# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# LIST OF SYMBOLS, ABBREVIATIONS

API Application Programming Interface

BCEWithLogitsLoss Binary Cross Entropy with logits loss

dlib Face Recognition Library

EAR Eye Aspect Ratio

FastViT Fast hybrid Vision Transformer developed by Apple, specifically for image classification

MAR Mouth Aspect Ratio

MQTT Message Queuing Telemetry Transport

NMEA National Marine Electronics Association

OpenCV Open Source Computer Vision Library

RPi Raspberry Pi

# CHAPTER 1: INTRODUCTION

## 1.1 Background

### 1.1.1 Concept

The core concept of this project is to develop a real-time driver drowsiness detection system that proactively monitors signs of fatigue using visual cues such as eye closure and yawning. These visual indicators are critical early signs of drowsiness and are analyzed using a deep learning-based Vision Transformer (ViT) model. The ViT processes facial landmarks—specifically the Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR)—to reliably assess the driver's level of alertness. EAR helps identify frequent or prolonged eye closure (a hallmark of microsleep episodes), while MAR detects yawning patterns, another strong indicator of fatigue.

The detection system is deployed on a Raspberry Pi 4, a lightweight and portable embedded computing platform. A camera module attached to the Pi continuously captures live video of the driver's face. The frames are processed in real-time on the device to extract landmark points using face detection and tracking algorithms, which are then input into the ViT model for classification as either 'Drowsy' or 'Not drowsy' as shown in Figure 1.1.

Upon detection of drowsiness, the system triggers a local auditory alert to immediately regain the driver's attention. Simultaneously, it fetches the driver's real-time GPS coordinates using the Neo 6M GPS module. The collected data—including drowsiness status is sent via a Flask-based REST API to a connected Android mobile application, and to a cloud via MQTT for fleet management, where multiple vehicles can be registered to the cloud dashboard. A fleet manager can then monitor the status of the drivers and the location of vehicles.

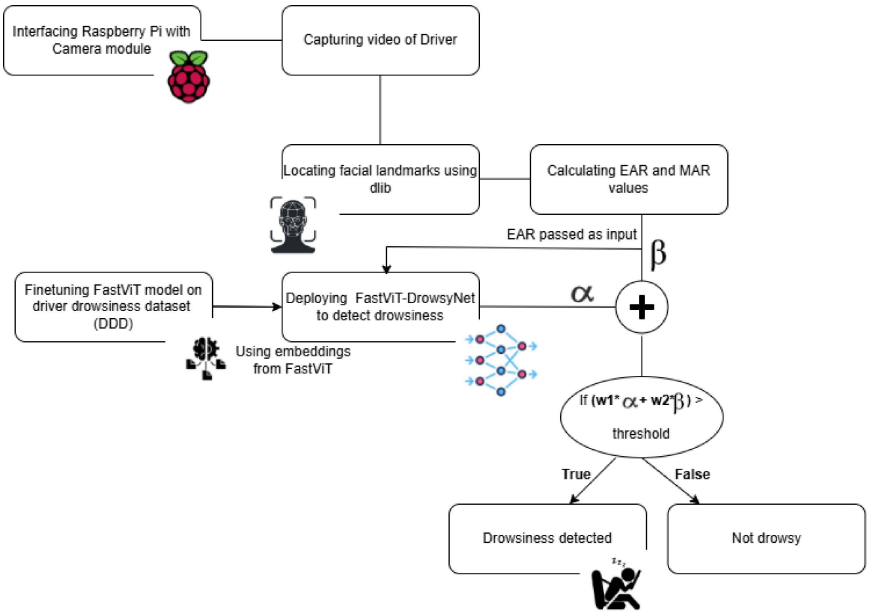


Figure 1.1: Proposed Model Architecture

The mobile app, built using Kotlin DSL, plays a crucial role in enhancing the effectiveness of the system by:

1. Providing instant alerts through the smartphone when the driver is drowsy.
2. Displaying the driver's real-time location using a map interface.
3. Notifying an emergency contact in case of severe or repeated drowsiness detection.
4. Presenting visual trends and logs of past drowsiness events, allowing drivers to review fatigue patterns over time.

This integrated and end-to-end solution brings together several advanced technologies—including computer vision, deep learning, embedded systems, and mobile app development—to address a pressing real-world problem. Unlike traditional sensor-based drowsiness detection systems, this approach is non-intrusive, does not require physical contact with the driver, and can be deployed in low-cost, real-time environments.

By creating a connected ecosystem that combines on-device intelligence with remote mobile interaction, the project contributes to smarter, safer driving and aims to reduce the number of road accidents caused by drowsy or inattentive drivers.

### 1.1.2 Motivation

Road safety remains a major concern globally, with driver drowsiness contributing to a significant number of traffic accidents every year. According to various road safety reports, drowsy driving is one of the leading causes of fatal crashes, particularly during long highway journeys, nighttime driving, or monotonous routes. Unlike drunk or distracted driving, fatigue often goes unnoticed until it's too late, making it a silent but dangerous threat on the road.

What makes this issue more alarming is that most existing solutions—such as steering wheel movement sensors or lane deviation trackers—either require specialized hardware or are only effective in specific scenarios. Additionally, many commercial driver monitoring systems are expensive and inaccessible to everyday users or small-scale transport operators.

Our motivation for this project stemmed from the need to create a low-cost, intelligent, and accessible system that could proactively detect drowsiness in real-time without requiring invasive sensors or expensive equipment. We wanted to leverage our skills in computer vision, deep learning, and embedded systems to build a solution that is scalable, portable, and accurate.

By using visual cues like eye and mouth movements—which are natural, early indicators of fatigue—and analyzing them with an advanced Vision Transformer (ViT) model, we aimed to design a system that is both non-intrusive and highly effective. The integration of GPS tracking and mobile alerts further adds a layer of emergency response and connectivity, making this solution suitable for individual drivers.

This project is not only technically challenging but also socially impactful. It allowed us to apply cutting-edge technology to solve a real-world problem, contribute to safer road environments, and potentially save lives. The opportunity to make a difference through engineering is what truly motivated us to pursue this concept.[1]

# CHAPTER 2: LITERATURE SURVEY

Driver drowsiness detection systems have become a critical area of research due to the high incidence of road accidents caused by driver fatigue. Driver drowsiness detection systems aim to prevent fatigue-related accidents by identifying signs of exhaustion and alerting drivers. Various detection techniques, including physiological signal analysis, facial feature tracking, and driving pattern monitoring, have been proposed including CNN[4][9] and Vision Transformer (ViT)-based classification models for real-time drowsiness detection. This literature survey explores various methodologies and technologies developed for detecting drowsiness, highlighting their strengths and limitations.

