

Zagazig University

Faculty of Computers and Informatics Computer Science Department

**Project ID:** **GP\_2405**

**RouteEye**



Zagazig University

Faculty of Computers and Informatics Computer Science Department

**Project ID : GP\_2405**

**RouteEye**

**Detection and Alert System for Drivers Drowsiness Using Deep learning and IOT Technologies.**

**BY**

1. Youssef Ahmed Muhammad Khalil
2. Yousef Ibrahim Hassan Emara
3. Mariam Hamada Mohammed Al-hussieny
4. Mariam Ahmed Mohamed Said Bendary
5. Mariam Osama Shaker Elsayed
6. Nawal Abdelkhalek Mohamed Nasr Eldorf

**Under Supervision Of**

**Prof. Mohammed Abdel-Basset**

**2024**

**RouteEye: Detection and Alert System for Drivers Drowsiness Using Deep learning and IOT Technologies.**

Youssef Khalil, Yousef Emara, Mariam Al-hussieny, Mariam Ahmed, Mariam Osama, Nawal Abdelkhalek

**A B S T R A C T**

It is important to note that driver status is crucial for road safety since a large percentage of accidents are caused by drivers who are not paying attention or being too sleepy. With an increased number of fatalities associated with drowsy driving crashes, there is need for a solution that is accessible, cheap and yet able to function well. In this regard, we propose a mobile application as drowsiness detection system which can be used by different categories of drivers. This system works in real time utilizing deep learning algorithms and the Dlib algorithm in detecting droopy eyes and yawn lines on someone’s face. The major model utilized for such facial analysis is Convolutional Neural Network (CNN). For instance, it has achieved high accuracy levels at 98% in terms of driver drowsiness detection rate. The application is designed to notify drivers during several stages effectively. These include alarm sounds, hardware integration as well as periodic notifications. It continuously monitors and analyzes facial expressions. Thereby making the system contribute to improved road safety through early warning signs of drowsiness leading to dozing off while driving.

This project demonstrates the creation of a mobile app that utilizes Flutter for the sake of drowsy driver detection using facial recognition and alerting options. Our driver's facial pictures are taken by the application and are to be received by Flask-based server backend. The Dlib library should be able to find landmarks as well as crop human faces from photos on the server side in preparation for drowsiness forecasting with a specific model while also cropping lethargy prediction models out of those same pictures using masked-out shapes. When the application detects a sign of drowsiness, it goes through four alert mechanisms in order: the first one is to play warning sound, the second one is to flash light, the third one is use vibrating motor located in the driver’s seat together with sending a message about the driver’s condition to some chosen person.

**1. INTRODUCTION**

Every year the lives of approximately 1.19 million people are cut short as a result of a road traffic crash. Between 20 and 50 million more people suffer non-fatal injuries, with many incurring a disability. Road traffic injuries cause considerable economic losses to individuals, their families, and to nations as a whole. These losses arise from the cost of treatment as well as lost productivity for those killed or disabled by their injuries, and for family members who need to take time off work or school to care for the injured. Road traffic crashes cost most countries 3% of their gross domestic product [1][2]. Fatigue has been labelled as part of the “fatal five” for driving safety risks, alongside speeding, drugs/alcohol, failure to wear a seat belt and driver distraction. Research has also shown and strongly advertised that the effects of drowsiness when driving is similar to that of driving above the legal alcohol limit [2], [3]. Hence, the monitoring of fatigued driving is required to make driving safer and has emerged as a key priority to reduce road related fatalities and serious injuries.

Many reports have quantified the impact of driver fatigue and drowsiness on public safety. According to the European Sleep Research Society, a significant number of driving accidents in various European countries are caused by drowsiness. For instance, in the UK, around 25–30% of accidents are associated with drivers feeling sleepy [3]. Similarly, in the Netherlands, approximately 35% of drivers have admitted to falling asleep while driving, whereas in Spain, the percentage is even higher at 70%. Shockingly [4], in 2011, there were 3,970 fatal accidents on the roads in France, with 85% being caused by drivers who were tired. Similarly, in Germany, 25% of all fatal road traffic accidents were the result of drowsy driving [5]. Therefore, car safety technology must include driver drowsiness detection for preventing accidents. Drowsy drivers can cause accidents. One way to decrease the number of road traffic accidents is through prompt recognition of driver fatigue by providing alarm for alert.

Many existing review papers have thoroughly covered technical aspects such as signal acquisition, feature extraction and detection of driver drowsiness. The first approach uses the measurement of physiological activities of the human body such as brainwave (using electroencephalogram - EEG) [6], [7], heart rate (using electrocardiogram – ECG) [8] and electric signals from muscle cells (using electromyogram – EMG) [9]. The measured signal is usually reliable enough to detect drowsiness because its correlation with driver alertness is relatively strong. However, this technique has some limitations due to the necessity to apply tools to the user's body which is somewhat intrusive. The second approach uses driver's fatigue-related behavior status that can also be detected on the basis of vehicle behaviors, such as steering movements, accelerations and braking, a speed of vehicles, and lane crossings [8:12]. Unfortunately, this approach can only be used for very limited conditions, because it could be a sudden movement that occurs due to road conditions, not because of the sleepy condition of the driver [8],[12]. A third approach to estimate driver drowsiness is by employing a set of cameras and image processing technique to detect changes in driver behavior, such as movement of eyes, yawning [13],[14], head movement [15], and facial expression [16:20]. The appearance and behavior of a sleepy person are usually different compared to fit drivers. This technique is increasingly popular, because of its non-intrusive approach in monitoring the alertness of a driver. A set of camera and image processing technology (computer vision) can be used to capture and process video or pictures. Based on driver's facial appearance, a technology is used to extract visual characteristics to define driver’s alertness level.

This paper introduces a drowsiness detection system that uses a CNN model to detect drowsiness, which is integrated with the Flutter mobile application. The system picks up images from the Flutter app and predicts them. It uses deep learning to extract meaningful patterns from facial images and classify them as asleep or awake. This system integrated with hardware components. Also, the system provides different levels of alerts when driver drowsiness is suspected. This cost-effective solution incorporates useful tools such as automated responses using smartphones that people already own. By leveraging these existing technologies, this app offers a practical and accessible way to promote road safety and prevent accidents.

**2. RELATED WORK**

According to research, there are multiple categories of technologies that can detect driver drowsiness. the use of cameras to monitor a person’s behavior. This includes monitoring their pupils, mouth for yawning, head position, and a variety of other factors. And other technologies are voice recognition. Often a person’s voice can give off clues on how fatigued they are. also, Techniques are such as:

ECG and EEG: the author of that paper [21] said that “Several authors have proposed different approaches for Drowsiness detection systems, most of them using ECG, Vehicle Based approaches. A robust real- time embedded platform to monitor the loss of attention of the driver during day and night driving conditions. A drowsiness detection system using both brain and visual activity. The brain activity is monitored using a single electroencephalographic (EEG) channel”. Nissan company has been researching and developing technology to detect driver fatigue and drowsiness using EEG signals. and integrating this information with vehicle control systems [22].

Steering Wheel Movement (SWM): An important vehicle-based method for determining drowsy driving is the monitoring of a driver’s Steering Wheel Movements (SWM) for drowsy patterns. The correlation between a driver’s intervals of steering adjustments and their level of drowsiness has been consistently seen by researchers. It has been demonstrated that as the majority of sampled drivers become drowsy, they tend to increasingly trend towards faster and larger steering corrections [23][24]. Valeo company uses this technique as part of the hole drowsiness detection System [25].

Optical Detection: the author of the paper [26] said that “A novel method based on an optical correlator is proposed for eye detection first, followed by eye state estimation. The second aim of this work was to investigate the simulated optical correlator on a real-world driving video for driver drowsiness detection. For the first time, a numerical simulation of the Vander Lugt Correlator (VLC) was used for automatically detecting the eye center “. Smart Eye: Smart Eye Technology provides eye tracking technology for a range of applications, including automotive safety. Their systems often incorporate optical sensors, such as cameras, to track the driver's eye movements and analyze patterns indicative of drowsiness or distraction.

