



PRIFYSGOL CYMRU
Y Drindod Dewi Sant
UNIVERSITY OF WALES
Trinity Saint David
SWANSEA - ABERTAWE



School of
Applied Computing

DRIVER FATIGUE DETECTION SYSTEM BASED ON ANDROID DEVICES AND DEEP LEARNING MODELS

Hongye Yuan

2304718

Supervisor: Dr. Michael Dacey

Project submitted as part of the
requirements for the award of MSc:
Software Engineer and Artificial Intelligence

September 2024

Declaration of Originality

I, ...(Hongye Yuan)..... declare that I am the sole author of this Project; that all references cited have been consulted; that I have conducted all work of which this is a record, and that the finished work lies within the prescribed word limits.

This work has not previously been accepted as part of any other degree submission.

Signed :Hongye Yuan

Date :07/01/2024

FORM OF CONSENT

I Hongye Yuan hereby consent that my Project, submitted in candidature for the Master Degree , if successful, may be made available for inter-library loan or photocopying (subject to the law of copyright), and that the title and abstract may be made available to outside organisations.

Signed :Hongye Yuan.....

Date : 07/01/2024.....

Copyright Acknowledgement

I acknowledge that the copyright of this project report, and any product developed as part of the project, belong to University of Wales Trinity Saint David, Swansea.

ABSTRACT/SYNOPSIS

This project developed a driver fatigue monitoring system based on the Android platform, which integrates Single Shot MultiBox Detector (SSD) and MobileNetV3 deep learning models to achieve real-time, non-intrusive fatigue detection. Experimental validation on the YawDD dataset demonstrated that the system performs excellently in detecting driver fatigue, with a testing accuracy of 97.25%. The study first analyzed various fatigue detection methods and selected vision-based facial detection technology as the primary research focus to address the hardware limitations of older vehicles. The results show that the combination of SSD and MobileNetV3 offers significant advantages in processing real-time video streams and accurately identifying drivers' facial features. Additionally, by leveraging Android devices and their open development environment, this system effectively addresses the challenges of deploying complex detection systems in older vehicles, achieving low-cost, high-efficiency fatigue monitoring.

Table Of Content

SUPERVISOR: DR. SEENA JOSEPH	I
DECLARATION OF ORIGINALITY	II
FORM OF CONSENT	II
COPYRIGHT ACKNOWLEDGEMENT	III
ABSTRACT/SYNOPSIS	III
CHAPTER 1 INTRODUCTION	3
1.1 BACKGROUND	3
1.2 RELATED TECHNOLOGIES	3
1.3 RESEARCH PROBLEM STATEMENT	6
1.4 RESEARCH AIM	7
1.5 RESEARCH OBJECTIVES	8

1.6 STRUCTURE OF THE PROJECT	8
CHAPTER 2 - RESEARCH & REVIEW OF LITERATURE	9
2.1 FATIGUE DETECTION TECHNOLOGIES	9
2.2 ANDROID DEPLOYMENT	10
2.3 LITERATURE SUMMARY	13
CHAPTER 3 - RESEARCH METHODOLOGY	13
3.1 RESEARCH PHILOSOPHY	13
3.2 RESEARCH APPROACH	14
3.3 METHODOLOGICAL CHOICE	15
3.4 RESEARCH STRATEGY	15
3.5 RESEARCH PROCESS	15
CHAPTER 4- DEVELOPMENT METHODOLOGY	16
4.1 INITIAL PLANNING AND REQUIREMENTS ANALYSIS	16
4.2 SOFTWARE SYSTEM DESIGN	17
4.3 IMPLEMENTATION	18
4.4 TESTING	19
CHAPTER 5- IMPLEMENTATION	20
5.1 INITIAL PLANNING AND REQUIREMENTS ANALYSIS	21
5.2 SOFTWARE DESIGN	22
5.3 SOFTWARE AND MODELS DEVELOPMENT	24
5.4 PERFORMANCE EVALUATION AND TESTING	34
REAL-TIME VIDEO DETECTION	35
CHAPTER 6- EXPERIMENT AND RESULT ANALYSIS	37
6.1 DATASET EFFECTIVENESS EVALUATION	38
6.2 FACE DETECTION EVALUATION	38
6.3 FATIGUE DETECTION EVALUATION	39
6.4 ANDROID APPLICATION TESTING	42
CHAPTER 7- CONCLUSIONS AND RECOMMENDATIONS	42
CHAPTER 8- REFERENCES	45
PROJECT MANAGEMENT	55

CHAPTER 1 INTRODUCTION

1.1 Background

Driver fatigue is increasingly recognized as a critical factor contributing to road accidents. For this reason, traffic accidents involving driver fatigue and driver carelessness have been followed by researchers [1]. This fatigue diverts drivers' focus, impairs standard driving practices, and potentially jeopardizes public safety. Research indicates that the mortality and serious injury incidence from fatigue-related terrestrial vehicle accidents could be 50% greater due to the driver's compromised ability to execute timely braking or maneuvering [4]. According to statistics from the World Health Organization (WHO), traffic accidents result in the loss of millions of lives annually. The American Automobile Association reported that 7% of all accidents and 21% of fatal traffic accidents were caused by tired drivers [2].

Typically, when drivers exhibit frequent blinking, yawning, and nodding, they are already showing signs of fatigue; however, it is difficult for drivers to be aware of their own fatigue. If drivers could react just half a second earlier before an accident occurs, approximately 60% of traffic accidents could be avoided [3]. Therefore, in order to ensure the safety of drivers' lives and property, reduce the incidence of road traffic accidents, and minimize the adverse consequences and chain reactions caused by fatigued driving, it is necessary to propose and design an efficient and real-time monitoring system that provides timely feedback on the driver's fatigue state.

1.2 Related technologies

Fatigue driving has become a significant research focus in the fields of automotive safety and deep learning, both domestically and internationally. Researchers have proposed various measures to detect signs of drowsiness early and prevent accidents. These measures can be divided into Three main categories: first, image-based measures, which are obtained by analyzing the driver's actions and facial expressions using cameras; second, bio-based measures, which are related to the driver's biological signals and can be recorded by placing special sensors

on the driver; third, vehicle-based measures, which rely on monitoring the vehicle's behavior and movement [5].