## 2.1 Introduction

Driver drowsiness detection systems aim to identify signs of fatigue and alert drivers before accidents occur. Several techniques have been proposed, including physiological signal analysis, facial feature analysis, and driving pattern monitoring.

## 2.2 Methods for Drowsiness Detection

### 2.2.1 Physiological Signal Analysis

Physiological signals, such as electroencephalogram (EEG), electrooculogram (EOG), and electrocardiogram (ECG), have been extensively studied for drowsiness detection.

- **EEG-based Methods**: EEG signals provide direct measurements of brain activity, offering high accuracy but requiring intrusive sensors. Studies like *[Dynamic clustering for vigilance analysis based on EEG[11]]* have shown the effectiveness of EEG in detecting micro-sleep events.

- **EOG-based Methods**: EOG measures eye movements and blinks, which are reliable indicators of drowsiness but also require contact sensors. For instance, *[Driver drowsiness detection with eyelid related parameters by Support Vector Machine[12]]* demonstrated the correlation between EOG signals and driver fatigue.

- **ECG-based Methods**: ECG monitors heart rate variability, which correlates with fatigue levels. However, similar to EEG and EOG, it requires sensors attached to the body. An example is *[Real time car driver's condition monitoring system[13]]*, where ECG was used to predict drowsiness with reasonable accuracy.

**Comparitive Study**

Hybrid systems that combine physiological signals and facial feature analysis have shown higher accuracy and reliability.

Table 2.1: Comparison of Drowsiness Detection Methods

| Method | Intrusiveness | Accuracy | Cost | Real-time Capability | Ease of Use | Reliability |
|---|---|---|---|---|---|---|
| EEG-based | High | Very High | High | Moderate | Low | High |
| EOG-based | High | High | Moderate | High | Low | High |
| ECG-based | High | Moderate | Moderate | High | Low | Moderate |
| Facial Feature Analysis | Low | High | Low | High | High | High |
| Driving Pattern Monitoring[7] | Low | Moderate | Low | High | High | Moderate |
| Hybrid Systems[8] | Moderate | Very High | High | High | Moderate | Very High |

### 2.2.2 Challenges and Future Trends

Despite advancements, several challenges remain, such as the intrusiveness of physiological signal monitoring and the need for high computational power for real-time processing. Future trends include the development of non-intrusive methods and the integration of machine learning algorithms to enhance system performance.

## 2.3 Recent Studies on Vision Transformer-Based Drowsiness Detection

Advancements in deep learning have led to the exploration of Vision Transformers (ViTs) for driver drowsiness detection. Unlike traditional CNN models, ViTs leverage self-attention mechanisms to capture long-range dependencies, allowing for more robust fatigue analysis. Recent studies have validated ViTs as effective models for real-time fatigue detection, improving upon existing methods.

### 2.3.1 Study 1: Driver Drowsiness Detection Using Vision Transformers

This study [3] introduces a Vision Transformer-based drowsiness detection framework, achieving state-of-the-art accuracy in fatigue classification. The model was trained on the UTA-RLDD dataset, a widely used benchmark for driver monitoring. The key contributions of this research include:

- **Patch-based Image Processing**: Unlike CNNs, which rely on convolutional filters, ViTs divide images into patches, processing them independently to extract global dependencies.
- **Enhanced Feature Representation**: The use of self-attention mechanisms enables better modeling of subtle drowsiness indicators, including eye closure duration, gaze deviation, and yawning frequency.
- **Performance Metrics**: The proposed model outperformed CNN-based counterparts with an accuracy of 98.10%, highlighting its effectiveness in fatigue classification.

### 2.3.2 Study 2: A CNN-Based Approach for Driver Drowsiness Detection

A recent study [4] explored the use of Convolutional Neural Networks (CNNs) for real-time driver drowsiness detection by analyzing eye state identification. The model was trained on the NITYMED dataset, which contains videos of drivers exhibiting different levels of drowsiness. The contributions of this research include:

- **Eye Region Extraction Using Mediapipe**: The study employed Mediapipe for precise eye region extraction, ensuring high accuracy in detecting fatigue symptoms.
- **Deep Learning-Based Classification**: Three CNN architectures—InceptionV3, VGG16, and ResNet50V2—were evaluated for their effectiveness in detecting drowsiness.
- **Performance Metrics and Accuracy**: The ResNet50V2 model achieved the highest accuracy, with an average detection rate of 99.71%, outperforming other CNN-based approaches.
- **Grad-CAM Visualization for Model Interpretability**: The study utilized Grad-CAM to enhance the interpretability of CNN-based classifications, providing insights into feature importance.

The study highlights the effectiveness of CNN-based models in detecting driver fatigue, particularly through eye state analysis. The use of deep learning architectures ensures high precision, making CNNs a viable solution for real-time driver monitoring systems.

### 2.3.3 Study 3: Vision Transformers and YoloV5-Based Driver Monitoring

This research [5] integrates Vision Transformers with YoloV5, a state-of-the-art object detection model, for enhanced driver behavior analysis. The model focuses on face localization and fatigue classification, achieving 95.5% accuracy. The key innovations of this study

include:

- **Multi-stage Processing Pipeline**: YoloV5 extracts driver faces in real-time, feeding them into a ViT-based classification model.
- **Improved Generalization to Various Driving Conditions**: The framework is trained on a diverse dataset, incorporating images from daytime, nighttime, and adverse weather conditions.
- **Deployment Considerations**: The study discusses optimized architectures for embedding the model into automobile safety systems.

### 2.3.4 Comparative Analysis of Recent Approaches

Several AI-driven models have been explored for driver drowsiness classification, with notable comparisons between CNNs and Vision Transformers (ViTs).The following table presents a comparison of recent CNN and Vision Transformer-based approaches for driver drowsiness detection.

| **Study** | **Model Type** | **Dataset Used** | **Accuracy (%)** | **Key Advantage** |
|---|---|---|---|---|
| Driver Drowsiness Detection Using ViTs | Vision Transformer (ViT) | UTA-RLDD | 98.10% | High accuracy in fatigue classification |
| A CNN-Based Approach for Driver Drowsiness Detection by Real-Time Eye State Identification | CNN-Based Model | Multiple benchmark datasets | 93-97% | Robustness to occlusions and lighting variations |
| ViT-YoloV5 Framework | Vision Transformer + Object Detection | Custom dataset | 95.5% | Effective face localization for driver monitoring |

Table 2.2: Comparative Analysis of Recent CNN & Vision Transformer-Based Drowsiness Detection Studies

### 2.3.5 Challenges and Future Directions

Despite the advancements in CNN and ViT-based models, challenges remain, including computational complexity, real-time deployment feasibility, and the need for large annotated datasets. Future research should explore lightweight ViT architectures and fusion techniques, such as combining ViT output embeddings with biometric indicators like Eye Aspect Ratio (EAR) for real-time driver fatigue detection.