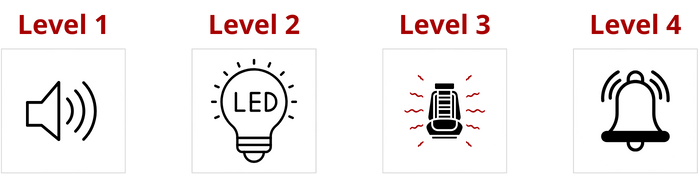
All those methods work will really and make sense with real world, but the problem is that they forget the budget of the different user all those methods are directed towards those owners of luxury cars, without interest to taxi drivers, long-haul truckers, and other simple individuals.

**3. SOFTWARE DESCRIPTION**

This application was developed to improve road safety by alerting drivers when they show signs of fatigue. The system consists of three main components: a convolutional neural network (CNN) model, a Flask application, and a Flutter mobile app. The CNN model is the core of the system, as it is responsible for analyzing the images of the driver and classifying them as either awake or drowsy. The model was trained on a large dataset of driver images, and it can extract facial landmarks from the images to make accurate predictions. The Flask application is the backend server of the system, which hosts the CNN model and communicates with the Flutter app. The Flask application receives the images from the Flutter app via HTTP requests, processes them using the CNN model, and returns the prediction results to the Flutter app.

The Flutter mobile app is the user interface of the system, which runs on the driver’s smartphone and interacts with the camera and the Flask server. The Flutter app captures images of the driver periodically using the camera preview and a timer, sends them to the Flask server for analysis, and notifies the driver when drowsiness is detected using various alert methods, such as shown in **Fig.1:**

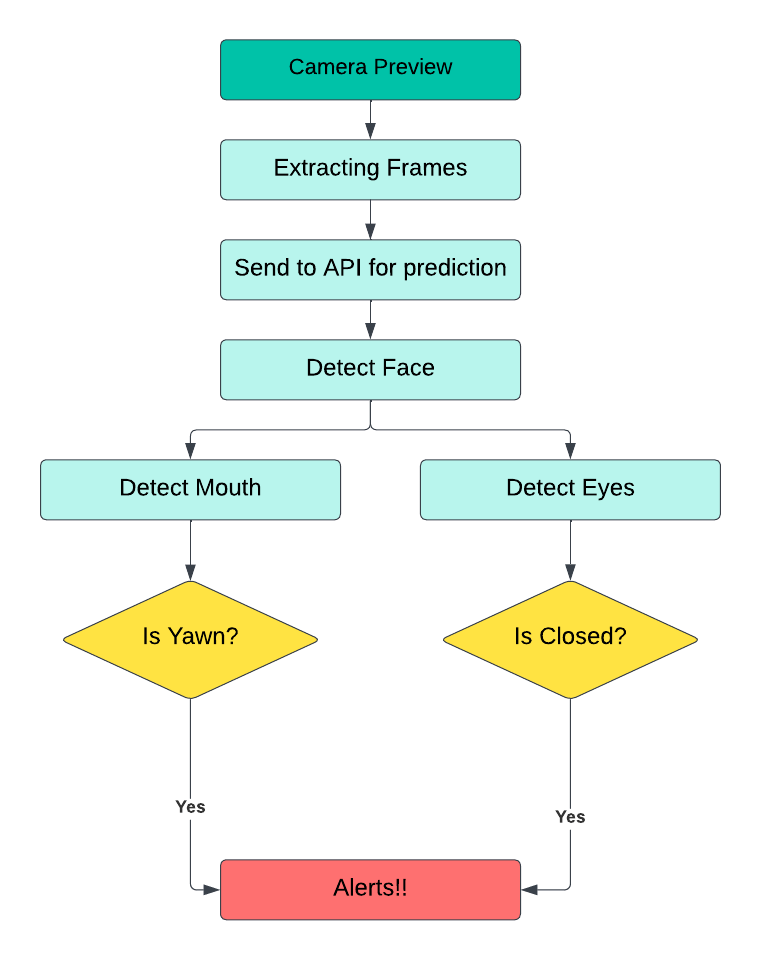
1. **Sound:** The app plays a loud and distinctive sound to wake up the driver.
2. **LED:** The app connects to an LED bulb attached to an ESP board, which flashes a bright light when the driver is drowsy.
3. **Vibration:** The app connects to a vibration motor attached to the driver’s seat, which vibrates when drowsiness is detected.
4. **Notification:** The app sends a notification to an interested person (e.g., a family member or a supervisor) of the status of the driver. This notification is sent periodically (e.g., every 2 seconds).



**Fig.1** The levels of alerts

**3.1. Software architecture**

Shows the architecture layers of the application is presented in **Fig.2** The mobile application container is divided into two parts: frontend and backend.



**Fig.2** General architecture of drowsiness detection

**Backend (Server side):** is a Flask-based web service designed to predict drowsiness in individuals based on facial features extracted from input images. The architecture follows a simple yet effective structure, incorporating image processing, deep learning model integration, and RESTful API principles. The application uses (**dlib**) [27] to detect facial landmarks. These landmarks are crucial for accurately cropping specific facial regions, such as the eyes and mouth. The cropped and resized regions of the eyes and mouth are input images into a CNN model. The result is determined by a combination of eye and mouth predictions, classifying the person as either **"Drowsy"** or **"Awake."**

**Frontend (Client side):** is a dynamic and user-friendly Flutter mobile application developed using Android Studio. This app seamlessly integrates with the Flask-based backend through HTTP requests, utilizing the camera package for image capture and the http package for communication. Additionally, Firebase services, including Realtime Database, Firestore, and Cloud Messaging, enhance the app's functionality and user experience.

**3.2. Software Functionalities**

The key features of the proposed system are using deep learning algorithms to identify the drowsy state of a driver and monitoring the drivers while driving. Moreover, the system also involves an alert feature that helps us keep the driver awake when they fall asleep the alert comprises several levels, with the first level being an alarm in the Flutter application. The second level is hardware such as a light bulb that flashes and can be controlled by ESP. The third level is a vibration motor connected to the driver’s seat. The fourth and last level is a message from the driver’s phone to another phone that wants information about the driver’s status.

**3.3. Software Implementation**

**Flutter app** is designed with a primary focus on ensuring driver safety by effectively detecting drowsiness and providing timely alerts. The key functionalities of the app involve real-time monitoring, drowsiness prediction, and alert mechanisms. By using ‘camera’ package allowing the app to capture images that serve as input for the drowsiness prediction process. These images are taken by using ‘CameraController’ that is used to establish a connection with the device's camera and manage its configuration. It provides methods and properties to control various aspects of the camera, such as setting the resolution, adjusting exposure, capturing images, and starting/stopping video recording.

A periodic timer is implemented using **Timer.periodic** that triggers every 1000 milliseconds (1 seconds). HTTP is primarily designed for text-based data transfer. Images, being binary data, are not directly suitable for inclusion in the body of an HTTP request. By converting the image to a base64-encoded string, it becomes a text-based representation that can be easily included in the request body.Base64 encoding is a way to represent binary data using only ASCII characters. This is useful when you need to serialize binary data, such as images, into a format that can be safely and reliably transmitted over text-based protocols like HTTP. “http” package is used as a means to send images to the API. The http package provides convenient methods to handle request parameters and response data. This includes encoding and decoding JSON data, handling query parameters, and managing response status codes and bodies. Networking operations can be time-consuming, and Flutter applications should not block the main thread while waiting for a response from an API. The http package supports asynchronous operations, allowing requests to be made without freezing the user interface. HTTP post request is represented as a function called ‘predict’ in flutter source code.