Many biological signals, such as brain activity, heart rate, respiratory rate, pulse rate, and body temperature signals, have been used to detect driver drowsiness [6]. These biological signals, also known as physiological measurements [7]. Methods based on physiological measurements typically have higher accuracy. This accuracy is due to their ability to capture early biological changes that may occur, such as in the case of drowsiness, thus alerting the driver before any physical signs of drowsiness appear. For instance, Li et al. [8] introduced a smartwatch-based EEG system that uses an SVM probabilistic model to provide continuous drowsiness measures, achieving up to 91.92% accuracy. Kaur and Singh [9] utilized empirical mode decomposition and ANN, reaching 88.22% accuracy. Budak et al. [10] proposed an EEG-based method using LSTM networks, with an accuracy of 94.31%. Taran and Bajaj [11] employed adaptive Hermite decomposition with an ELM classifier, achieving 92.28% accuracy. Lastly, Khare and Bajaj [12] developed an entropy-based method using AVMD and an ensemble boosted tree classifier, resulting in a 97.19% accuracy. These methods represent the forefront of drowsiness detection technology, offering robust and reliable solutions.

However, there are inherent drawbacks to using physiological detection methods for drowsiness, such as the need for specialized equipment, and the intrusive nature of the detection can cause discomfort and inconvenience for the driver. These methods are also sensitive to noise and external factors, which can reduce accuracy in real-world conditions.

According to Pratama et al. [13], vehicle-based detection methods are less studied due to the challenge of accurately identifying drowsy driving features. To improve accuracy, researchers often combine this method with image-based or biological approaches [14,15,16]. The most common vehicle-based measures are steering wheel angle (SWA) and lane departure, which can be tracked using sensors and lane curvature analysis. Vehicle-based drowsiness detection methods offer non-intrusive solutions by analyzing driving patterns, particularly through steering

wheel angle (SWA) data. For example, McDonald et al. [15] used SWA data with a random forest algorithm, achieving 79% accuracy, while Arefnezhad et al. [16] further improved detection by using an ANFIS-based feature selection method, achieving 98.12% accuracy. However, these methods have some limitations that restrict their practicality: firstly, many older vehicles lack the necessary tools and equipment for detection; secondly, the variability in individual driving styles and the difficulty of accurately detecting drowsiness features under different driving conditions further hinder their effectiveness.

Despite their advantages, vehicle dynamics-based fatigue detection methods face significant challenges. The installation and maintenance costs of onboard sensors and systems are relatively high, limiting the widespread adoption of this technology. Furthermore, because this method infers driver fatigue primarily from changes in vehicle behavior, it may be susceptible to external environmental factors such as weather, road conditions, and individual driving styles, potentially leading to detection errors. These external variables, particularly in complex driving scenarios like nighttime or long-distance driving, can increase the complexity and uncertainty of detection, further complicating the effectiveness of these systems.

Some signs of drowsiness are visible and can be recorded by cameras or visual sensors. These signs include the driver's facial expressions and movements, especially head movements. In the literature, these signs are referred to as visual [17] or image-based measures [18]. In our work, we refer to them as image-based measures to emphasize that these approaches typically involve extracting features from images or videos [19,20,21]. Image-based drowsiness detection systems can be broadly categorized into three techniques: observing mouth movements, head movements, or eye movements. These systems are generally based on eye-state features and offer several significant advantages. First, they perform detection in a non-intrusive and non-invasive manner, causing no discomfort or interference to the driver. Second, they are relatively cost-effective, requiring only a camera to achieve effective drowsiness detection. Additionally, image-based systems can operate under various lighting conditions and can be continuously optimized and

enhanced through software upgrades, making them a flexible and efficient solution for monitoring driver fatigue.

1.3 Research problem statement

The non-intrusive nature of computer vision-based fatigue detection methods eliminates the need for physiological signal sensors and avoids interference from weather conditions or road environments that can affect vehicle dynamic data. These advantages have positioned computer vision-based methods as a major focus in current research and applications for fatigue driving detection. However, deploying computer vision-based facial feature fatigue detection systems in older vehicle models presents certain limitations.

Older vehicles often lack the computational power and hardware support required to run complex deep learning algorithms and perform real-time image processing. These systems depend on high-performance cameras and processors to capture and analyze facial features accurately. Unfortunately, most older vehicles are not equipped with such advanced hardware, making the deployment and operation of these systems challenging. Additionally, older vehicles may not have sufficient power supply to support the continuous operation of these systems, further limiting their practicality in real driving environments.

The design and layout of older vehicles also pose challenges for the installation and use of detection systems. Accurate facial feature capture depends heavily on the position and angle of the camera, yet the design of older vehicle cabins may not support optimal camera placement, resulting in reduced system performance. Furthermore, the materials and lighting conditions inside older vehicles can impact the image quality captured by the camera, subsequently decreasing the accuracy of fatigue detection. These factors present significant hurdles to implementing computer vision-based facial feature fatigue detection systems in older vehicles.

Therefore, investigating how to develop a non-intrusive and easily deployable fatigue monitoring integrated system becomes crucial to address the existing research gaps.

Given Android's significant 70.69% market share in the global smartphone market [22], we can leverage the widespread availability and high performance of Android devices to address some of these challenges. The extensive reach of Android devices makes them an ideal platform for deploying computer vision-based facial feature fatigue detection systems in older vehicles.

Most modern Android smartphones are equipped with powerful processors and high-quality cameras capable of running complex deep learning algorithms efficiently and accurately capturing the driver's facial features. This means that even in hardware-limited older vehicles, Android devices can still deliver high-precision fatigue detection. Moreover, the flexibility and portability of Android devices allow them to be easily adapted to different vehicle interiors. By appropriately adjusting the camera's installation position and angle, optimal facial feature capture can be ensured, even in older models.