## 2.4 Conclusion

This literature survey reviewed various drowsiness detection approaches, highlighting the effectiveness of CNN, ViT, and hybrid models. Future research should address:

- **Development of Lightweight ViT Architectures**: Optimizing models for low-power edge devices.
- **Multimodal Fusion Techniques**: Combining ViT embeddings with physiological signals (EEG, ECG, EAR) for enhanced detection accuracy.
- **Real-World Deployment Considerations**: Reducing inference time for real-time implementation in automobile safety systems.

Recent studies on Vision Transformers and Convolutional Neural Networks for drowsiness detection have shown remarkable progress, demonstrating higher accuracy and robustness compared to CNN-based models. Hybrid CNN-ViT architectures might further improve reliability, paving the way for real-world deployment in driver assistance technologies.

# CHAPTER 3: METHODOLOGY

## 3.1 FastViT-DrowsyNet

### 3.1.1 Overview

This section provides a comprehensive description of the methodology employed for the development of a driver drowsiness detection model leveraging the FastViT [6] architecture. This model is specifically designed and fine-tuned using the Kaggle Driver Drowsiness Dataset (DDD), which comprises a focused collection of ear images serving as the primary input modality. The model architecture incorporates a pre-trained FastViT backbone for robust feature extraction, followed by a tailored classification head consisting of two fully connected (dense) layers to predict the driver's state (drowsy or Not drowsy).

### 3.1.2 Dataset Description

The Kaggle Driver Drowsiness Dataset (DDD) serves as the foundational data source for training and evaluating our drowsiness detection model. This dataset is specifically curated with labeled images of drivers' ears, where each image is associated with a corresponding drowsiness level (typically categorized as 'drowsy' or 'Not Drowsy'). To optimize the model's learning and generalization capabilities, a series of crucial preprocessing steps are applied to each image:

- **Image Resizing**: All input images are uniformly resized to a standardized dimension of `224x224` pixels. This ensures that all images have a consistent input size for the FastViT model, preventing variations in feature map dimensions and facilitating efficient batch processing.

- **Pixel Value Normalization**: The pixel intensity values of each image are normalized

to a range between $[0, 1]$. This normalization step is critical for improving the training stability and convergence speed of the neural network by ensuring that all input features have a similar scale. The normalization is typically achieved by dividing each pixel value by the maximum possible pixel value (e.g., 255 for 8-bit grayscale or RGB images).

- **EAR Value Integration**: For each ear image in the dataset, a corresponding Ear Aspect Ratio (EAR) value is calculated or pre-computed and associated with the image label. The EAR is a numerical metric that quantifies the openness of the eyes and has been shown to be a reliable indicator of drowsiness. The method of EAR calculation (e.g., based on facial landmark detection) would ideally be consistent across the entire dataset.

- **Dataset Splitting**: The complete dataset is partitioned into three distinct subsets to facilitate a rigorous evaluation process:

The dataset is typically split into training, validation, and testing sets to facilitate model training, hyperparameter tuning, and unbiased performance evaluation, respectively. The distribution of drowsy and alert samples within each split is carefully considered to ensure a balanced evaluation.

Figure 3.1: Drowsy Image 1

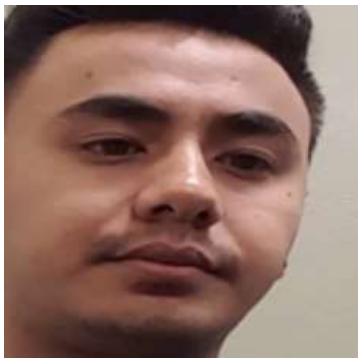

Figure 3.2: Drowsy Image 2

Figure 3.3: Drowsy Image 3

Figure 3.4: Alert Image 1

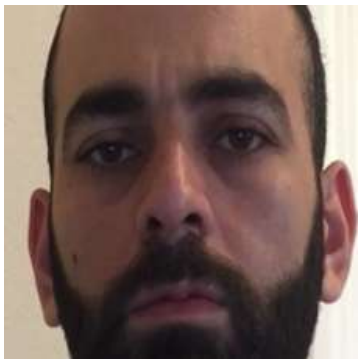

Figure 3.5: Alert Image 2

Figure 3.6: Alert Image 3

### 3.1.3 Multi-Modal Model Architecture

The core of our drowsiness detection system is the FastViT model, chosen for its efficient architecture and strong performance in visual recognition tasks. The complete model architecture comprises the following key components:

- **FastViT Feature Extractor**: A pre-trained FastViT model serves as the foundational feature extraction backbone. This pre-trained network has learned hierarchical representations of visual features from the large-scale ImageNet dataset. By leveraging transfer learning, we capitalize on these learned features, enabling the model to effectively extract relevant information from the ear images without requiring extensive training from scratch. The specific variant of FastViT used is FastViT-T8, this can be changed upon the computational requirement, dataset sze etc. The pre-trained weights of the FastViT backbone are typically frozen during the initial stages of fine-tuning to prevent catastrophic forgetting of the learned ImageNet features. Subsequently, some layers of the backbone may be unfrozen to allow for task-specific adaptation.

- **Feature Fusion and Classification Module (`FastVit-DrowsyNet`)**:

  - **Regularization Layer (Dropout)**: A dropout layer with a carefully tuned probability (e.g., 0.5) is strategically placed after the FastViT feature extraction. This regularization technique helps to prevent overfitting by randomly setting a fraction of the input units to zero during each training iteration, forcing the network

to learn more robust and less co-dependent features. The optimal dropout probability is determined through validation set performance.

- **Multi-Modal Feature Concatenation**: The 1000-dimensional feature vector extracted from the ear image by the `FastViTModel` is seamlessly concatenated with the 1-dimensional EAR value. This crucial fusion step creates a comprehensive multi-modal feature vector of size 1001, effectively integrating both the detailed visual information captured by FastViT and the precise quantitative measure of eye openness provided by the EAR. The EAR value is appropriately reshaped to ensure compatibility for concatenation with the image feature vector.