The function uses the **http.post** method to send an HTTP POST request to the specified uriAPI. The uriAPI is a variable that holds the endpoint or URL of the prediction API. The request includes headers, specifically setting the 'Content-Type' to 'application/json'. This indicates that the data being sent in the body of the request is in JSON format. The body of the request is a JSON-encoded string containing an object with a single key-value pair. The key is 'image', and the value is the base64-encoded image data. The app UI has the capabilities of taking a photo from camera or from gallery to be sent for prediction as a single image. The Main capability is displaying the ‘CameraPreview’ widget that is used for showing a live feed from the device's camera, and the ‘CameraController’ is used to manage the camera. Above the live camera preview, recognition values are dynamically printed, providing insights into the stream of processed images.

App functionality includes a counter to keep track of consecutive drowsiness detections. Different actions are taken based on the number of consecutive drowsiness detections, such as sending alerts to Firebase and playing alert sounds. Logic and action related to value of counter is described below:

* After 3 consecutive drowsiness detections, send level 1 to Firebase Realtime Database and play an alert sound.
* After 6 consecutive drowsiness detections, send level 2 to the firebase and play an alert sound.
* After 9 consecutive drowsiness detections, send level 3 to the firebase and play an alert sound.
* After 12 or more consecutive detections with even counts (12, 14, 16, …), play an alert sound and send a notification.
* If the recognition result does not indicate the driver to be drowsy, reset the counter to zero, send to firebase level 0 and stop playing alert sound.

This system incorporates an **ESP** (Embedded Systems) device equipped with a flash LED bulb and a vibration motor, designed to enhance real-time alerts based on drowsiness detection. The communication between the Flutter application and the ESP is facilitated through Firebase Realtime Database. The Flutter application utilizes Firebase Realtime Database to store and retrieve alert levels (1, 2, 3) based on the consecutive detection of drowsiness. When the Flutter application detects a specific threshold of consecutive drowsiness instances (3, 6, 9), it sends corresponding alert levels (1, 2, 3) to the Firebase Realtime Database.

The ESP device is configured to continuously monitor the alert level stored in the Firebase Realtime Database. - Upon detecting a change in the database, the ESP interprets the new alert level and responds accordingly. When the database reflects an alert level of 2, the ESP activates the flash LED bulb, providing a visual alert hint. In response to an alert level of 3, the ESP engages the vibration motor, generating a tactile alert to notify the user of a detected drowsiness state. By integrating Firebase Realtime Database with the ESP, the system creates a synchronized and responsive mechanism. As the Flutter application analyzes real-time data and triggers alert levels, the ESP seamlessly adapts its visual and vibration alerts. This integration enhances user awareness and safety, especially in scenarios where visual and tactile cues are critical, such as during driving.

This integration between the Flutter application, Firebase Realtime Database, and the ESP device establishes an effective and responsive drowsiness detection and alerting system, contributing to overall user safety and attentiveness.

Several hardware elements, including an LED bulb and vibrating motor, are linked and managed by the ESP32 microcontroller. In **Fig.6** illustrates the hardware components, highlighting the ESP32 and the mentioned parts. The fundamental concept revolves around the ESP32 monitoring a value from the Firebase Realtime Database, which is stored in a variable named "level," denoting the instances when the driver is detected as drowsy. At the initial level, a sound is prompted to play within the application. Subsequently, upon reaching the second level, the ESP32 activates the LED bulb, while the third level initiates the vibrating motor.

In addition to the alerts described above provided to the driver, the drowsiness detection system incorporates a feature to notify a family member or interested party about the driver's state. This is achieved through a notification sent to their mobile device, where the same application is installed. The system monitors the driver's drowsiness using a counter mechanism. When the drowsiness counter reaches or exceeds 12 consecutive instances, indicating an extended period of potential drowsiness, the notification alert mechanism is triggered. Notifications are not sent on every drowsy detection but are triggered strategically. After the drowsiness counter exceeds 12 times and becomes an even number (14, 16, 18, ...), a notification is generated and sent to the family member's phone. Notification content includes a prominent alert message, such as "Alert!!!!: Driver is drowsy" and additional information conveying the critical nature of the situation. By extending alerts beyond the driver's immediate environment, the system enhances the safety net for the driver. Family members, informed through timely notifications, can take appropriate actions to ensure the well-being of the driver. This notification alert system adds an additional layer of safety and awareness by extending alerts to trusted individuals. By using the same application on the family member's device, the system establishes a network of communication that contributes to a comprehensive and supportive drowsiness detection and alerting solution.

**The backend** of the drowsiness detection system is implemented as a Flask API, providing a robust and efficient solution for processing images, detecting facial features, and predicting drowsiness. The key functionalities include image processing, feature extraction (eyes and mouth cropping), and real-time drowsiness prediction. Upon receiving an image from the Flutter application, the Flask API utilizes advanced image processing techniques to prepare the input for drowsiness detection. Image processing includes tasks such as resizing, normalization, enhancement for optimal feature extraction. The API employs facial recognition algorithms to accurately locate and crop the regions of interest, specifically the eyes and mouth, from the processed image. - Cropping these facial features provides focused input for the drowsiness prediction model, enhancing the accuracy of the detection process. A CNN deep learning model for drowsiness detection is integrated into the Flask API. This model is capable of analyzing the cropped eye and mouth regions to predict the likelihood of drowsiness. The prediction results are generated in real-time, providing immediate feedback on the driver's alertness level. Once the drowsiness prediction is complete, the Flask API sends the results back to the Flutter application. This communication is often facilitated through a structured JSON response, containing the prediction outcome and any relevant metadata. The Flask API is designed to handle real-time image processing and prediction requests efficiently. To ensure the security and integrity of data, the communication between the Flutter app and the Flask API is typically secured using encryption protocols (e.g., HTTPS). This safeguards sensitive information during transmission.

The backend system is built to accommodate future improvements, such as model updates or enhancements to image processing techniques. This ensures adaptability and long-term effectiveness in drowsiness detection. The Flask API utilizes several libraries and frameworks, including:

* **Flask** for creating the web application.
* **Keras** for loading the pre-trained drowsiness prediction model.
* **OpenCV** for image processing.
* **base64** for decoding Base64-encoded ASCII characters into binary data.
* **PIL** is a powerful library for image processing tasks.
* **NumPy** for numerical operations
* **dlib** for facial feature detection.

Image is sent to an endpoint that can handle HTTP POST requests. It assumes that the request body contains JSON data. The extracted value associated with the key 'image' from the JSON data is assumed to be A base64-encoded string representing an image. The base64-encoded string is then decoded into its original binary form using the base64 module. The result is a binary representation of the image data.

