecb151#:~:text=Why%20Use%20CRISP->

[DM%3F%20Versatility%3A%20It%E2%80%99s%20industry-agnostic%2C%20meaning%20it,Given%20its%20cyclical%20nature%2C%20it%20encourages%20continuous%20improvement.](#)

Appendix A: Sample Code.



The screenshot shows a JupyterLab environment with a 'Trained Model' checkpoint from yesterday. The interface includes a menu bar (File, Edit, View, Run, Kernel, Settings, Help) and a toolbar with icons for saving, opening, and running code. The code is written in Python 3 (ipykernel) and is organized into three cells. The first cell sets the directory path and categories. The second cell sets the image size and initializes a data list. The third cell is a loop that iterates over each category and image, processing them into a grayscale array and appending them to the data list.

```
setting the path to our eye dataset

[ ]: Directory = r'C:\Users\mille\DDD\Tensor\dataset\dataset_new\train'
    CATEGORIES = ['Closed', 'Open']

set image size

[ ]: img_size = 24
    data = []

iterating over each image and get the image in array form

[ ]: for category in CATEGORIES:
    # Construct the path to the folder containing images for the current category
    folder = os.path.join(Directory, category)
    # Get the label/index of the current category from the CATEGORIES list
    label = CATEGORIES.index(category)
    # Iterate over each image file in the category's folder
    for img in os.listdir(folder):
        # Construct the full path to the image file
        img_path = os.path.join(folder, img)
        # Read the image from the file
        img_arr = cv2.imread(img_path)
        # Convert the image to grayscale
        img_arr = cv2.cvtColor(img_arr, cv2.COLOR_BGR2GRAY)
        # Resize the image to the specified size (img_size x img_size)
        img_arr = cv2.resize(img_arr, (img_size, img_size), 1)
        # Append the processed image array and its corresponding label to the data list
        data.append([img_arr, label])
```

```
5]: # see the length of data:  
len(data)
```

```
5]: 1234
```

shuffle the data to get random images of open eyes and closed eyes:

```
6]: random.shuffle(data)
```

```
7]: # dividing features and label for training the model:  
X = []  
Y = []  
  
for features, label in data:  
    X.append(features)  
    Y.append(label)
```

```
8]: #covert them into array:  
X = np.array(X)  
Y = np.array(Y)
```

```
9]: # save the data into system:  
pickle.dump(X , open('X.pkl' , 'wb'))  
pickle.dump(Y , open('Y.pkl' , 'wb'))
```

```
0]: # normalize the image array:  
X = X/255
```

```
[11]: X
```

```
[11]: array([[0.85882353, 0.85098039, 0.81960784, ..., 0.51372549,
          0.50980392, 0.49019608],
          [0.85882353, 0.81568627, 0.80392157, ..., 0.49411765,
          0.49019608, 0.50980392],
          [0.85882353, 0.74901961, 0.72941176, ..., 0.52156863,
          0.49411765, 0.46666667],
          ...,
          [0.63921569, 0.64313725, 0.58039216, ..., 0.44705882,
          0.56862745, 0.60392157],
          [0.65098039, 0.67843137, 0.66666667, ..., 0.48627451,
          0.58431373, 0.60392157],
          [0.65490196, 0.67058824, 0.69411765, ..., 0.50980392,
          0.57254902, 0.59607843]],

          [[0.38431373, 0.51372549, 0.5372549 , ..., 0.73333333,
          0.73333333, 0.73333333],
          [0.44313725, 0.51764706, 0.53333333, ..., 0.7254902 ,
          0.72941176, 0.7372549 ],
          [0.50196078, 0.56078431, 0.55294118, ..., 0.72941176,
          0.70196078, 0.7254902 ],
          ...,
          [0.56862745, 0.60392157, 0.63137255, ..., 0.58823529,
          0.58431373, 0.59215686],
          [0.56470588, 0.61176471, 0.65490196, ..., 0.60392157,
          0.58039216, 0.60392157]])
```