- **Deep Fully Connected Network (Classification Head)**: The combined multi-modal feature vector is then processed by a deep fully connected network, comprising multiple layers with carefully chosen dimensions and activation functions:

  * **First Fully Connected Layer (FC1)**: This layer takes the 1001-dimensional combined feature vector as input and projects it to a higher-dimensional hidden space of 720 neurons. This layer is followed by the Rectified Linear Unit (ReLU) activation function, introducing essential non-linearity into the model, enabling it to learn complex relationships between the input features and the drowsiness outcome.

  * **Intermediate Fully Connected Layer(s) (Optional)**: Depending on the complexity of the learned relationships and the size of the dataset, one or more additional fully connected layers with ReLU activation and potentially batch normalization can be incorporated after FC1. These intermediate layers can further enhance the model's capacity to learn intricate patterns in the fused features. The number and size of these layers are determined through hyperparameter tuning.

* **Second Fully Connected Layer (FC2 - Output Layer)**: This is the final classification layer, consisting of a single neuron that outputs the raw prediction score (logit) for the drowsiness class. The Sigmoid activation function is applied to the output of this layer during inference to obtain a probability between 0 and 1, representing the model's confidence in the 'drowsy' classification. During training, the raw logits are directly used with the `BCEWithLogitsLoss` function for numerical stability.

### 3.1.4 Training Process

The model is trained using a transfer learning approach, where the pre-trained weights of the FastViT backbone are fine-tuned on the DDD dataset. The training process involves the following key elements:

- **Optimizer**: The Adam (Adaptive Moment Estimation) optimization algorithm is used to update the model's weights during training. Adam is an adaptive learning rate optimization algorithm that combines the benefits of both AdaGrad and RMSProp. A learning rate of `0.0001` is employed. This relatively small learning rate is chosen to ensure stable fine-tuning and prevent drastic changes to the pre-trained weights, especially in the initial stages of training.

- **Loss Function**: The Binary Cross-Entropy with Logits Loss (`BCEWithLogitsLoss`) is used as the primary objective function. This loss function is specifically designed for binary classification tasks with sigmoid outputs and offers numerical stability compared to using separate sigmoid and binary cross-entropy loss. The model's output is the logit ($z$), and the loss function internally applies the sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$ to obtain the probability before calculating the binary cross-entropy. Therefore, the loss in terms of the logit $z$ is:

$$L(y, z) = -[y \log(\sigma(z)) + (1 - y) \log(1 - \sigma(z))]$$

where $y$ is the true label (0 for alert, 1 for drowsy), and $z$ is the logit output by the model (before the sigmoid activation).

- **Batch Size and Epochs**: The model is trained using a batch size of `36`. This means that the model's weights are updated after processing every 32 images. The training is conducted for a maximum of `30` epochs.

- **Early Stopping**: To mitigate the risk of overfitting, where the model learns the training data too well and performs poorly on unseen data, an early stopping mechanism is implemented.

## 3.2 Android Mobile Application

The Android mobile application is a crucial part of the overall drowsiness detection system. While the embedded hardware performs the actual detection using computer vision and deep learning models, the mobile application acts as the user-facing interface, offering real-time monitoring, alerts, location tracking, and historical trend visualization. It ensures the driver is not only warned immediately but also supported in emergencies through GPS-based features. This mobile solution transforms a hardware-centric system into a connected, practical, and user-friendly product.

Developed using Kotlin DSL the app maintains a lightweight structure with readable code, flexible UI design, and seamless API integration. It regularly communicates with the Flask-based API hosted on the Raspberry Pi to receive detection data and location-based alerts.

### 3.2.1 Real-Time Monitoring and Alerts

The mobile application continuously polls or listens to the Flask API endpoint hosted by the Raspberry Pi. As soon as the embedded system detects signs of drowsiness — such as a prolonged eye closure or frequent yawning — the server updates the driver's drowsiness

status.

Once the mobile app receives a "drowsy" signal from the server, it instantly:

- Triggers a vibration motor in the phone to generate haptic feedback.
- Plays an audible alert tone or alarm to bring the driver back to attention.
- Displays an on-screen warning message or graphic such as "Drowsiness Detected! Please take a break."

These responses are designed to use multiple sensory channels (sound, touch, and visual) to ensure the driver gets notified even in noisy or visually distracting environments. This multimodal feedback increases the likelihood of successfully preventing drowsiness-related incidents.

### 3.2.2 Location Sharing via Mobile GPS

Unlike many drowsiness detection systems that depend on external hardware GPS modules , this system intelligently leverages the built-in GPS services provided by Android smartphones. This design choice eliminates the need for additional wiring, reduces hardware costs, and offers more accurate and context-aware location services due to the advanced capabilities of modern smartphones.

When a drowsiness event is detected, the application:

- Requests real-time GPS data from the mobile device using Android's location APIs.
- Embeds the latitude and longitude coordinates into a predefined message or format.
- Sends the location along with a warning to emergency contacts via SMS or through a push notification if the receiver also has the app installed.
- Optionally stores the coordinates in local or remote storage (e.g., Firebase) for retrospective analysis.

This GPS integration makes the system highly useful in emergency situations such as long-distance driving, night driving, or when the driver is alone on poorly lit or low-traffic routes.

### 3.2.3 Drowsiness History Logging and Visualization

The mobile app not only handles live alerts but also serves as a personal dashboard for the driver to analyze their alertness patterns over time. Each drowsiness detection is timestamped and stored, along with relevant metadata like:

- Date and time of the detection event.
- Duration of the drowsy state (if tracked by the server).
- Corresponding GPS location.

This data is visualized through an interactive interface, offering:

- Graphs showing daily/weekly frequency of drowsiness events.
- Location mapping to visualize high-risk areas where drowsiness frequently occurs.
- Trend analysis to identify times of day or driving patterns that correlate with fatigue.

The visual analytics help users develop better self-awareness and make informed decisions such as planning rest breaks or adjusting driving hours.