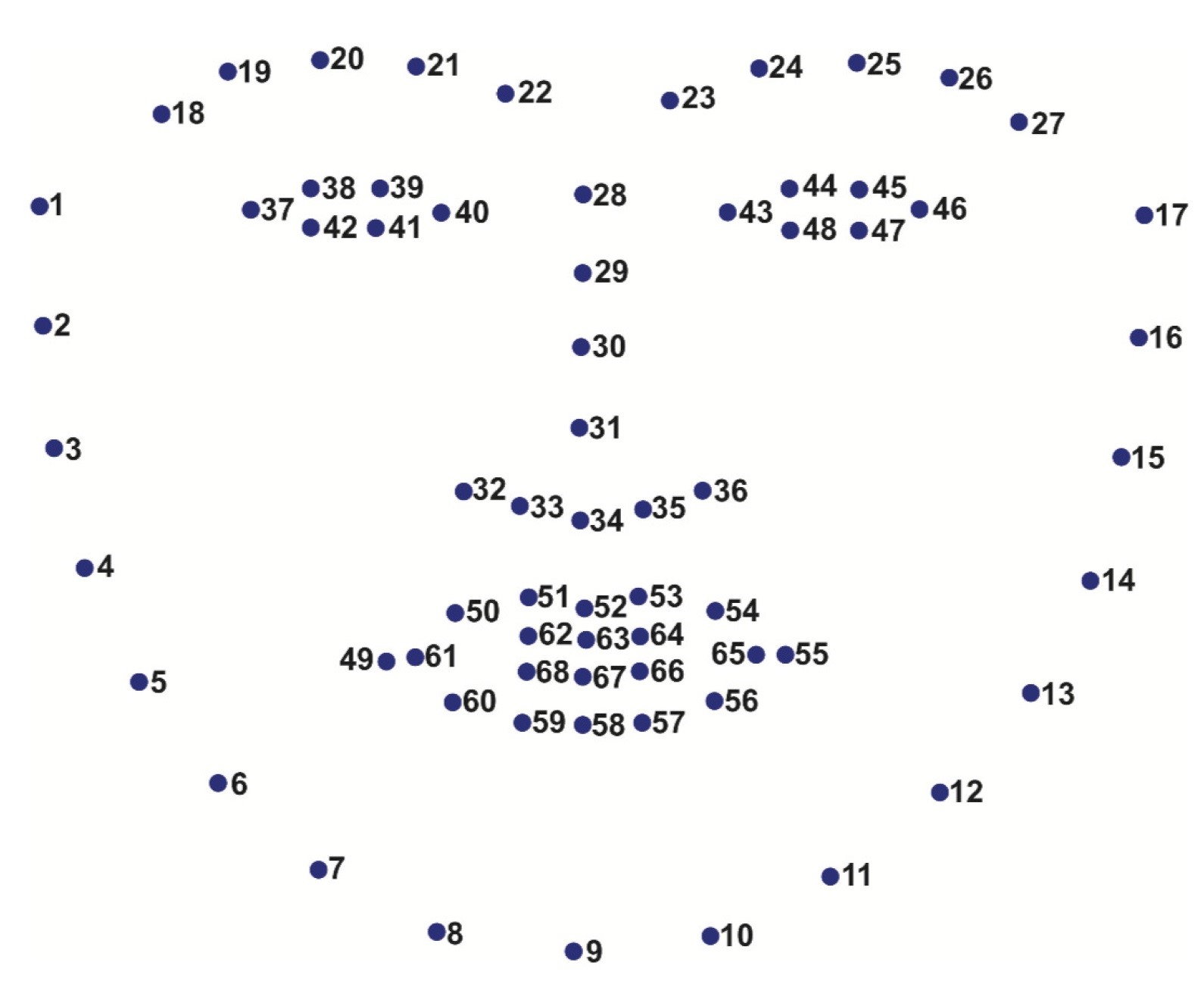
The binary image data is wrapped in a ‘**BytesIO’** object from the ‘io’ module and then opened as an image using the Image.open method from the Pillow (PIL) library. This method effectively converts the binary data into an image object that can be further processed or analyzed. Image is passed to a function called ‘**exif\_transpose’.** The purpose of this function is to examine the ‘Exif’ metadata of an image and, if an orientation correction is required, apply the necessary transformations to bring the image to its default orientation. The function iterates through the Exif tags and identifies the 'Orientation' tag. If found, it retrieves the orientation value from the Exif metadata. Depending on the detected orientation value, the function applies rotation transformations to the image using the **`transpose`** method from the Pillow (PIL) library.

The following corrections are applied:

* If the orientation is 3, the image is rotated 180 degrees.
* If the orientation is 6, the image is rotated 270 degrees.
* If the orientation is 8, the image is rotated 90 degrees.

The function is wrapped in a try-except block to handle potential exceptions during the Exif orientation check or the image transformation process. Any encountered errors are printed to the console for debugging purposes. This function is typically applied to the image received from the Flutter application after decoding the base64-encoded image. It ensures that the image is in its default orientation. The corrected image is returned from the function, and it is then ready for further analysis within the drowsiness detection system. Driver image is then passed to functions (**crop\_mouth** and **crop\_eye)** which utilize the functionalities of ‘dlib’ library for facial landmarks detection.

#### Identification of Facial Landmarks **dlib** [27]is a C++ library that provides tools and algorithms for machine learning, computer vision, and image processing tasks. It presents a method that precisely estimates the positions of facial landmarks using a training set of labeled facial landmarks on images. This method can be used for real-time detection to identify the facial features after detecting faces on an image. This face features detector identifies 68 main positions (x, y coordinates) on human faces, as shown in **Fig.3**

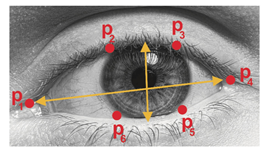


**Fig.3** Illustration of 68 landmarks on a human face.

The positions of facial landmarks are defined as follows: jaw: points 1 to 17; right eyebrow: points 18 to 22; left eyebrow: points from 23 to 27; nose: points 28 to 36; right eye: points 37 to 42; left eye: points 43 to 48; mouth: points 49 to 68. In this work, we only use 32/68 feature points, namely coordinates of the left eye, right eye, and mouth to calculate eye-opening (EAR) and mouth-opening (𝐿𝐼𝑃𝑑𝑖𝑠𝑡𝑎𝑛𝑐𝑒) [28].

#### **Blink Detection:** To determine the feature positions, we apply facial feature detection including the eyes, eyebrows, nose, ears, and mouth [29][30]. We extract specific facial structures by knowing the index of specific parts of a face. For blink detection, we are interested in eyes features. Each eye is represented by six coordinates, as illustrated in **Fig.4**, starting at the left corner of the eye and going clockwise around the rest of the area [28].

𝐸𝐴𝑅=(|𝑝2−𝑝6|+|𝑝3−𝑝5|)(2|𝑝1−𝑝4|). (1)

 **Fig.4** Example of 6 facial landmarks related to eyes.

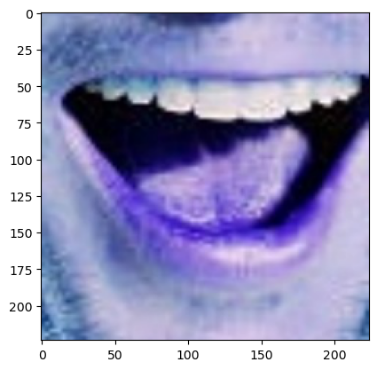
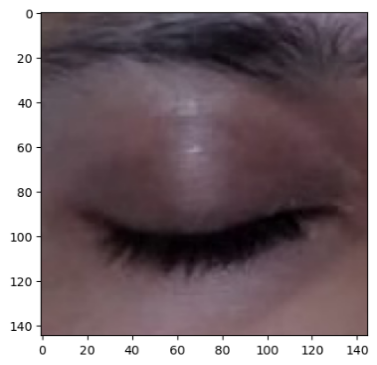
Based on Soukupova and Cech [13], we can derive **Equation (1)**, reflecting the eye-opening value, where 𝑝1,…,𝑝6 are features located on a 2D face. The numerator of **Equation (1)** calculates the distance between the eye features vertically, while the denominator calculates the distance between the eye features horizontally. Denominator balancing is suitable since there is only one set of horizontal points but there are two sets of vertical points. The eye-opening value is almost unchanged when the eye is opened, but it quickly decreases to 0 when blinks occur. Using this method, we can avoid image processing techniques and simply rely on eye-opening values to determine blinking actions.

**Yawning Detection:** Yawning is characterized by mouth-opening. Like blink detection, facial features are used to detect an open mouth. The mouth-opening value is a parameter used to determine whether a mouth is open. It is calculated by subtracting the mean of all points in the upper mouth with the mean of all points in the lower mouth [16], as in **Equation (2)**.

𝐿𝐼𝑃𝑑𝑖𝑠𝑡𝑎𝑛𝑐𝑒=|𝐿𝐼𝑃𝑇𝑜𝑝−𝐿𝐼𝑃𝐿𝑜𝑤| (2)

**‘crop\_mouth`** Function: This function takes an image (`img`) as input and aims to detect the landmarks of the driver's face using the dlib library. It specifically focuses on extracting the coordinates of the mouth landmarks. The facial landmark detection is performed by first identifying bounding boxes around faces in the image (`dets`). The landmarks are then obtained for each detected face using the `predictor` object. The coordinates of the mouth landmarks (indexed from 48 to 67) are retrieved, and a rectangular region of interest (ROI) around the mouth is extracted from the original image. The extracted mouth region is then converted to RGB format, resized to a target size (default is 224x224 pixels), and returned as the output. If no face is detected, or an error occurs, `None` is returned. **‘crop\_eye`** Function: Similar to the `crop\_mouth` function, the `crop\_eye` function is responsible for detecting the landmarks of the driver's face, with a focus on extracting the coordinates of the eye landmarks. The facial landmark detection process is the same as in the `crop\_mouth` function, identifying bounding boxes, obtaining landmarks, and extracting the eye coordinates (indexed from 18 to 41). A rectangular region of interest (ROI) around the eye is then extracted from the original image, converted to RGB format, resized to a target size (default is 224x224 pixels), and returned as the output.