Additionally, the user-friendly nature of Android devices, coupled with the availability of extensive development tools, enhances their suitability for use in older vehicles. Existing Android apps and the development ecosystem can be leveraged to quickly develop and deploy computer vision-based fatigue detection systems without requiring significant hardware modifications to older vehicles. This approach not only reduces costs but also improves the system's scalability and user experience. Therefore, using Android devices as a platform in older vehicles provides a cost-effective solution while opening new avenues for improving driving safety.

1.4 Research Aim

This study research Aim to develop an efficient, real-time, non-intrusive driver fatigue monitoring system based on smartphones, integrating the precision of AI models with the widespread deployment capabilities of smartphones. Utilizing existing smartphone hardware and software features, this system can be widely applied at a lower cost, thereby effectively enhancing road safety.

1.5 Research Objectives

This research aims to address this issue through three key objectives: 1) Precisely identifying human faces to provide a basis for fatigue detection; 2) Training and validating the deep learning model for accurate determination of fatigue states in detected faces; 3) Integrating the trained models into Android software, enabling its easy invocation through an app for any car with Android devices.

This research aims to develop a real-time drowsy driving detection system by integrating two advanced and lightweight neural network models, SSD and MobileNetV3, into the Android platform. The system enhances the accuracy and response speed of drowsy driving detection while reducing false alarm rates and improving user convenience compared to existing fatigue monitoring systems. The study employs quantitative analysis to validate the hypothesis, evaluating the accuracy, real-time performance, and stability of the model, while also considering the impact of hardware configuration, software environment, and data preprocessing on detection outcomes. This system offers an easily accessible and deployable solution, promising a significant contribution to the field of fatigue driving monitoring.

1.6 Structure Of The Project

The structure of this paper is arranged as follows: Chapter One provides a background summary of the report, clearly defines the research objectives, and emphasizes the specific goals necessary for achieving successful outcomes. Chapter Two focuses on the literature review, analyzing relevant studies and highlighting the advantages of the models used. Chapter Three details the research methodology, emphasizing its crucial role in guiding the entire research process. Chapter Four outlines the development methodology, explaining the advantages and practicality of the adopted approach in system development. Chapter Five, based on the research and development methodologies, presents

the specific details of the development and implementation. Chapter Six focuses on the presentation of experimental results and the analysis process. Chapter Seven concludes the paper and offers a perspective on future research directions.

CHAPTER 2 - RESEARCH & REVIEW OF LITERATURE

In the first chapter introduction, we discussed the background of drowsy driving and various types of fatigue detection methods, highlighting the advantages of computer vision-based facial detection technology as the most advanced and mainstream approach. We also pointed out its deployment limitations in low-end and older vehicle models. In the following literature review, we will further explore the research findings related to computer vision-based facial detection and provide a detailed explanation of the models used in this project.

2.1 Fatigue Detection Technologies

The face is a crucial part of the body and provides many features. When a driver feels fatigued, their facial expressions (such as blinking and yawning) differ from those in a normal state. Currently, most vision-based fatigue driving detection models are built on convolutional neural networks (CNNs), long short-term memory networks (LSTMs), and their variants. Kim et al. [23] were the first to attempt eye state classification using CNNs. Subsequently, Zhao et al. [24] integrated features extracted from pre-trained CNNs and fully connected neural networks to perform eye state classification. However, these deep models did not attempt to link eye states with the driver's fatigue state.

To address this issue, Chirra et al. [25] predicted fatigue states by detecting 10 consecutive frames of closed eyes. Additionally, more researchers (such as Hao et al., [27]; Liu et al., [26]) combined the PERCLOS criterion with CNNs to predict fatigue states. Xiao et al. [28] proposed an end-to-end model that replaced the PERCLOS criterion with an LSTM network and achieved higher prediction accuracy.

Furthermore, numerous studies have focused on enhancing CNNs and LSTMs to improve the effectiveness of fatigue driving detection. Savas and Becerikli proposed a multi-task ConNN model for detecting driver drowsiness in their research [29]. They determined the level of driver fatigue by calculating the eye closure time/percentage of eye closure (PERCLOS) and mouth frequency (FOM). Additionally, the model is reported to achieve an accuracy of 98.81% on the YawDD and

NthuDD datasets. [30] proposed an improved ReLU-BiLSTM deep neural network for driver mental fatigue detection based on head pose. The results showed that the proposed classifier outperformed state-of-the-art methods and traditional machine learning tools, achieving a training accuracy of 99.2%, sensitivity of 97.54%, and F1 score of 97.46%. [31] introduced a convolutional neural network called EM-CNN to detect eye and mouth states from ROI images. The percentage of eyelid closure over time (PERCLOS) and the percentage of mouth opening (POM) were used as key parameters for fatigue detection. Experimental results demonstrated that the proposed EM-CNN effectively detects driver fatigue using driving images. The algorithm outperformed other CNN-based methods, such as AlexNet, VGG-16, GoogLeNet, and ResNet50, achieving an accuracy of 93.623% and a sensitivity of 93.643%.

CNNs, LSTMs, and their variants, while powerful, have performance limitations, particularly in older vehicle models that often lack robust computing power and high-resolution camera systems. These models typically require significant computational resources and high-quality input data, which may not be available in such environments. Therefore, there is a need to develop lightweight detection models that are optimized for mobile devices, ensuring effective fatigue detection even in scenarios with limited hardware capabilities.

Although the CNNs, LSTMs, and their variants mentioned in the literature demonstrate strong performance, they also have certain limitations and shortcomings. Firstly, in older vehicle models, these vehicles often lack powerful computing equipment and high-resolution cameras. These models typically rely on substantial computational resources and high-quality input data, which may be difficult to achieve in such environments. Therefore, there is an urgent need to develop lightweight detection models suitable for mobile devices to ensure effective driver fatigue detection even under limited hardware conditions.

Secondly, the methods proposed in the literature lack clear guidance and strategies for actual deployment, particularly in processing camera video and converting it into suitable images for detection. The absence of effective video processing methods creates difficulties in converting video into a detectable state, further limiting the practical application of these models in real-world scenarios.