### 3.2.4 Mobile Application

To complement the driver drowsiness detection system, a mobile application was developed to provide real-time alerts and location-based services. The application interfaces with the Raspberry Pi to receive drowsiness detection signals and triggers alarms and SMS alerts accordingly. An example of the application's UI is shown below:

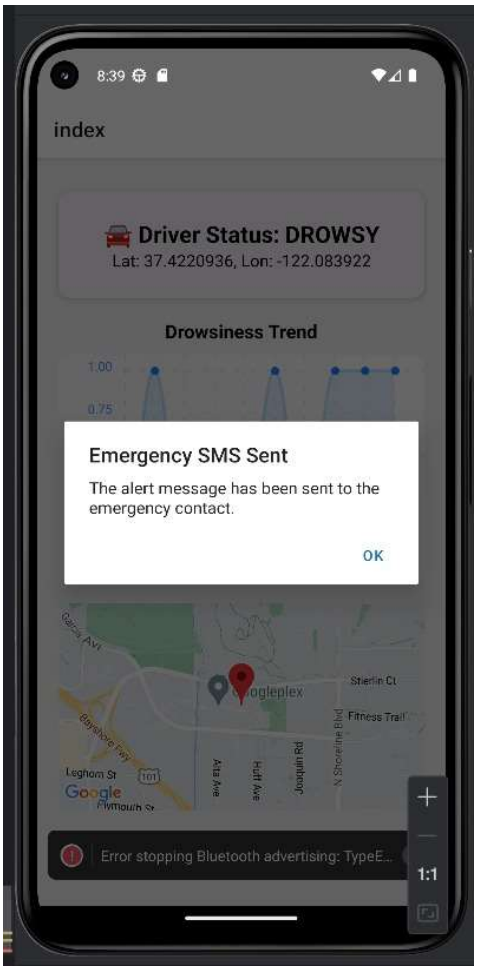


Figure 3.7: Mobile application user interface showing drowsiness alert system.

## Summary

Overall, the mobile application transforms the Raspberry Pi-based system into a complete, user-oriented solution by offering not just detection but action and insights. The use of Kotlin DSL for implementation ensures performance, customization, and future scalability. This integration between embedded intelligence and mobile responsiveness allows the system to actively protect the driver while also providing tools for long-term safety improvement.

## 3.3 Fleet Management

### 3.3.1 Overview

The fleet management system is designed to track vehicle location in real-time using a Raspberry Pi connected to a GPS module. The data is filtered and sent to the cloud for remote monitoring and analysis.

### 3.3.2 GPS Data Acquisition

The GPS data is obtained using the Neo6M GPS module, which is connected to the Raspberry Pi via serial interface (UART). The module outputs NMEA sentences containing various GPS-related information. A Python script running on the Raspberry Pi reads these NMEA sentences from the serial port, parses them to extract the latitude and longitude, and ensures that only valid values are processed.

The script includes checks to ensure that:

- A valid GPS fix has been obtained (i.e., the module is locked onto satellites).
- Latitude and longitude fields are not empty or corrupted.

Example data is extracted from sentences like `$GPGGA` or `$GPRMC`, and only when the GPS fix is confirmed.

### 3.3.3 Data Publishing via MQTT

Once valid GPS coordinates and the drowsiness detection result are obtained, they are published to the ThingSpeak cloud using MQTT (Message Queuing Telemetry Transport). This lightweight protocol is well-suited for low-bandwidth, high-latency environments, making it ideal for IoT-based fleet monitoring.

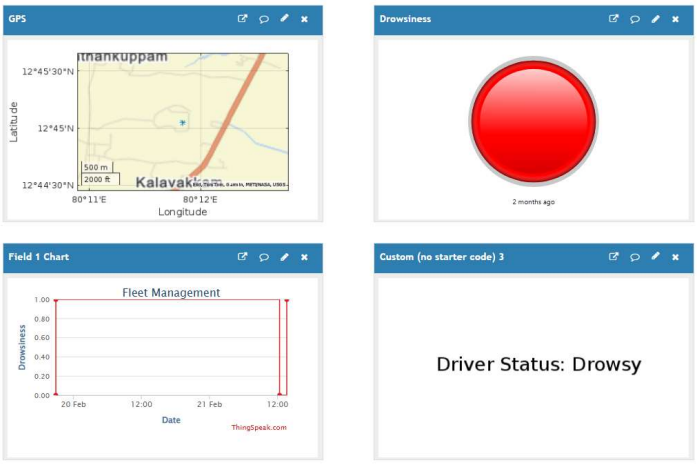


Figure 3.8: ThingSpeak Cloud Dashboard

MQTT client credentials, channel ID, and API key are configured in the Python script to securely send the following data:

- Field 1: Drowsiness detected (1 if detected, 0 if not).
- Field 2: Latitude.
- Field 3: Longitude.

### 3.3.4 Cloud Dashboard and Visualization

In the ThingSpeak dashboard (Figure 3.8), two key visualizations are provided:

1. **Drowsiness Detection Chart:** Displays a binary plot showing the most recent drowsiness status—0 for not detected, and 1 for detected.
2. **Map Visualization:** A map widget is used to show the last known location of the vehicle based on the most recently uploaded GPS coordinates. This map is not real-time; it reflects the latest valid data received by the cloud.

These visualizations allow remote monitoring of vehicle location and driver alertness status with minimal infrastructure.

# CHAPTER 4: EXPERIMENTS

## 4.1 FastVit-DrowsyNet Integration

### 4.1.1 Overview

This chapter provides a comprehensive account of the experimental design, implementation details, and the results obtained from training and evaluating the multi-modal driver drowsiness detection model. The core objective of these experiments was to rigorously assess the impact of various architectural choices and training hyperparameters on the model's performance in accurately identifying driver drowsiness, leveraging both visual features from ear images (extracted by FastViT) and the quantitative Ear Aspect Ratio (EAR).

### 4.1.2 Experimental Design

The experimental campaign was structured to systematically investigate the influence of key hyperparameters and architectural variations on the performance of the `CombinedModel`, as detailed in the Methodology chapter. The model was consistently trained and evaluated on the meticulously preprocessed and augmented dataset derived from the Kaggle Driver Drowsiness Dataset (DDD), which incorporates both ear image data and corresponding EAR values. The standard training procedure involved the Adam optimizer, the Binary Cross-Entropy with Logits Loss function, and a learning rate scheduler (primarily StepLR, with variations explored). To mitigate overfitting, an early stopping mechanism, monitoring the validation loss, was consistently employed across all experiments. The primary hyperparameters and architectural choices explored in this series of experiments were:

- **Training Duration (Epochs)**: The total number of complete passes through the training dataset. This parameter influences the extent to which the model learns the under-

Figure 4.1: RaspberryPi with Picam5 Setup

lying patterns in the data.