In **Fig.5** the cropped mouth and eye images are normalized by dividing the pixel values by 255. This step ensures that the pixel values are within the range [0, 1], in addition to adding an extra dimension to represent the batch size, which is a common practice when working with deep learning frameworks.

**Fig.5** Example of cropped eye and mouth

The normalized images are then fed into the pre-trained drowsiness prediction model. The model predicts the likelihood of drowsiness based on the characteristics of the mouth and eyes regions. Flask App then identifies the index of the maximum prediction value and determines the corresponding drowsiness classification using a predefined mapping list. If the highest prediction probability exceeds a threshold of 0.50, the result is considered reliable; otherwise, it is labeled as a 'Weak Prediction. As for the last step, “final\_result” is initialized as an empty string. This variable will eventually hold the overall drowsiness state determined by the combined analysis of the mouth and eye regions.

The if statement checks whether either the mouth indicates a "yawn" or the result from the eye analysis is "Closed." If either of these conditions is true, it concludes that the driver is drowsy, and the “final\_result” is set to "Drowsy." Otherwise, if neither condition is met, it is assumed that the driver is awake, and “final\_result” is set to 'Awake.' **API** returns **“final\_result”** as a JSON response. This is common in web applications or APIs where data is often exchanged in JSON format.

**3.4. Hardware Components**

As previously noted, the system is enhanced with hardware components, including the incorporation of an ESP32 microcontroller board which is a widely used microcontroller and Wi-Fi module developed by Espressif Systems. It is part of the ESP (Espressif Systems' Platform) family of microcontrollers, known for their versatility, low cost, and integrated Wi-Fi and Bluetooth capabilities. The ESP32 is popular in various IoT (Internet of Things) applications, including home automation, industrial automation, smart devices, and more. It offers a powerful dual-core processor, ample memory, a variety of built-in peripherals, and support for multiple communication protocols, making it suitable for a wide range of projects.

Several hardware elements, including an LED bulb and vibrating motor, are linked and managed by the ESP32 microcontroller. **Fig.6** illustrates the hardware components, highlighting the ESP32 and the mentioned parts. The fundamental concept revolves around the ESP32 monitoring a value from the Firebase Realtime Database, which is stored in a variable named "level," denoting the instances when the driver is detected as drowsy. Esp32 has access to the firebase file using its URL over the internet network by the embedded Wi-Fi module in esp32 then read the value of level. If it two, esp32 open the LEd by making ‘MED\_PIN’ high and it still working in the next level. If the application reached to the third level esp32 open the motor using relay module and battery, as esp32 can't itself run our motor so we helped by external battery, and relay module that works as switch, so it connected first with esp32 and second between the battery and the motor, the operation work as the following, first when esp32 read value three from the firebase file it make the ‘RELAY\_PIN’ high so it work as A circuit board with wires and a battery

Description automatically generatedopened switch then the voltage passes from the battery and run the motor. On the fourth level we hope that our system can be embedded in the car and connected with its system so if the car has self-drive system, it takes place automatically or park the car.

**Fig.6** Hardware components

The Arduino Code that implemented to ESP32 is designed to integrate the ESP32 with Firebase for the purpose of receiving and processing data pertaining to driver behavior detection. Here's a summary of its functionality:

* **Library Inclusion:** The code includes essential libraries for handling JSON data and Firebase interaction.
* **Wi-Fi Setup**: It specifies the Wi-Fi credentials and the URL reference for the Firebase project. Pin Definitions: Pins for controlling the relay, initialization, medium level, and self-drive are defined.
* **Setup Procedure:** Serial communication is initialized, Wi-Fi connection is established, and pin modes are configured.
* **Firebase Initialization:** Connection to Firebase is initialized. Main Loop: This loop continuously operates after setup. It retrieves data ("data") from the Firebase database, parses it using the Arduino Json library, and stores it in variables. Subsequently, it processes the received data based on the "level" value. If "level" is 1 or greater, it activates the initialization pin. If "level" is 2 or greater, it activates the medium level pin which operate the led bulb. If "level" is 3 or greater, it activates the relay pin to operate the motor. If "level" is less than 1, it deactivates all pins.
* **Delay:** A delay is incorporated to control the frequency of data processing.

# **3.5. Results and discussion**

"The Convolutional Neural Network (CNN) model architecture used in this project was adapted from a Kaggle notebook authored by [‘ADITTA DAS NISHAD’] [31]. The original work provided valuable insights and served as a foundational reference for implementing the model in this project. The drowsiness detection system employs a Convolutional Neural Network (CNN) as the underlying machine learning model. This CNN has been trained on a diverse and comprehensive dataset obtained from Kaggle. The CNN architecture is designed to effectively analyze and extract features from images, making it well-suited for computer vision tasks such as facial feature recognition.

The model has been trained on a Kaggle dataset that includes a variety of facial images representing different states related to drowsiness. The dataset is categorized into four distinct classes: "yawn": Represents images where the driver is exhibiting a yawn. "no\_yawn": Indicates images where there is no evidence of a yawn. "Closed": Corresponds to images capturing closed eyes. "Open": Represents images with open eyes. Each image in the dataset is labeled with one of the previously mentioned classes, allowing the model to learn and generalize patterns associated with different facial expressions and eye states. The model's output provides probabilities for each of the four classes. The class with the highest probability is considered the predicted class for a given input image. In the context of the complete drowsiness detection system, the predictions from the CNN model contribute to the comprehensive analysis of the driver's facial features, including the mouth and eye regions.

**3.5.1 Utilized Dataset Collection**

The dataset utilized in this project has been collected from various sources available on Kaggle, a prominent platform for sharing and discovering datasets. The dataset is designed to facilitate the development and training of a drowsiness detection system, particularly focusing on the analysis of facial features such as eyes and mouth.

* Eyes Closed and Open Dataset: This subset of the dataset comprises images capturing the condition of closed-open eyes. The dataset includes a diverse range of individuals with closed-open eyes, reflecting variations in lighting conditions, facial expressions, and orientations [32].
* Mouth Images of Yawning Dataset: This section of the dataset focuses on images capturing instances of yawning. Yawning is a common indicator of drowsiness, and the dataset includes images reflecting different individuals displaying yawning behavior [31][32][33].

The dataset is carefully selected to ensure diversity in terms of subjects, capturing a wide range of facial features, skin tones, and backgrounds. This diversity contributes to the model's ability to generalize across various demographics. Each image in the dataset is labeled according to its corresponding class, such as "Closed Eyes," "Open Eyes," "Yawn," or "No\_Yawn." Proper labeling is crucial for supervised learning, allowing the model to associate visual patterns with specific drowsiness indicators. Efforts have been made to maintain a balanced representation of different classes within the dataset. Balancing helps prevent the model from becoming biased toward any specific class, ensuring fair and accurate predictions. Prior to model training, the dataset underwent quality assurance checks to filter out any irrelevant or corrupted images. This ensures that the model learns from high-quality, relevant examples.