Therefore, to successfully deploy a real-time driver fatigue detection system in practical applications, we need to address two key challenges: first, how to effectively capture and process faces in real-time video images; second, ensuring that the deep detection model is suitable for running on mobile devices.

For the first challenge, real-time face capture requires the detection system to quickly and accurately recognize faces, even under complex driving conditions such as varying lighting and

changes in posture. The Single Shot MultiBox Detector (SSD) is an efficient object detection algorithm known for its speed and low computational cost [32]. SSD achieves precise localization and classification of multi-scale objects by simultaneously detecting them on feature maps at different scales. It uses a CNN as the base network, performing detection across multiple layers, which allows it to maintain high detection accuracy while significantly reducing computational complexity [33].

SSD is particularly well-suited for face detection on mobile devices due to its lightweight architecture and real-time capabilities. Firstly, SSD does not require the generation of candidate regions like two-stage detectors (e.g., Faster R-CNN), which reduces computational steps and increases detection speed — a critical factor for devices with limited computing power [34]. Secondly, SSD can perform detection on smaller feature maps, reducing memory and computational resource requirements, making it efficient for mobile devices. Additionally, SSD's end-to-end design enables direct face detection in images without the need for complex preprocessing or postprocessing steps, further enhancing its practicality for mobile deployment.

For the second challenge, the adaptability of deep detection models on mobile devices is crucial. These models must be lightweight and have low computational complexity to ensure efficient operation on devices with limited computing resources. Additionally, the model should feature low latency and low power consumption to meet the real-time processing demands of mobile devices while extending battery life.

In tackling these challenges, MobileNetV3 demonstrates significant advantages. MobileNetV3 is a lightweight convolutional neural network developed by Google, optimized specifically for mobile and embedded devices [35]. Compared to similar models, such as MobileNetV2 [38], ShuffleNet [39], and EfficientNet-Lite [40], MobileNetV3 reduces both parameter count and computational complexity while maintaining high accuracy. It employs Neural Architecture Search (NAS) technology, combined with Squeeze-and-Excitation (SE) modules and the hard Swish activation function, greatly enhancing network efficiency and performance [36].

Specifically, MobileNetV3 improves upon MobileNetV2 and ShuffleNet not only in terms of model size and computational complexity but also in inference speed and energy efficiency [37]. It reduces multiply-add operations (MAdd) and parameter counts while still achieving high classification accuracy on datasets like ImageNet. Furthermore, compared to EfficientNet-Lite, which performs well in balancing model size, computational load, and accuracy, MobileNetV3 excels in inference efficiency and power management on mobile devices [41]. The architecture of MobileNetV3 is better suited for real-time processing needs, providing faster response times in low-resource environments and effectively extending battery life.

Therefore, MobileNetV3 is an ideal choice for lightweight, real-time driver fatigue detection, achieving a superior balance between accuracy, efficiency, and power consumption, making it particularly suitable for deployment in mobile and embedded systems.

2.2 Android Deployment Of Deep Neural Network Tech

To enhance the accessibility of this project, it is essential to choose a fatigue driving detection system that is widely applicable and easy to deploy on common mobile devices. Android devices, with their extensive market share and open operating system, have become one of the most prevalent mobile platforms worldwide [42]. Their widespread adoption is attributed to diverse hardware support, a flexible development environment, and a large user base. Android devices are typically equipped with high-performance processors and multi-core architectures, providing sufficient computational power for deep learning-based applications. Furthermore, the Android ecosystem supports a rich set of development tools and libraries, such as TensorFlow Lite [43] and ML Kit [44], enabling developers to easily deploy complex deep learning models on mobile devices.

These advantages make Android devices particularly well-suited for deploying deep learning-based fatigue driving detection systems. By leveraging the computational power and sensors, such as cameras, available on Android devices, the system can analyze the driver's facial features and behavior in real-time to detect signs of fatigue. Moreover, Android's openness and widespread use ensure that the system can be widely applied across various vehicles and driving environments, thereby enhancing the system's adoption and practicality.

To deploy deep learning detection models to Android devices, we need to use ONNX. ONNX (Open Neural Network Exchange) is an open format designed to represent machine learning models [45]. It establishes standardized operators for both machine learning and deep learning models and offers a universal file format that facilitates the transfer and deployment of models across different frameworks and tools. This interoperability is particularly valuable for AI developers, as it allows them to use a single model across various platforms without the need for extensive reconfiguration.

First, convert the deep learning model to the ONNX (Open Neural Network Exchange) format, an open standard that supports interoperability across multiple frameworks. Once converted to ONNX, the model can be more easily deployed across different hardware and software environments. Next, use ONNX Runtime Mobile to integrate the optimized model into the Android application. ONNX Runtime Mobile [46] is an inference engine specifically optimized for mobile devices, capable of delivering fast and low-latency inference performance in resource-constrained environments. This approach allows the deep learning model to fully leverage the computational capabilities of Android devices, enabling real-time driver fatigue detection while ensuring the application's operational efficiency and battery life.

2.3 Literature summary

This project addresses driver fatigue by selecting MobileNetV3 and SSD for facial detection and fatigue assessment. It leverages the low latency and compact design of MobileNetV3 combined with SSD's multi-scale feature detection capabilities, overcoming the limitations of traditional CNNs in processing speed and computational resources. Given the widespread use and hardware advantages of Android devices, the project utilizes ONNX to optimize the deployment of the model on mobile devices, offering a real-time, scalable, and cost-effective fatigue detection solution aimed at enhancing driver safety.

CHAPTER 3 - RESEARCH METHODOLOGY

This study focuses on developing a driver fatigue detection system using SSD and MobileNetV3 as the core models, which are then deployed on an Android platform to achieve real-time monitoring and alerting capabilities. To ensure the systematic and scientific nature of this research, we have structured our methodology based on the research onion framework, detailed as follows.

3.1 Research Philosophy

This research is based on positivism, a philosophical approach that emphasizes understanding phenomena through observation and measurement to ensure the objectivity and reliability of results [47]. Positivism asserts that by systematically collecting and analyzing data, we can achieve an accurate understanding of the

research subject, thereby reducing subjective bias. This principle aligns closely with our goal of developing a reliable fatigue detection system based on quantifiable data and models [48].