Reshape

reshape the X array to (24,24,1)

```
2]: img_rows,img_cols = 24,24
   # resizes the array X to ensure it has the correct dimensions for input into a CNN.
   X = X.reshape(X.shape[0],img_rows,img_cols,1)
   # new shape
   X.shape
```

```
2]: (1234, 24, 24, 1)
```



```
[14]: # Create a directory named 'models' if it doesn't already exist
os.makedirs('models', exist_ok=True)

# Initialize a Sequential model, which is a linear stack of layers
model = Sequential()

# Define the input layer
# Add an input layer specifying the shape of the input data, derived from the shape of X
model.add(Input(shape=X.shape[1:]))

# Add the convolutional and pooling layers
# Add a 2D convolutional layer with 64 filters, a 3x3 kernel size, and ReLU activation
model.add(Conv2D(64, (3,3), activation='relu'))
# Add a max pooling layer with a 1x1 pool size, which does not reduce the spatial dimensions
model.add(MaxPooling2D((1,1)))

model.add(Conv2D(64, (3,3), activation='relu'))
model.add(MaxPooling2D((1,1)))

model.add(Conv2D(64, (3,3), activation='relu'))
model.add(MaxPooling2D((1,1)))

# Flatten the output and add dense layers
# Flatten the output from the convolutional layers to create a 1D feature vector
model.add(Flatten())
# Add a fully connected (dense) layer with 128 units and ReLU activation
model.add(Dense(128, activation='relu'))

# Add the output layer
# Add an output layer with 2 units (for binary classification) and softmax activation
model.add(Dense(2, activation='softmax'))
```

Compile and saveModel:

compile model that we have created

```
15]: model.compile(optimizer = 'adam' , loss = 'sparse_categorical_crossentropy' , metrics = ['accuracy'])
```

fit X, Y to the model to see accuracy of model:

```
16]: # trains the Keras model on the provided dataset.
model.fit(X, Y, epochs = 5 , validation_split = 0.1 , batch_size = 32)

Epoch 1/5
35/35 ————— 6s 75ms/step - accuracy: 0.6813 - loss: 0.5967 - val_accuracy: 0.8871 - val_loss: 0.3378
Epoch 2/5
35/35 ————— 2s 59ms/step - accuracy: 0.9248 - loss: 0.2027 - val_accuracy: 0.8790 - val_loss: 0.3359
Epoch 3/5
35/35 ————— 3s 79ms/step - accuracy: 0.9267 - loss: 0.1910 - val_accuracy: 0.9194 - val_loss: 0.2482
Epoch 4/5
35/35 ————— 3s 76ms/step - accuracy: 0.9615 - loss: 0.1033 - val_accuracy: 0.9677 - val_loss: 0.1445
Epoch 5/5
35/35 ————— 3s 75ms/step - accuracy: 0.9653 - loss: 0.0831 - val_accuracy: 0.8952 - val_loss: 0.2603
16]: <keras.src.callbacks.history.History at 0x1cc948730d0>
```

save model and architecture to single file

```
17]: model.save('models/my_model.keras')
```

Check if model is loaded:

```
8]: file_path = 'models/my_model.keras'

# Check if the file exists
if os.path.exists(file_path):
    # Load the model
    model = load_model(file_path)
    print("Model loaded successfully.")
else:
    print(f"File not found: {file_path}")

Model loaded successfully.
```

1. Importing all the needed libraries:

```
[ ]: import cv2
import os
from keras.models import load_model
import numpy as np
from pygame import mixer
import time
import tensorflow as tf
```