**3.5.2 Input Data Preparation Functions**

* mouth\_images: This function is designed to facilitate the loading and preprocessing of images associated with yawning and non-yawning facial expressions. A list comprising pairs of resized image arrays and their corresponding class labels. The class labels are encoded as follows: "yawn": 0 ,"no\_yawn": 1 For each image within these classes, this function reads, resizes, and appends the image array along with its encoded class label to the defined list.
* eyes\_images: This function focuses on loading and preprocessing images related to open and closed eyes. a list containing pairs of resized image arrays and their corresponding class labels. The class labels are encoded as follows: "Closed": 2, "Open": 3

Two datasets are combined into a single list, creating a unified dataset for subsequent model training. The next crucial step is segregating image data from their corresponding class labels within the augmented dataset. This process is fundamental for organizing the data into feature sets (X) and label sets (y) required for deep learning model training. At the conclusion of this process, the X list contains all the image arrays, while the y list contains their corresponding class labels. These separated lists provide a clear distinction between the input data (features) and the target variable (labels). Images array(X) is converted to Numpy array and reshaped to (-1,224,224,3), where ‘224’ and ‘224’ are the height and width dimensions of the images. Labels set(y) is fed into ‘LabelBinarizer’ Object, this method transforms the labels into binary format. Dataset is augmented to ‘ImageDataGenerator’ that contributes to the balanced generation of training and testing datasets. Data augmentation techniques in the training set help increase the diversity of the data, while the testing set remains untouched to assess model performance on the original, unaltered data.

**3.5.3. Model Architecture Explanation**

The model begins with an input layer, shaped to accommodate the dimensions of the training data (224,224,3), which represents the dimensions of the images. A series of convolutional layers (Conv2D) with increasing filter sizes (256, 128, 64, 32) and a common kernel size of (3, 3). Each layer employs the Rectified Linear Unit (ReLU) activation function to introduce non-linearity. After each convolutional layer, a max-pooling layer (MaxPooling2D) is applied with a pool size of (2, 2).

Max-pooling reduces spatial dimensions, capturing the most essential features. The Flatten layer reshapes the output of the convolutional layers into a flat vector, preparing it for input into the fully connected layers. A Dropout layer with a dropout rate of 0.5 is incorporated for regularization. Dropout helps prevent overfitting by randomly dropping out a fraction of input units during training. Two dense layers follow, with 64 units and ReLU activation for the hidden layer and 4 units with softmax activation for the output layer. Softmax activation normalizes the output, providing probabilities for each class. The model is compiled using the categorical crossentropy loss function, accuracy as the evaluation metric, and the Adam optimizer. Model Architecture is shown in **Fig.7**.

A diagram of a diagram of a diagram

Description automatically generated

**Fig.7** Model Architecture

**3.5.4 Model Performance Metrics.**

As mentioned before the described model relies on a Kaggle notebook authored by 'ADITTA DAS NISHAD.'. This model involves four distinct classes: 'no\_yawn,' 'yawn,' 'Closed,' and 'Open.' The dataset utilized includes images classes of closed and open eyes, along with full face images capturing the driver for 'yawn' and 'no\_yawn' categories. Subsequently, these face images are processed within the notebook, employing **Haarcascade** for frontal face detection, and the resulting images are then used to train the model across the four classes. However, it was observed that the model encountered challenges when capturing images using a camera phone and attempting to crop the driver's image to extract the face and eyes using **Haarcascade** files. Many full images were noted that the **Haarcascade** files failed to accurately extract the face or eyes, exploring methods for a more effective approach to cropping facial landmarks and achieving a more precise detection of the driver's drowsy state.

In response, the **MTCNN** (Multi-task Cascaded Convolutional Networks) function was implemented, proving to be highly effective but takes much time-consuming for processing and cropping the relevant areas. Attempts were made to combine MTCNN for facial cropping and **Haarcascade** for subsequent eye extraction, but challenges still existed, particularly concerning processing time and unsuccessful eye extraction. we named this model CNN\_Model\_1.

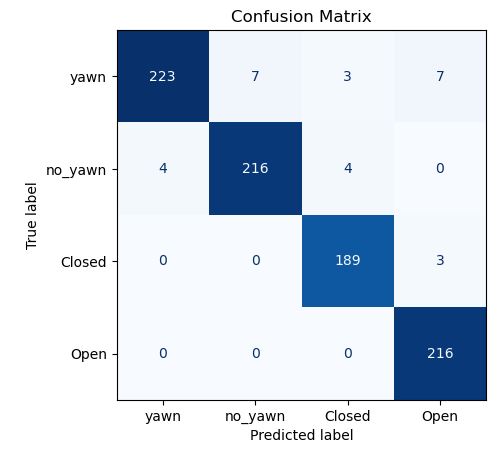
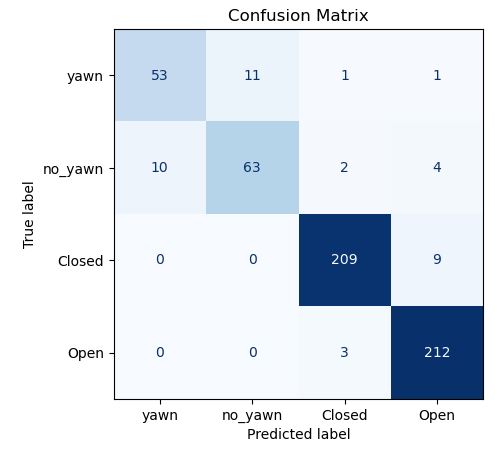
Consequently, an alternative strategy was proposed by make use of the capabilities of **Dlib** landmarks, which demonstrated remarkable effectiveness in extracting facial landmarks from driver images. A focused effort was then directed towards identifying specific facial landmarks, particularly the eyes and mouth, which play a crucial role in detecting the drowsy state of the driver. The approach involved using full driver images to crop the mouth and eyes and employing these cropped images for model training. This optimization significantly improved the model's ability to predict images captured from camera phones. we named this model CNN\_Model\_2

It is essential to note that these cropping functions for the mouth and eyes are applied within the Flask application before entering the prediction phase. This comprehensive approach enhances the overall effectiveness of the system in accurately identifying and predicting the drowsy state of the driver based on images obtained from camera phones.

The primary purpose for integrating the **dlib** library and retraining the model on cropped images of eyes and mouth is to enhance the performance of the prediction process when utilizing images captured by mobile phones. By employing efficient methods for extracting facial landmarks of eyes and mouth and then passing these cropped images to a model trained on similar cropped data, we achieve better results in real-time prediction of the driver's status using a camera phone. It is crucial to mention that the same model architecture was applied in both scenarios, with the only variation being the dataset on which the model was trained. Table 1 presented the performance comparison between CNN\_Model\_1 and CNN\_Model\_2, the confusion matrices are shown in figure 8.

**Table 1**: Performance comparisons between CNN\_Model\_1 and CNN\_Model\_2

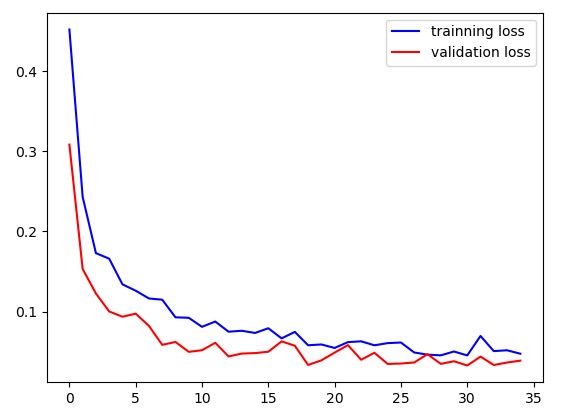
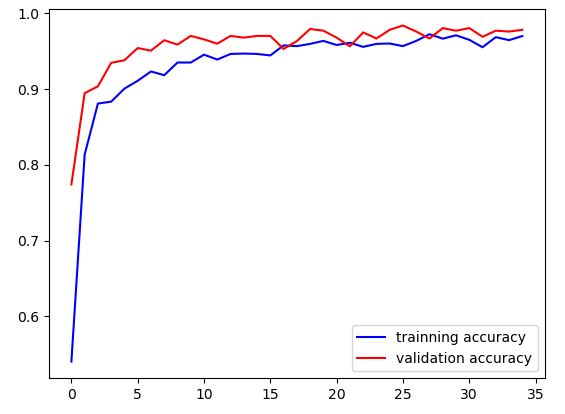
|  |  |  |
| --- | --- | --- |
|  | CNN\_Model\_1 | CNN\_Model\_2 |
| **Dataset** | Includes images of pre-cropped eyes and face images obtained through the utilization of the **Haarcascade** classifier, specifically applied for frontal face detection. | Images consist of pre-cropped eyes and mouth, achieved through the utilization of the **dlib** toolkit for accurate facial landmark detection and extraction. |
| **Accuracy** | accuracy: 0.9540  val\_accuracy: 0.9585 | accuracy: 0.9700  val\_accuracy: 0.9782 |
| **Precision** | 0.864 | 0.968 |
| **Recall** | 0.853 | 0.981 |
| **F1 Score** | 0.928 | 0.968 |