Applying positivist principles offers significant benefits to our system. First, data-driven methods allow us to establish and validate models through actual observation and measurement, ensuring their accuracy and effectiveness. Second, positivism enables us to test and optimize the system in various real-world scenarios through quantitative data analysis, enhancing its stability and generalization capabilities [49]. By conducting scientific experiments and data analyses, we can clearly identify the strengths and weaknesses of the system, leading to continuous improvement and optimization of the fatigue detection algorithm. Finally, this data-based objective analysis method minimizes the influence of personal subjective judgment, making the evaluation and adjustment process of the system more transparent and credible.

3.2 Research Approach

This study adopts a deductive approach, starting from established theoretical frameworks and using systematic data collection and analysis to test hypotheses [50]. The core principle of the deductive method is to derive specific predictions or hypotheses from widely accepted theories and then test these predictions through experiments or data analysis to determine their validity [51]. One significant advantage of this method is that it ensures the logical rigor and reliability of the research conclusions, as the process is grounded in well-established theoretical foundations [52].

In this project, the deductive method is particularly suitable because it allows us to rely on existing theoretical frameworks in deep learning and computer vision, applying these theories to guide the design and implementation of a driver fatigue detection system. By using this approach, we can more effectively select and optimize models (such as SSD and MobileNetV3) and validate their effectiveness in practical applications through experimentation. Ultimately, this method not only

helps us verify the applicability of relevant theories in fatigue detection but also provides a scientific basis for further model optimization and improvement.

3.3 Methodological Choice

We adopted a quantitative research methodology, which involves systematically collecting and analyzing numerical data to identify potential patterns and test hypotheses. Quantitative methodology emphasizes the objectivity and measurability of data, typically involving the use of statistical analysis, regression analysis, and other mathematical tools to gain a deeper understanding of the underlying patterns in the data [53]. This approach is particularly well-suited for evaluating model performance, as it provides precise numerical results that help researchers quantify the accuracy, efficiency, and other key metrics of the models.

In this study, the quantitative methodology allows us to rigorously assess the performance of deep learning models such as SSD and MobileNetV3. By analyzing the numerical outputs of these models, we can determine their strengths and weaknesses in various scenarios and optimize them accordingly. Ultimately, the quantitative methodology not only provides a scientific basis for evaluating the effectiveness of the models but also offers clear direction for further improvement and refinement.

3.4 Research Strategy

This project adopts an experimental research strategy, which involves manipulating variables to observe their effects on other variables. In this study, we conducted a series of experiments to evaluate the performance of the models under various conditions, such as different lighting environments and hardware configurations. This strategy allows us to thoroughly investigate and optimize the models' adaptability and performance across different scenarios, ensuring their effectiveness and reliability in practical applications.

3.5 Research process

This study follows a six-stage design science research framework to systematically develop and implement a driver fatigue detection model, as illustrated in Figure 1 [54]. The process begins with (1) identifying and precisely defining the problem to be addressed, followed by (2) detailing the research objectives and expected outcomes to guide the study's direction. The next stage involves (3) designing and developing the necessary artifacts or models to solve the problem, which are critical for achieving the research goals. These models are then (4) practically tested to verify their functionality and effectiveness. After testing, we (5) thoroughly analyze the data to assess the model's performance and its ability to solve the identified problem. Finally, (6) the results are communicated to relevant stakeholders to promote the application and further development of the research findings. The techniques and procedures employed include data collection, model development, performance evaluation, and software development. Specifically, we first collect training and testing data from the YAWDD dataset, then develop the models using PyTorch and DGL frameworks in a Linux environment. These models are trained and optimized on a server equipped with NVIDIA GPUs to ensure efficient computational performance. Subsequently, the trained models are deployed on an Android platform, where we develop the corresponding fatigue detection software, completing the entire process from model development to real-world application. Various statistical metrics are used to evaluate the model's performance, ensuring its effectiveness and reliability in practical use.

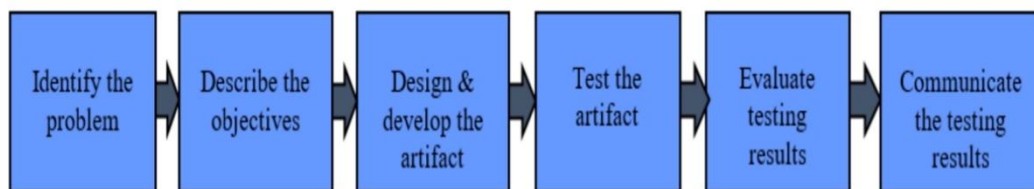


Figure 1. Research Framework

CHAPTER 4- DEVELOPMENT METHODOLOGY

During the development phase, the project will be divided into four main stages based on its characteristics: initial planning and requirements analysis, design phase, implementation phase, testing phase. This phased approach helps ensure

transparency and manageability of the development process, while also allowing resources and attention to be focused at each stage for optimal development outcomes.

4.1 Initial Planning And Requirements Analysis

During the initial planning and requirements analysis phase, requirement lists and user stories are essential tools that help the team comprehensively and systematically understand the functional and non-functional requirements of the software to be developed. Requirement lists thoroughly outline all the necessary technical and non-technical conditions, such as performance standards, user interface design, and security requirements. This ensures that the team has a clear understanding of the project's goals and scope, providing a definitive reference standard throughout the development process.

User stories, starting from the user's perspective, describe functional needs in the format of "As a [role], I want [to do something] so that [I achieve some benefit]," allowing the development team to deeply understand users' expectations and needs [55]. This approach not only ensures that development focuses on enhancing user experience and meeting actual needs but also helps identify key requirements early in the project, avoiding misunderstandings or oversights later on, thereby reducing rework, enhancing development efficiency, and improving product quality [56]. In summary, requirement lists and user stories are effective communication tools that provide a solid foundation for project success.