2. Setting an alarm sound file, and setting a path of haar cascade files to detect face, detect left eye, and detect right eye.

```
[ ]: # this is used to get beep sound (when person closes his eyes for more them 10sec)
mixer.init()
alarm_sound = mixer.Sound('alarm.wav')

[ ]: # this xml files are used to detect face , left eye , and right eye of a person.
face_detection = cv2.CascadeClassifier('haar cascade files\haarcascade_frontalface_alt.xml')
left_eye_detection = cv2.CascadeClassifier('haar cascade files\haarcascade_lefteye_2splits.xml')
right_eye_detection = cv2.CascadeClassifier('haar cascade files\haarcascade_righteye_2splits.xml')

# Define the paths for the XML files
face_detection_path = 'haar cascade files/haarcascade_frontalface_alt.xml'
left_eye_detection_path = 'haar cascade files/haarcascade_lefteye_2splits.xml'
right_eye_detection_path = 'haar cascade files/haarcascade_righteye_2splits.xml'

# Load the cascade classifiers
face_detection = cv2.CascadeClassifier(face_detection_path)
left_eye_detection = cv2.CascadeClassifier(left_eye_detection_path)
right_eye_detection = cv2.CascadeClassifier(right_eye_detection_path)

# Check if the cascades are loaded properly
if face_detection.empty():
    print(f"Error loading face detection cascade from {face_detection_path}")
else:
    print("Face detection cascade loaded successfully.")

if left_eye_detection.empty():
    print(f"Error loading left eye detection cascade from {left_eye_detection_path}")
else:
    print("Left eye detection cascade loaded successfully.")

if right_eye_detection.empty():
    print(f"Error loading right eye detection cascade from {right_eye_detection_path}")

    if right_eye_detection.empty():
        print(f"Error loading right eye detection cascade from {right_eye_detection_path}")
    else:
        print("Right eye detection cascade loaded successfully.")

Face detection cascade loaded successfully.
Left eye detection cascade loaded successfully.
Right eye detection cascade loaded successfully.

[ ]: #Define labels for the model's predictions.
label = ['Close', 'Open']
```

3. Load model, and using OpenCV access the webcam that will capture each frame.

```
[5]: #Load the pre-trained Keras model for eye state classification.
model = load_model('models/my_model.keras')

# Load the pre-trained Keras model
model_path = 'models/my_model.keras'
try:
    model = load_model(model_path)
    print("Model loaded successfully.")
except Exception as e:
    print(f"Error loading the model from {model_path}: {e}")

# Check if the model is loaded correctly
if isinstance(model, tf.keras.Model):
    print("Model is a valid Keras Model.")
    # Print the model summary
    model.summary()
else:
    print("Model loading failed.")

Model loaded successfully.
Model is a valid Keras Model.
```

Model: "sequential"

| Layer (type) | Output Shape | Param # |
|--------------------------------|--------------------|-----------|
| conv2d (Conv2D) | (None, 22, 22, 64) | 640 |
| max_pooling2d (MaxPooling2D) | (None, 22, 22, 64) | 0 |
| conv2d_1 (Conv2D) | (None, 20, 20, 64) | 36,928 |
| max_pooling2d_1 (MaxPooling2D) | (None, 20, 20, 64) | 0 |
| conv2d_2 (Conv2D) | (None, 18, 18, 64) | 36,928 |
| max_pooling2d_2 (MaxPooling2D) | (None, 18, 18, 64) | 0 |
| flatten (Flatten) | (None, 20736) | 0 |
| dense (Dense) | (None, 128) | 2,654,336 |
| dense_1 (Dense) | (None, 2) | 258 |

Total params: 8,187,272 (31.23 MB)
Trainable params: 2,729,090 (10.41 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 5,458,182 (20.82 MB)

check if the webcam is opened correctly

```
] : # Initialize the webcam
capture = cv2.VideoCapture(0)
# Check if the webcam is opened
if capture.isOpened():
    print("Webcam is open and ready to use.")
else:
    raise IOError("Cannot open webcam")
```

#Set font for displaying text on the frame.

```
font = cv2.FONT_HERSHEY_COMPLEX_SMALL
```

Webcam is open and ready to use.

4. Initializing variables

declaring variables for counting and other purposes.

```
] : counter = 0
time = 0
thick = 2
right_eye_pred=[99]
left_eye_pred=[99]
```

5. This is the main logic of code.

```

: #Begin an infinite loop to continuously capture frames from the webcam.
while(True):
    # Capture a frame from the webcam.
    ret, frame = capture.read()
    # Check if frame is captured successfully
    if not ret:
        print("Failed to grab frame")
        break

    # Convert the frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Check if the conversion is successful by verifying the number of channels
    if len(gray.shape) == 2:
        print("Frame successfully converted to grayscale.")
    else:
        print("Failed to convert frame to grayscale.")

    # Perform face and eye detection
    faces = face_detection.detectMultiScale(gray, minNeighbors=5, scaleFactor=1.1, minSize=(25, 25))
    left_eye = left_eye_detection.detectMultiScale(gray)
    right_eye = right_eye_detection.detectMultiScale(gray)

    # Check if faces are detected
    if len(faces) > 0:
        print("Face detected!")
        # Draw rectangles around detected faces
        for (x, y, w, h) in faces:
            cv2.rectangle(gray, (x, y), (x + w, y + h), (100, 100, 100), 1)
    else:
        print("No face detected!")