1. (B)

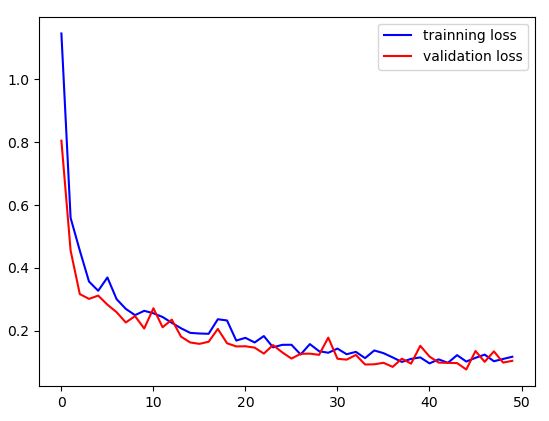
**Fig.8** Confusion Matrices (A) for CNN\_Model\_1, (B) for CNN\_Model\_2.

By comparing certain attributes between two CNN models, it was observed that model CNN1 achieved a validation accuracy of 0.9585 after 50 epochs. In contrast, our modified in model CNN2 reaching a validation accuracy of 0.9782 after 35 epochs. Additionally, a detailed comparison of the training and validation accuracy/loss curves is provided for further analysis. Shown in **Fig.9, 10**



1. (B)

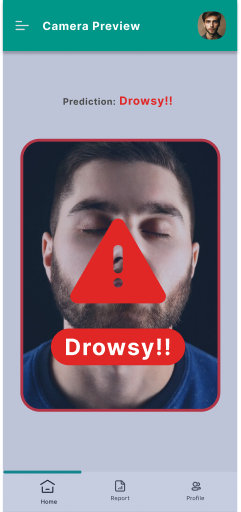
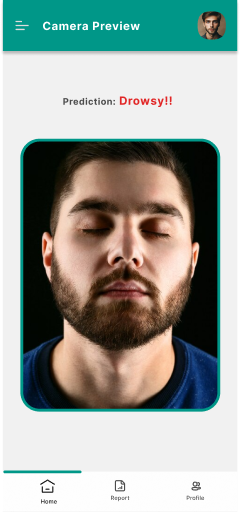
**Fig.9** Training Validation Accuracy/Loss for CNN\_Model\_1.



**Fig.10** Training Validation Accuracy/Loss for CNN\_Model\_2.

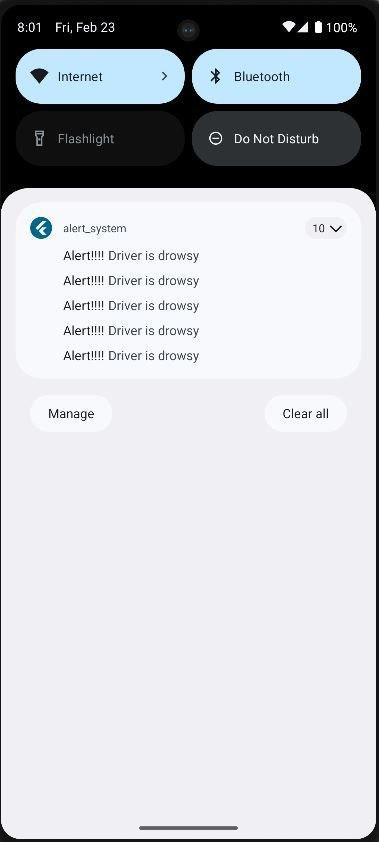
**4. ILLUSTRATIVE EXAMPLES**

When a driver starts their journey, they activate the RouteEye mobile app on their smartphone. The app uses the smartphone camera to periodically take photos of the driver's face while they are behind the wheel (see **Fig.11A**). During the trip, a CNN model embedded within the app analyzes these images in real-time, monitoring the driver's facial expressions and eye movements for signs of drowsiness. If the model detects any indicators of drowsiness, such as yawning of the mouth or closing of the eyes, the application triggers an alert mechanism to notify the driver and prevent potential accidents (see **Fig.11B**). If the app detects drowsiness, it may sound a loud and distinctive alarm (see **Fig. 11C**), flash bright LED lights, and activate a vibration motor installed in the driver's seat. Also, the driver can take precautionary action that the system can take if the system alerts failed to awaken the driver such as Contact one of the driver relatives the app sends a notification to the driver's fleet manager or supervisor, providing updates on the driver's condition (see **Fig. 11E**). The driver can also know the statistics of the alerts sent to him at each level (see **Fig. 11D**).





**A**   **B C**



**D E**

**Fig.11** (A) The camera preview that provide real time detection, (B) Start detection drowsy state then app in (c), send Alerts to the driver to attention, (D) Report recording alerts to the driver, (E) Notification sent to those following the driver when a state of drowsiness is detected.

**5. IMPACT**

The drowsiness detection application offers a comprehensive solution to a critical safety problem in the transportation sector and beyond. By integrating a convolutional neural network (CNN) model with an easy-to-use and efficient mobile application, it effectively identifies signs of driver drowsiness in real-time.

This innovative approach allows continuous monitoring of driver alertness through periodic analysis of facial expressions and eye movements captured by the smartphone camera. The app's multimedia alert system, including sound, light, vibration and notifications, ensures timely interventions to prevent drowsiness-related incidents. Furthermore, the application's integration with fleet management systems enables proactive safety measures and real-time monitoring of driver behavior for commercial carriers. Rigorous testing and optimization ensure the solution's reliability and effectiveness across diverse driving conditions, contributing to enhanced road safety, improved productivity, and improved quality of life for drivers and other road users with low-cost mobile app, and accessible for all users. Ultimately, drowsiness detection represents an important step toward mitigating the risks of drowsy driving and promoting safer transportation practices. Collaboration between researchers, automotive manufacturers, and policymakers is vital for the successful implementation and standardization of drowsiness detection systems. Establishing guidelines, regulations, and industry standards can ensure the consistent development and deployment of effective solutions across different vehicles and regions.

**6. CONCLUSION**

This work solves driver drowsiness problem and the ways to preventing accidents on roads: Our drowsiness detection system utilizes various technologies to assess and analyze signs of drowsiness in individuals such as: Eye Tracking: monitoring eye movements, such as eyelid closure and blink patterns, can provide good indicators of drowsiness. Slow or slipping eyelids, long eye closure, or irregular blinking can suggest drowsiness. The system has levels of alerts to wake the driver and keep safety. Deep Learning Algorithms: Advanced algorithms can be trained using data from various sensors to recognize patterns associated with drowsiness. These algorithms can learn to identify specific features and patterns indicative of drowsiness and make real-time predictions. The application described in this document is a Flutter app that employs deep learning principles and advanced image processing techniques from CNN models, as a result our CNN\_Model\_1 achieved accuracy: 0.9540 and val\_accuracy: 0.9585 and our CNN\_Model\_2 achieved accuracy: 0.9700 and val\_accuracy: 0.9782. Its primary objective is to identify signs of driver drowsiness during driving activities and implement various alert methods to awaken the driver, thereby improving road safety and minimizing the risk of accidents caused by driver fatigue. IN the future, Continuous research and improvement are essential for refining existing algorithms and techniques. Collecting large-scale datasets, considering diverse driving scenarios, and addressing individual differences in drowsiness patterns can lead to more accurate and robust drowsiness detection models. User acceptance and usability of drowsiness detection systems should be considered. Factors such as system reliability, user comfort, and privacy concerns should be considered during the design and implementation of these systems to encourage widespread adoption among drivers.