4.2 Software System Design

During the design phase, we will use Unified Modeling Language (UML) to assist in building and refining the software architecture. UML is a standardized modeling language used for visualizing, specifying, constructing, and documenting the design of software systems [60]. It provides a range of diagrams to describe various aspects of the system, including system architecture diagrams, class diagrams, and flowcharts. The key tasks of this phase are to define the overall structure of the system, design interfaces that meet user needs, and meticulously

plan how components will work together to ensure the stability and efficiency of the software.

Various graphics used in the design phase each have their unique composition and function, playing a crucial role. System architecture diagrams provide a high-level overview of the software, showing the main components of the system and their interactions, helping designers and developers understand the system's framework and module division [57]. Class diagrams detail the classes in the system, including their attributes, methods, and the relationships between classes, such as inheritance and associations, which are especially important for object-oriented programming [58]. Finally, A flowchart is a graphical tool that uses specific symbols and arrows to represent the steps and relationships within a process or system [59]. It is commonly used in scenarios like workflow, algorithms, business processes, or project management to help users intuitively understand complex systems. The main advantage of a flowchart is its visual nature, which makes complex processes easier to understand and communicate. It helps clarify task sequences, decision points, and process branches, reducing misunderstandings, improving efficiency, and supporting project management and process optimization.

4.3 implementation

This project will employ Agile development methods during the implementation phase, offering significant advantages over traditional waterfall development models, especially in rapidly changing market and technology environments. The core advantages of Agile include its high adaptability, iterative nature, and flexibility [61]. It allows teams to continuously assess the feasibility of timelines and requirements, enabling quicker responses to changes. Additionally, Agile methodologies encourage frequent communication and collaboration, ensuring synchronization among project members, which helps in early problem detection and resolution, reducing misunderstandings and rework.

Scrum is an agile development framework that emphasizes iterative progress, flexibility, and team collaboration, making it particularly well-suited for dynamic and

complex projects [62]. Although Scrum is typically used for team-based development, applying it to individual projects can still enhance development efficiency through iterative cycles and continuous feedback. The core of Scrum consists of several key components, including the Product Backlog, Sprint, Daily Stand-up, Sprint Review, and Sprint Retrospective.

In an individual project, the Product Backlog can be constructed by creating user stories, which represent the user's needs and feature requirements. Each user story is broken down into smaller tasks and prioritized in the Product Backlog. Next, the content of each Sprint can be planned by selecting the highest priority tasks for development. Each Sprint usually lasts from one to four weeks, during which the focus is on completing the selected tasks. At the end of each Sprint, a review can be conducted to evaluate the work results and processes.

In this way, Scrum allows individual projects to maintain efficient progress while continuously adjusting and optimizing development strategies, ensuring that the project can flexibly respond to changing requirements and challenges. The iterative process not only helps individual developers better manage their time and resources but also ensures the quality of the final product and user satisfaction.

4.4 Testing

During the testing phase of this project, we will conduct both functional and performance tests to ensure the reliability and effectiveness of the real-time fatigue driving detection system [63]. Functional tests will verify whether the system's core functionalities meet the specified requirements, including the accurate detection of a driver's fatigue state and the issuance of warnings [64]. Through black box testing, we will focus on the system's performance under various input conditions to ensure the accuracy and timeliness of the detection results, without delving into the details of internal implementations [65]. Meanwhile, white box testing will deeply analyze the internal logic and structure of the system to ensure that all modules function as expected across a range of complex scenarios. These tests are crucial for identifying functional issues and ensuring the system's effectiveness in real-world use.

Performance testing is particularly critical for a real-time fatigue driving detection system, with a focus on response times and stability under high loads. The design of test cases is key in this process; they should cover a wide range of normal and exceptional usage scenarios to ensure that the system can accurately identify fatigue states under various conditions and maintain robust performance even under high loads or in abnormal situations. By conducting these tests, we aim to minimize system failures and enhance the system's reliability and user experience.

CHAPTER 5- IMPLEMENTAION

5.1 initial planning and requirements analysis

In the initial planning and requirements analysis phase of the project, we will use a requirements list and user stories to systematically define our needs. Given that the goal of this project is to develop a fatigue driving detection software based on deep learning models, deployable on the Android platform, we will detail all relevant technical and non-technical requirements. This step is crucial to ensure the smooth progress of the project and helps us clearly understand the project's goals and scope. Below is our requirements list, which details the functionalities and performance standards that the software should meet. the requirements list show as in table 1.

Requirement Classification	Requirement Description
Model detection	1. Obtain datasets of diverse facial images and fatigue driving behavior. 2. Utilize the collected dataset to train the model, ensuring it is fine-tuned for accurate detection and analysis based on real-world data. 3. Validate the model's accuracy, recall, and execution speed on an independent test set. 4. Ensure efficient Android integration with optimized model performance.

Fatigue Driving Detection Software	1. Real-time video capture and processing for deep learning analysis via the device's camera. 2. Display fatigue detection results in real-time. 3. Trigger alerts (visual, audio, vibration) for detected driver fatigue.
------------------------------------	--

Table1. Requirements List

Further, we will refine the above requirements into user stories to ensure a more specific understanding of the project needs, making the development process more transparent and efficient.

- As a researcher, I want to gather data on fatigue driving behavior, so that the model can identify signs of driver fatigue accurately.
- As a researcher, I want the model to converge stably during training so that it can achieve higher accuracy and reliability in real-world applications.
- As a researcher, I want the model to achieve stable convergence during training so that it can deliver higher accuracy and reliability in real-world applications.
- As a researcher, I want the model to achieve high detection accuracy on the independent test set so that it can accurately identify and prevent fatigue driving in real-world applications, ensuring driving safety.
- As a researcher, I want the model to be convertible into a deployable form on mobile devices so that users can monitor their driving status in real-time, thereby enhancing driving safety.
- As a driver, I want the app to capture and process real-time video from my device's camera, so that it can analyze my driving behavior continuously and provide accurate fatigue detection.
- As a driver, I need the app to display fatigue detection results in real-time, so that I can immediately understand my fatigue status and take necessary actions if needed.