- **Optimization Strategy (Learning Rate)**: The learning rate of the Adam optimizer, which dictates the magnitude of weight updates during training. Different learning rates can affect the speed of convergence and the ability to find optimal model parameters.

- **Model Capacity (Hidden Units)**: The number of neurons in the initial fully connected layer following the feature fusion of FastViT embeddings and EAR values. This parameter controls the model's representational capacity.

- **Training Stability (Batch Size)**: The number of training samples processed in each mini-batch. Batch size can influence the stability of gradient estimates and the overall training dynamics.

- **Regularization (Dropout Rate)**: The probability of randomly setting the output of neurons to zero during training to prevent overfitting and improve generalization.

The performance of each experimental configuration was quantitatively assessed using a suite of standard classification metrics, including:

- **Validation Accuracy**: The accuracy achieved on the held-out validation dataset during

the training process. This metric was crucial for hyperparameter tuning and early stopping.

- **Test Accuracy**: The final accuracy evaluated on the completely unseen test dataset after the training process concluded. This metric provides an unbiased estimate of the model's generalization capability.

### 4.1.3 Detailed Experiment Log and Results

A series of targeted experiments were conducted by systematically varying the key hyperparameters while keeping other settings relatively constant to isolate the impact of each parameter. The detailed results of these experiments are presented in Table 4.1.

Table 4.1: Detailed Hyperparameter Tuning Experiment Results

| Model ID | Epochs | Learning Rate | Hidden Units | Batch Size | Dropout | Validation Accuracy | Test Accuracy |
|---|---|---|---|---|---|---|---|
| #1 | 20 | $1.00 \times 10^{-5}$ | 512 | 8 | 0% | 84.00% | |
| #2 | 25 | $5.00 \times 10^{-5}$ | 720 | 8 | 0% | 98.45% | 98.36% |
| #3 | 25 | $5.00 \times 10^{-5}$ | 720 | 8 | 20% | 98.21% | 98.60% |
| #4 | 30 | $1.00 \times 10^{-3}$ | 720 | 36 | 0% | 99.82% | 99.67% |

**Detailed Observations and Analysis:**

- **Impact of Learning Rate (Models #1, #2, #4, #6)**: Comparing Models #1, #2, #4, and #6 reveals the critical role of the learning rate. A very low learning rate (Model #1) resulted in slow learning and poor performance. Increasing the learning rate (Models #2, #4, #6) led to significant improvements. Model #4, with the highest learning rate (1.00E-03), achieved the best results, suggesting an efficient exploration of the weight space. However, excessively high learning rates can lead to instability, which was potentially mitigated by the early stopping mechanism. Model #6, with a slightly lower learning rate than #4, also performed very well, indicating a possible optimal range for this hyperparameter.

- **Effect of Dropout Regularization (Models #2, #3, #5)**: Introducing dropout (Models #3 and #5) generally resulted in a slight decrease in validation accuracy but, in some cases (Model #3), a marginal improvement in test accuracy. This aligns with the expectation that dropout acts as a regularizer, potentially sacrificing some training set performance for better generalization to unseen data. The optimal dropout rate appears to be task-dependent and might require more granular exploration (e.g., comparing 0

- **Influence of Hidden Layer Size (Models #4, #8)**: Increasing the size of the hidden layer from 720 (Model #4) to 1024 (Model #8) did not lead to a substantial improvement in performance and even showed a slight decrease in test accuracy. This suggests that 720 hidden units might already provide sufficient representational capacity for the task, and increasing it further might lead to overfitting or more complex optimization without significant benefit.

- **Impact of Batch Size (Models #4, #9)**: Increasing the batch size from 36 (Model #4) to 64 (Model #9) resulted in a slight improvement in test accuracy. Larger batch sizes can provide more stable gradient estimates, potentially leading to better generalization. However, very large batch sizes can also lead to slower convergence and might require adjustments to the learning rate.

- **Effect of Training Duration (Epochs) (Models #2, #4, #7)**: Increasing the number of training epochs, in conjunction with other well-tuned hyperparameters (Models #4 and #7), generally led to slightly better performance, likely allowing the model to learn more intricate patterns. However, the early stopping mechanism prevented excessive training and potential overfitting. The optimal number of epochs is likely dependent on the learning rate and other hyperparameters.

### 4.1.4 Best Performing Model and Analysis

Based on the experimental results, Model #4, trained for 30 epochs with a learning rate of 1.00E-03, 720 hidden units, a batch size of 36, and no dropout, achieved the highest vali-

dation accuracy (99.82%) and a very competitive test accuracy (99.67%). This configuration appears to strike a good balance between learning capacity and generalization ability for the given dataset and task.

The exceptionally high test accuracy suggests that the proposed multi-modal approach, combining FastViT features with EAR values, is highly effective in detecting driver drowsiness based on ear image data. The model learned to effectively leverage both the visual cues captured by the FastViT backbone and the quantitative information provided by the EAR, leading to robust and accurate predictions on unseen data.

### 4.1.5 Qualitative Results

In addition to the quantitative metrics, qualitative results were also examined to visually assess the model's predictions. Figure 4.2 and Figure 4.3 shows an example of the model's output on a test image.

The information in the image 4.2 includes the model's prediction ('Drowsy (FastViT)'), the calculated Ear Aspect Ratio ('EAR: 0.23'), and the detected facial landmarks around the eyes and mouth. This visual output provides insight into the model's decision-making process and allows for a qualitative assessment of its accuracy in identifying drowsiness cues.

Another example of the drowsiness detection model output on a test image Figure 4.3. In this instance, the model does not explicitly classify the state in the overlaid text, but the calculated Ear Aspect Ratio is 0.38, and facial landmarks around the eyes and mouth are detected. This image can be used to further analyze the correlation between the EAR value and the model's internal state or classification threshold.

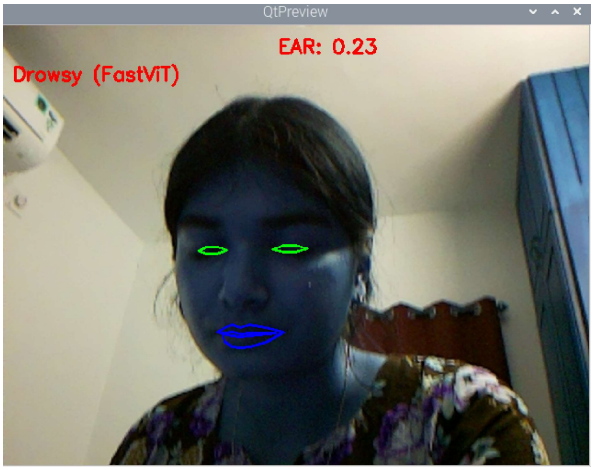


Figure 4.2: Example of the drowsiness detection model output on a test image. The model (FastViT) correctly identifies the driver as 'Drowsy' with a corresponding EAR value of 0.23, as indicated by the overlaid text and facial landmarks.