**7. FUTURE WORK**

The drowsiness detection mobile application has the potential to explore various future directions to enhance its functionality and effectiveness. One possibility involves integrating advanced sensors like infrared cameras or eye-tracking technology to gather additional physiological and behavioral data, thus improving drowsiness detection comprehensively. Furthermore, connecting the application with emerging technologies such as augmented reality (AR) or virtual reality (VR) could introduce innovative methods for real-time monitoring of driver alertness and timely delivery of alerts. Exploring multimodal approaches that combine data from various sensors, such as electroencephalography (EEG) or heart rate monitors, alongside facial feature analysis, could provide a more holistic assessment of driver alertness levels. Additionally, collaborating with sleep researchers and psychologists to gain insights into the underlying mechanisms of drowsiness and cognitive performance decline could inform the development of tailored interventions and personalized feedback strategies within the application. Lastly, Expanding the application's focus beyond drowsiness detection to encompass other aspects of driver behavior, like distraction or impairment, could further bolster overall road safety.

**References**

[1] World Health Organization, The top 10 causes of death. (2020, December 9). <https://www.who.int/news-room/fact-sheets/detail/the-top-10-causes-of-death>

[2] World Health Organization, Road traffic injuries. (2023, December 13). <https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries>

[3] ROSPA: Driver fatigue and road accidents: a literature review and position paper. Technical report, The Royal Society for the prevention of accidents, Birmingham U.K (2001).

[4] Morales, J.M., Stasi, L.L.D., Díaz-Piedra, C., Morillas, C., Romero, S.: Real-time monitoring of biomedical signals to improve road safety. In: International Work-Conference on Artificial Neural Networks, Pp. 89–97. Springer (2015).

[5] Akerstedt, T., Bassetti, C., Cirignotta, F., Garcia-borreguero, D., Goncalves, M., Horne, J., Leger, D., Partinen, M., Penzel, T., Philip, P., Verster, J.C.: Sleepiness at the wheel. Institut National du Sommeil et de la Vigilance European Sleep Research Society (2013).

[6] Basim N M A, Sathyabalan P and Suresh P 2015 Analysis of EEG signals and facial expressions to detect drowsiness and fatigue using Gabor Filters and SVM linear classifier Int. J. Computer Applications 115 (11)

[7] Mardi Z, Ashtiani S N M, Mikaili M. 2011 EEG-based drowsiness detection for safe driving using chaotic features and statistical tests. J Med Signals Sens 1 (2): 130–137

[8] Bhatt P P and Trivedi J A 2017 Various methods for driver drowsiness detection: an overview Int. J. Computer Science and Engineering (IJCSE) 9 (03 Mar 2017) 70-74

[9] Sahayadhas A, Sundaraj K, Murugappan M 2014 Electromyogram signal based hypovigilance detection. Biomedical Research: An International Journal of Medical Sciences ISSN: 0970- 938X

[10] Jabon ME, Bailenson JN, Pontikakis E, Takayama L and Nass C 2011 Facial-expression analysis for predicting unsafe driving Behaviour Face and Gesture Recognition O*ctober– December 2011 Pervasive Computing*

[11] Karchani M, Mazloumi A, Saraji G N, Gharagozlou F, Nahvi A, Haghighi K S, Abadi B M and Foroshani AR 2015 Presenting a model for dynamic facial expression changes in detecting drivers’ drowsiness *Electronic Physician* **7**(2):1073-1077

[12] Li Z, Li S E, Li R, Cheng B, and Shi J 2017 Online detection of driver fatigue using steering wheel angles for real driving conditions *Sensors 2017*, **17** 495; doi:10.3390/s17030495

[13] Alioua N, Amine A and Rziza M 2014 Driver’s fatigue detection based on yawning extraction, Int. J. Vehicular Technology, Volume 2014 7 pages

[14] Kumar N and Barwar N. 2014 Detection of eye blinking and yawning for monitoring driver’s drowsiness in real time, Int. J. Application or Innovation in Engineering & Management, 3 (11): November 2014

[15] Diddi V K and Jamge S B 2014 Head pose and Eye state Monitoring (HEM) for driver drowsiness detection: Overview, *IJISET – Int. J. Innovative Science, Engineering &* *Technology*, 1 (9): November 2014

[16] Hachisuka S, Ishida K, Enya T and Kamijo M 2011 Facial expression measurement for detecting driver drowsiness Int. Conf. on Engineering Psychology and Cognitive Ergonomics, EPCE 2011: Engineering Psychology and Cognitive Ergonomics pp 135-144

[17] Ilbeygi M and Hosseini H 2012 A novel fuzzy facial expression recognition system based on facial feature extraction from color face images Engineering Applications of Artificial Intelligence 25 (1):130-146.

[18] Jayswal A S and Modi R V 2017 Face and eye detection techniques for driver drowsiness detection Int. Research. J. Engineering and Technology (IRJET) 04 (04): Apr 2017

[19] Shamian M, Khan I, Abdullah H 2014 Efficient drowsiness detection by facial features monitor Research J. Applied Sciences, Engineering and Technology 7 (11): 2376-2380

[20] Sigari MH, Pourshahabi Soryani Fathy M 2014 A review on driver face monitoring systems for fatigue and distraction detection Int. J.of Advanced Science and Technology 64 (2014) 73- 100

[21] Saini, V., & Saini, R. (2014). Driver drowsiness detection system and techniques: a review. *International Journal of Computer Science and Information Technologies*, *5*(3), 4245-4249.‏

[22] Ghafarian, A., & Sardari, S. (2020, March). An Analysis of Connected Cars Technology and Security. In *Teoksessa International Conference on Cyber Warfare and Security* (Vol. 14, pp. 195-203).‏

[23] Lawoyin, S. A., Fei, D. Y., & Bai, O. (2014). A novel application of inertial measurement units (IMUs) as vehicular technologies for Drowsy driving detection via steering wheel movement. *Open Journal of Safety Science and Technology*, *4*(04), 166.‏

[24] Ramzan, M., Khan, H. U., Awan, S. M., Ismail, A., Ilyas, M., & Mahmood, A. (2019). A survey on state-of-the-art drowsiness detection techniques. *IEEE Access*, *7*, 61904-61919.‏

[25] Fernandes, M. J. C. N. (2019). Driver drowsiness detection using non-intrusive eletrocardiogram and steering wheel angle signals.‏

[26] Ouabida, E., Essadike, A., & Bouzid, A. (2020). Optical correlator based algorithm for driver drowsiness detection. *Optik*, *204*, 164102.‏

[27] Dlib-ml: A Machine Learning Toolkit July 2009Journal of Machine Learning Research 10:1755-1758

[28] Phan, A. C., Nguyen, N. H. Q., Trieu, T. N., & Phan, T. C. (2021). An efficient approach for detecting driver drowsiness based on deep learning. *Applied Sciences*, *11*(18), 8441.‏

[29] Sagonas, C., Tzimiropoulos, G., Zafeiriou, S., & Pantic, M. (2013). 300 faces in-the-wild challenge: The first facial landmark localization challenge. In *Proceedings of the IEEE international conference on computer vision workshops* (pp. 397-403).‏

[30] Soukupova, T., & Cech, J. (2016, February). Eye blink detection using facial landmarks. In *21st computer vision winter workshop, Rimske Toplice, Slovenia* (p. 2).‏

[31] driver drowsiness using keras. <https://www.kaggle.com/code/adinishad/driver-drowsiness-using-keras>

[32] Drowsiness dataset (Kaggle). <https://www.kaggle.com/datasets/dheerajperumandla/drowsiness-dataset>

[33] Yawn Dataset (Kaggle). <https://www.kaggle.com/datasets/davidvazquezcic/yawn-dataset>