- As a driver, I want the app to trigger visual, audio, or vibration alerts when fatigue is detected, so that I can be promptly alerted to any potential risks and ensure my safety while driving.

5.2 Software Design

As shown in Figure 2, the system architecture of this project includes several key modules, such as the Fatigue Driving Detection Model, Video Processing Module, Device Camera Access Module, Real-time Display and Alert Module, and User Interface Module. The Fatigue Driving Detection Model uses deep learning algorithms to analyze video data and accurately identify the driver's fatigue state. The Video Processing Module processes raw video streams from the camera in real time, providing optimized input for the model. The Device Camera Access Module manages access and control of the camera, ensuring stable real-time video capture. The Real-time Display and Alert Module displays the driver's status in real-time based on the model's analysis and triggers necessary alerts when needed. Finally, the User Interface Module offers an intuitive and user-friendly interface, making it easy for users to view detection results and receive alerts. These modules work closely together to enhance driving safety.

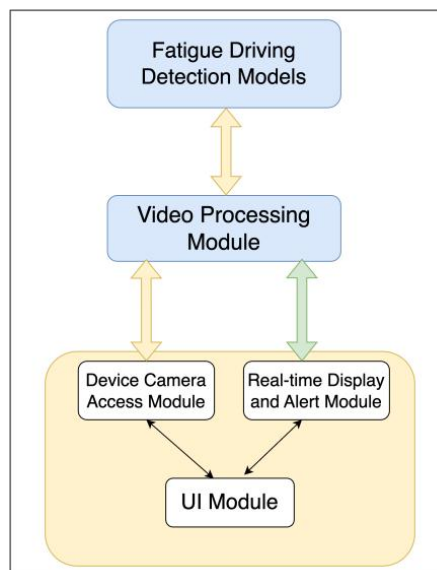


Figure2. System Architecture

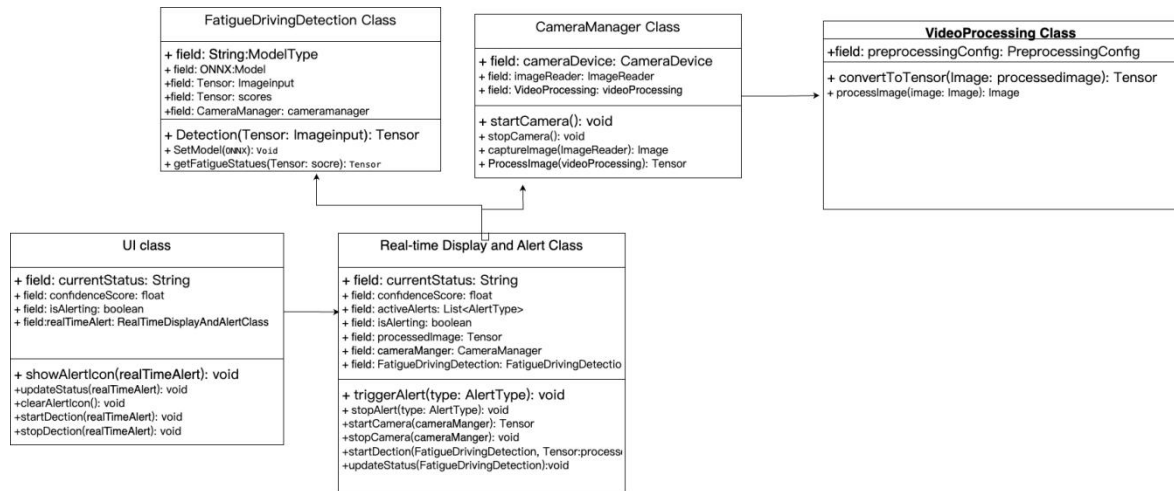


Figure3. Class Diagram

As shown in Figure 3, this project consists of six main classes: FatigueDrivingDetection Class, CameraManager Class, VideoProcessing Class, UI Class, and Real-time Display and Alert Class. The FatigueDrivingDetection Class is the core component of the system, responsible for fatigue detection on input image data through deep learning models and for outputting the detection results. The CameraManager Class manages the camera and handles image acquisition, ensuring that the system can capture real-time images of the driver. The VideoProcessing Class deals with image preprocessing and tensor conversion, ensuring that the input data is suitable for model processing. The UI Class provides user interface display and interaction functions, helping users understand the current system status and alert conditions. Finally, the Real-time Display and Alert Class implements real-time monitoring and alert triggering, ensuring timely warnings are issued when potential fatigue driving is detected. These classes interact through well-defined interfaces, collectively forming a comprehensive fatigue driving detection system.

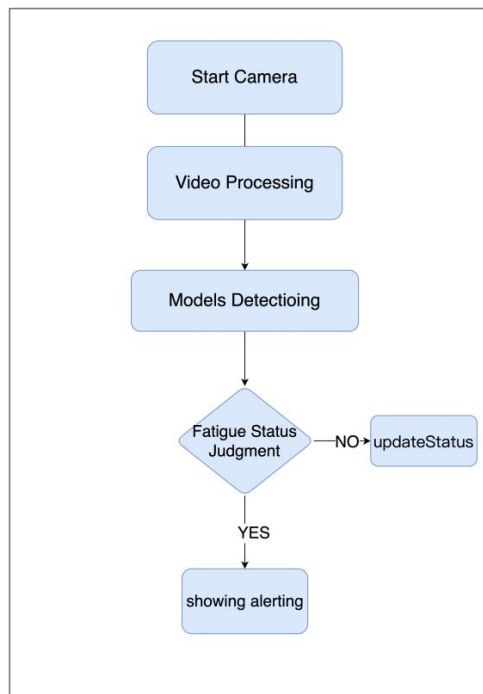


Figure 4. Flowchart

As shown in Figure 4, the image illustrates the core workflow of the fatigue driving detection system, divided into five main steps. First, the system starts the camera to capture real-time image data, which serves as the foundation of the entire process. Next, the captured video data undergoes processing to prepare it for model detection. In the detection stage, the system employs deep learning models to analyze the processed data for signs of driver fatigue. The system then assesses the detection results in the fatigue status judgment step. If fatigue is detected, an alert is triggered to warn the driver; if not, the system updates its status and continues monitoring. This workflow enables real-time monitoring and early warning of driver fatigue, enhancing driving safety.