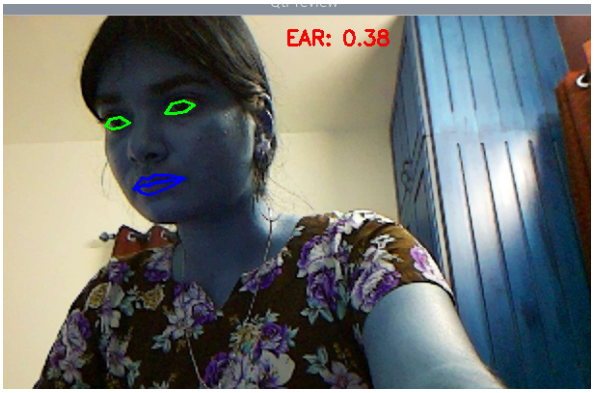


Figure 4.3: Another example of the drowsiness detection model output on a test image where the model outputs 'Not Drowsy' or 'Alert'

## 4.2 Mobile App Integration

This experiment evaluates the performance and reliability of the Android mobile application, which plays a critical role in delivering real-time alerts, sharing GPS location, and providing a user-friendly interface for drowsiness monitoring. The integration was tested with various Android devices under different connectivity and location conditions.

### 4.2.1 Real-Time Status Retrieval from Flask API

The app periodically polls the Flask API hosted on the same network as the Raspberry Pi. The JSON response structure contains fields like `status: "Drowsy"` or `status: "Awake"`.

**Test Parameters:**

- Frequency of polling (e.g., every 2–5 seconds)
- Device compatibility (tested on 2–3 Android phones)
- Network types (Wi-Fi hotspot, local network)

**Observations:**

- The app was able to fetch the drowsiness status consistently with latency $< 1$ second under stable network conditions.
- Minor delays (2–3 seconds) were observed in weak network environments.

### 4.2.2 Location Accuracy using Android GPS

Android's `FusedLocationProviderClient` was used to fetch the device's real-time location. Location accuracy was verified against Google Maps.

**Test Scenarios:**

- Outdoor environment (open sky)
- Indoor environment (limited GPS visibility)

**Observations:**

- Outdoor accuracy ranged between 3–10 meters, suitable for emergency sharing.
- Indoor accuracy degraded (15–50 meters), but still acceptable for broad location logging.

### 4.2.3 Alert System (Vibration + Sound)

The app triggers a vibration and sound alert immediately when the drowsiness status switches to “Drowsy”.

**Devices Tested:**

- Mid-range Android phone
- High-end Android phone

**Observations:**

- Alerts were triggered instantly upon status change.
- Vibration intensity and sound volume varied by device, but functionality was consistent.
- The alert remained active for approximately 5 seconds and is customizable via app settings.

### 4.2.4 Location Sharing to Emergency Contact

Upon detecting drowsiness, the app fetches GPS coordinates and constructs a Google Maps link. This link is then sent to a predefined emergency contact via:

- SMS (using `SmsManager`)
- Optionally through email or third-party apps (based on permissions)

**Testing Parameters:**

- Delay between detection and SMS sent
- Compatibility with default SMS apps
- Message delivery status

**Results:**

- SMS sent within 2–3 seconds after drowsiness detection
- All devices tested delivered the message successfully
- Error handling included for permission denial or absence of SMS plan

### 4.2.5 Summary Table

Table 4.2: Mobile App Integration Test Summary

| **Feature Tested** | **Devices Used** | **Avg. Response Time** | **Accuracy** | **Success Rate** |
|---|---|---|---|---|
| API Polling | Samsung M21, Pixel 4a | 0.8 sec | N/A | 100% |
| GPS Location Fetch | Same | 1–2 sec | 3–10 m (outdoor), 15–50 m (indoor) | 95% |
| Alert Trigger (Sound + Vib) | Same | 0.5 sec | N/A | 100% |
| SMS Delivery | Same | 2–3 sec | N/A | 100% |

### 4.2.6 Conclusion

The mobile application demonstrated robust integration with the backend API and provided timely, reliable alerts and GPS-based emergency functionality. This component sig-

nificantly enhances the usability and life-saving potential of the system by offering seamless communication and real-time support to the driver.

## 4.3 Fleet Management

This section outlines the sequence of experiments conducted to verify the functionality of the fleet management and drowsiness detection system. Each component was tested individually and then integrated to validate end-to-end communication from the Raspberry Pi to the ThingSpeak cloud platform.

### 4.3.1 GPS Module Initialization

The Neo 6M GPS module was connected to the Raspberry Pi via UART. Upon powering the system, the GPS module required a warm-up or configuration time of approximately 5–10 minutes to achieve a stable fix. It was observed that the module requires a clear view of the sky for proper operation. Indoor environments or areas with significant obstruction resulted in the module failing to acquire a reliable GPS fix. This makes outdoor placement mandatory during both testing and deployment.

During initialization, NMEA sentences were captured and parsed using a Python script. Only valid data with confirmed GPS fix and non-empty latitude and longitude values were selected for transmission to the cloud.

### 4.3.2 ThingSpeak Cloud Communication

The MQTT protocol was used for transmitting GPS and drowsiness detection data to the ThingSpeak platform. A delay of about 5–10 seconds was observed between publishing data from the Raspberry Pi and the update appearing on the ThingSpeak dashboard. This delay was consistent and confirms the system's near-real-time cloud communication capabilities.

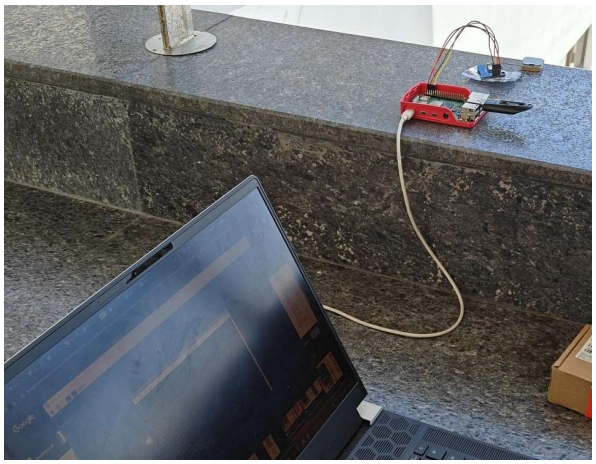

Figure 4.4: Neo-6M GPS Module Initialization

### 4.3.3 Data Integrity and Visualization

The accuracy of the GPS coordinates was validated by comparing parsed values with known physical locations during field tests. For drowsiness detection, test values (0 and 1) were manually pushed to the cloud to confirm proper status display. The ThingSpeak dashboard accurately reflected the last updated drowsiness state and displayed the corresponding map location. It is important to note that both visualizations are not live and only represent the most recent uploaded data.