```

```

# Check if left eyes are detected
if len(left_eye) > 0:
    print("Left eye detected!")
    # Draw rectangles around detected left eyes
    for (x, y, w, h) in left_eye:
        cv2.rectangle(gray, (x, y), (x + w, y + h), (50, 50, 255), 1)
else:
    print("No left eye detected!")

# Check if right eyes are detected
if len(right_eye) > 0:
    print("Right eye detected!")
    # Draw rectangles around detected right eyes
    for (x, y, w, h) in right_eye:
        cv2.rectangle(gray, (x, y), (x + w, y + h), (255, 50, 50), 1)
else:
    print("No right eye detected!")

# Get the height and width of the frame.
height,width = gray.shape[:2]

```

```

for (x,y,w,h) in right_eye:
    #pull out the right eye image from the frame:
    right_one=gray[y:y+h,x:x+w]
    print("Right eye region detected:", (x, y, w, h))

    counter += 1
    # right_one = cv2.cvtColor(right_one,cv2.COLOR_BGR2GRAY)
    right_one = cv2.resize(right_one,(24,24))
    right_one = right_one/255
    right_one = right_one.reshape(24,24,-1)
    right_one = np.expand_dims(right_one,axis=0)

    # Predict the state of the right eye
    right_eye_pred = model.predict(right_one)
    right_eye_pred_class = np.argmax(right_eye_pred, axis=1)
    print("Prediction results:", right_eye_pred)
    print("Predicted class:", right_eye_pred_class)

    if right_eye_pred_class[0] == 1:
        label = 'Open'
    elif right_eye_pred_class[0] == 0:
        label = 'Closed'
        break

```

```

3]: for (x,y,w,h) in left_eye:
    #pull out the left eye image from the frame:
    left_one=gray[y:y+h,x:x+w]
    print("Left eye region detected:", (x, y, w, h))

    counter += 1
    #left_one = cv2.cvtColor(left_one,cv2.COLOR_BGR2GRAY)
    left_one = cv2.resize(left_one,(24,24))
    left_one = left_one/255
    left_one = left_one.reshape(24,24,-1)
    left_one = np.expand_dims(left_one,axis=0)

    # Predict the state of the left eye
    left_eye_pred = model.predict(left_one)
    left_eye_pred_class = np.argmax(left_eye_pred, axis=1)
    print("Prediction results:", left_eye_pred)
    print("Predicted class:", left_eye_pred_class)

    if left_eye_pred_class[0] == 1:
        label = 'Open'
    elif left_eye_pred_class[0] == 0:
        label = 'Closed'
        break

```

6. In this code check that, if the person's left eye and right eye are closed, time will increase and if time increases more than 4 the alert sound will start, and if both eyes are open the time decreases and sound stops after some time.

```
if right_eye_pred_class[0] == 0 and left_eye_pred_class[0] == 0:
    time += 1
    cv2.putText(gray, "Inactive", (10, height-20), font, 1, (255, 255, 255), 1, cv2.LINE_AA)
    #alarm_sound = mixer.Sound('alarm.wav')
    if (time > 4):
        alarm_sound.play()

if right_eye_pred_class[0] == 1 and left_eye_pred_class[0] == 1:

    alarm_sound.stop()
    time = 0
    cv2.putText(gray, "Active", (10, height-20), font, 1, (255, 255, 255), 1, cv2.LINE_AA)
```

```
if(thick < 16):
    thick = thick+2
else:
    thick=thick-2
    if(thick<2):
        thick=2
```

```
# Optional: Add some text to the frame
cv2.putText(gray, 'Press Q to Quit', (10, 30), font, 1, (200, 0, 200), 2, cv2.LINE_AA)
cv2.rectangle(gray, (0, 0), (width, height), (0, 0, 255), thick)
# Display the grayscale frame
cv2.imshow('Webcam Feed - Grayscale', gray)

# Wait for 'q' key to stop the loop
if cv2.waitKey(1) & 0xFF == ord('q'):
    break
```



```
Frame successfully converted to grayscale.  
Face detected!  
Left eye detected!  
Right eye detected!  
Right eye region detected: (310, 309, 35, 35)  
1/1 ————— 0s 83ms/step  
Prediction results: [[9.9902284e-01 9.7719638e-04]]  
Predicted class: [0]  
Left eye region detected: (362, 314, 34, 34)  
1/1 ————— 0s 20ms/step  
Prediction results: [[9.9987006e-01 1.2986973e-04]]  
Predicted class: [0]  
Frame successfully converted to grayscale.  
Face detected!  
Left eye detected!  
Right eye detected!  
Right eye region detected: (309, 308, 37, 37)  
1/1 ————— 0s 16ms/step
```

```
[ ]: capture.release()  
    cv2.destroyAllWindows()
```

```
[ ]:
```