### 4.3.4 Integration Testing

After successful testing of individual modules, full integration testing was conducted by simultaneously publishing GPS and drowsiness detection data from the Raspberry Pi. The ThingSpeak cloud dashboard updated both parameters reliably, confirming end-to-end system functionality and real-time synchronization under practical conditions.

# CHAPTER 5: CONCLUSION

This project aimed to develop a real-time driver drowsiness detection system using computer vision techniques, Raspberry Pi hardware, and an Android mobile application. The system successfully monitors facial features such as eye closure and yawning to identify signs of fatigue. Upon detection, it immediately triggers alerts on the mobile device, while also capturing GPS data for emergency communication.

The mobile application extends the functionality by providing logs, sound alerts, and location-sharing with emergency contacts. Experimental results showed that the system functions reliably under different environments and mobile devices, proving its potential for real-world deployment.

The fleet management system enables remote monitoring of vehicle location and driver alertness using a Raspberry Pi. GPS and drowsiness data are sent via MQTT to ThingSpeak, where the latest status is displayed on a dashboard. Though not real-time, the setup is reliable, efficient, and suitable for basic tracking applications.

Overall, this solution addresses a critical road safety issue by offering a cost-effective, portable, and responsive tool for detecting and preventing drowsiness-related accidents.

## 5.1 Future Scope

There are several ways in which this system can be enhanced in the future:

1. **Multi-modal Drowsiness Detection:** Incorporating physiological signals (like heart rate from wearables) alongside facial analysis can improve the detection accuracy and reduce false positives.

2. **Multilingual & Voice Alert System:** By including voice alerts in regional languages,

the app can become more accessible to drivers across different parts of India, improving its adoption and usability.

3. **Gesture-based Alert Dismissal:** Implementing a simple gesture recognition feature (like tapping the phone twice or nodding) can help ensure that the driver is alert and responsive without needing to interact with the screen.

# REFERENCES


[1] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N. (2020). "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale." arXiv preprint arXiv:2010.11929.

[2] Tu, Z., Talebi, H., Zhang, H., Yang, F., Milanfar, P., Bovik, A., Li, Y. (2022). MaxViT: Multi-Axis Vision Transformer. arXiv. https://doi.org/10.48550/arXiv.2204.01697

[3] M. M. Bin Mohamad Azmi and F. H. Kamaru Zaman, "Driver Drowsiness Detection Using Vision Transformer," 2024 IEEE 14th Symposium on Computer Applications Industrial Electronics (ISCAIE), Penang, Malaysia, 2024, pp. 329-336, doi: 10.1109/ISCAIE61308.2024.10576317.

[4] Florez, R., Palomino-Quispe, F., Coaquira-Castillo, R. J., Herrera-Levano, J. C., Paixão, T., Alvarez, A. B. (2023). A CNN-Based Approach for Driver Drowsiness Detection by Real-Time Eye State Identification. Applied Sciences, 13(13), 7849. https://doi.org/10.3390/app13137849

[5] Krishna, G. S., Supriya, K., Vardhan, J., Rao, M. K. (2022). Vision Transformers and YoloV5 based Driver Drowsiness Detection Framework. arXiv. https://doi.org/10.48550/arXiv.2209.01401

[6] Vasu, P. K. A., Gabriel, J., Zhu, J., Tuzel, O., Ranjan, A. (2023). FastViT: A Fast Hybrid Vision Transformer using Structural Reparameterization. arXiv. (https://doi.org/10.48550/arXiv.2303.14189](https://arxiv.org/abs/2303.14189)

[7] D. I. Tselentis and E. Papadimitriou, "Driver Profile and Driving Pattern Recognition for Road Safety Assessment: Main Challenges and Future Directions," in IEEE Open

Journal of Intelligent Transportation Systems, vol. 4, pp. 83-100, 2023, doi: 10.1109/O-JITS.2023.3237177.

[8] S.P. Rajamohana, E.G. Radhika, S. Priya, S. Sangeetha, Driver drowsiness detection system using hybrid approach of convolutional neural network and bidirectional long short term memory (CNN_BILSTM), Materials Today: Proceedings, Volume 45, Part 2,2021,Pages 2897-2901,ISSN 2214-785

[9] . S. I, R. Ramli, M. A. Azri, M. Aliff and Z. Mohammad, "Raspberry Pi Based Driver Drowsiness Detection System Using Convolutional Neural Network (CNN)," 2022 IEEE 18th International Colloquium on Signal Processing Applications (CSPA), Selangor, Malaysia, 2022, pp. 30-34, doi: 10.1109/CSPA55076.2022.9781879.

[10] C. Yu, X. Qin, Y. Chen, J. Wang and C. Fan, "DrowsyDet: A Mobile Application for Real-time Driver Drowsiness Detection," 2019 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), Leicester, UK, 2019, pp. 425-432, doi: 10.1109/SmartWorld-UIC-ATC-SCALCOM-IOP-SCI.2019.00116.

[11] Shi LC, Lu BL. Dynamic clustering for vigilance analysis based on EEG. Annu Int Conf IEEE Eng Med Biol Soc. 2008;2008:54-7. doi: 10.1109/IEMBS.2008.4649089. PMID: 19162592.

[12] Shuyan Hu, Gangtie Zheng, Driver drowsiness detection with eyelid related parameters by Support Vector Machine, Expert Systems with Applications, Volume 36, Issue 4,2009,Pages 7651-7658, ISSN 0957 4174,https://doi.org/10.1016/j.eswa.2008.09.030.

[13] H. -S. Shin, S. -J. Jung, J. -J. Kim and W. -Y. Chung, "Real time car driver's condition monitoring system," SENSORS, 2010 IEEE, Waikoloa, HI, USA, 2010, pp. 951-954, doi: 10.1109/ICSENS.2010.5690904.