Appendix B: User Manual

Drowsy Driver Detection System - User Manual

Introduction

The Drowsy Driver Detection System is designed to monitor a driver's eye state and alert them when signs of drowsiness are detected. This system uses a webcam to capture video frames, detects the driver's face and eyes, and then uses a machine learning model to predict whether the eyes are open or closed. If the system detects that the driver's eyes have been closed for a prolonged period, it sounds an alarm to alert the driver.

System Requirements

Hardware

- Webcam
- Computer with sufficient processing power to run real-time video processing

Software

- Python 3.x
- OpenCV library
- Numpy library
- Pre-trained eye state detection model
- Pygame library for playing the alarm sound

Installation

1. **Install Python:** Ensure Python is installed on your computer. You can download it from python.org.

2. **Install Required Libraries:** Open a terminal or command prompt and install the required libraries using pip:

```
pip install openCV-python NumPy pygame
```

3. **Pre-trained Model:** Ensure you have a pre-trained model for eye state detection. Place the model file in an accessible directory.
4. **Alarm Sound:** Ensure you have an alarm sound file (e.g., alarm.wav). Place the sound file in an accessible directory.

Setup and Running the System

1. **Setup the Script**
2. **Update the Script:** Replace the path_to_your_model with the actual path to your model file.
3. **Run the Script:**
 - Open a terminal or command prompt.
 - Navigate to the directory where drowsy_driver_detection.py is saved.
 - Run the script using Python:

Using the System

1. **Starting the System:**
 - When you run the script, the webcam feed will open in a new window.
 - The system will start processing the video feed to detect the left and right eye region and predict its state.
2. **Monitoring Eye State:**
 - The system will display "Open" or "Closed" based on the detected eye state.
 - If the driver's eyes are detected as closed the system will continuously monitor the driver's eye state. If the eyes remain closed for a period longer than the set threshold, an alarm will sound to alert the driver.
3. **Quitting the System:**
 - To quit the system, press the 'Q' key. This will stop the video capture and close all OpenCV windows.

Troubleshooting

- **Failed to Grab Frame:** Ensure the webcam is properly connected and not being used by another application.
- **Prediction Errors:** Verify that the model path is correct and the model file is compatible with the script.
- **Incorrect Eye Detection:** Ensure the eye detection coordinates are accurate. You may need to implement or integrate an eye detection algorithm to obtain these coordinates.
- **Alarm Not Sounding:** Check if the 'alarm.wav' file is in the correct directory and ensure Pygame is correctly installed.
- **Frame Not Saving:** Ensure the 'images' directory exists. If not, create it manually.

Safety Precautions

- **Alertness:** Always ensure you are alert and awake while driving. This system is an aid and should not be solely relied upon.
- **Environment:** Use the system in well-lit conditions to ensure accurate detection.

Conclusion

The Drowsy Driver Detection System is a simple yet effective way to monitor driver alertness using real-time eye state detection. Follow the setup instructions carefully to ensure the system operates correctly. If you encounter any issues, refer to the troubleshooting section or consult the relevant documentation for the libraries used. By following this user manual, you can effectively set up and use the system to promote safer driving practices.