5.3 Software and Models Development

This chapter will provide a detailed description of the development and implementation process of the fatigue driving detection system. Based on the introduction in the development methodology section, we have decided to adopt the Scrum method to develop this fatigue driving detection system. Based on the

user stories and the software designs outlined in the 4.1 and 4.2, we can plan the corresponding Product Backlog, Sprint Planning, and Sprint Backlog to ensure orderly progress of the project and the gradual achievement of its goals.

5.3.1 sprint planning

First, we need to create the Product Backlog. It is a core tool in project development, used to record and prioritize all the features and requirements that need to be implemented.

- 1). Collect data on fatigue driving behavior.
- 2). Design the model and train it on the dataset to achieve stable convergence.
- 3). Achieve high detection accuracy on the independent test set.
- 4). Convert the model into a deployable form on mobile devices.
- 5). Capture and process video in real-time on mobile devices.
- 6). Display fatigue detection results in real-time.
- 7). Trigger alerts when fatigue is detected.

In each sprint, we will select the highest-priority tasks from the Product Backlog and create the Sprint Backlog accordingly. Below is a detailed plan for each sprint, outlining the specific tasks and objectives to be completed during each sprint to ensure the gradual progression and ultimate success of the project.

Sprint 1: Data Collection and Initial Model Setup

- Tasks:

Collect and preprocess data on fatigue driving behavior.

Process the dataset to make it suitable for model training.

Objective: Prepare a comprehensive dataset for model training.

Sprint 2: Model Training and Validation

Tasks:

Design the model architecture and train the model on the dataset.

Optimize the model based on initial training results.

Test the model on an independent test set to evaluate accuracy.

Objective: Train and test the fatigue driving detection model to achieve high detection accuracy.

Sprint 3: Model Deployment and Software Development

Tasks:

Convert the model into a deployable format for mobile devices.

Develop a user interface for real-time display of fatigue detection results.

Implement real-time video capture using the device's camera.

Integrate video processing with the fatigue detection model.

Design and implement audio and vibration alerts with the fatigue detection model.

Test the alert system in real-world scenarios.

Objective: Integrate the model into an Android application to provide accurate fatigue driving detection.

After completing each sprint, reviews and retrospectives will be conducted to ensure continuous improvement and optimization of the project.

5.3.2 Sprint 1: Data Collection and Initial Model Setup

1) Dataset Description

In this study, we utilize the YawDD dataset, which consists of videos recorded by an in-car camera of drivers in an actual car. This dataset includes various facial characteristics (male and female, with and without glasses/sunglasses, different ethnicities) and captures drivers talking, singing, being silent, and yawning. It is primarily used to develop and test algorithms and models for yawning detection, as well as for face and mouth recognition and tracking. The videos are recorded under natural and varying illumination conditions [66]. a sample of YAWDD show as in figure 6.

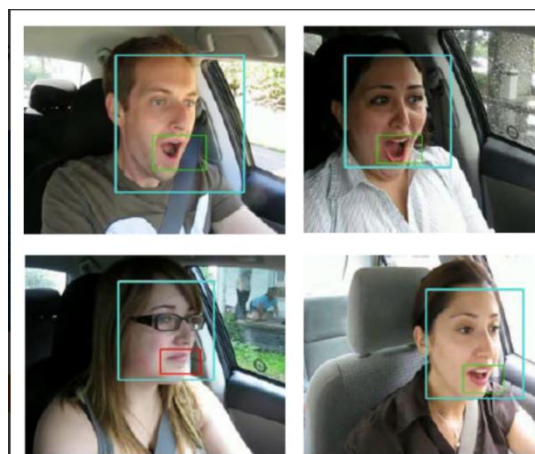


Figure 6. An illustration of YAWDD

This dataset is crucial for fatigue detection research as it provides the necessary data to extract and analyze features indicative of driver fatigue. Given that the images in the YawDD dataset contain valuable information about driver behavior, it is essential to preprocess these images to extract meaningful features. These features include facial landmarks, mouth states, and other related indicators critical for determining whether a driver is fatigued. To efficiently extract these features, we employ advanced computer vision techniques, including SSD and MobileNetV3 models.

2) Data Processing: Extracting Face Images Using SSD

Firstly this project use the SSD algorithm for preprocessing the dataset. This involves identifying and extracting facial features from the images, which are then saved for further analysis. SSD operates effectively by deploying a single deep neural network which combines features across various scales and aspect ratios. It processes an entire image in one go through the network, detecting objects at multiple scales and at different locations in parallel. This makes SSD not only fast but also accurate in detecting and localizing objects such as human faces within images. This approach ensures efficient and accurate facial feature extraction from the dataset.

```

01. def face_for_yawn_ssd(direc="./drowsiness-
dataset/train", ssd_model_path="./res10_300x300_ssd_iter_140000.caffemodel", prototxt_path="./depl
y.prototxt"):
02.     yaw_no = []
03.     IMG_SIZE = 145
04.     categories = ["yawn", "no_yawn"]
05.     # Load SSD model
06.     net = cv2.dnn.readNetFromCaffe(prototxt_path, ssd_model_path)
07.     for category in categories:
08.         path_link = os.path.join(direc, category)
09.         class_num1 = categories.index(category)
10.         for image in os.listdir(path_link):
11.             img_path = os.path.join(path_link, image)
12.             image_array = cv2.imread(img_path)
13.             (h, w) = image_array.shape[:2]
14.             blob = cv2.dnn.blobFromImage(cv2.resize(image_array, (300, 300)), 1.0,
15.                                           (300, 300), (104.0, 177.0, 123.0))
16.             # Face detection with SSD
17.             net.setInput(blob)
18.             detections = net.forward()
19.             for i in range(detections.shape[2]):
20.                 confidence = detections[0, 0, i, 2]
21.                 if confidence > 0.5: # Process only high confidence detections
22.                     box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
23.                     (startX, startY, endX, endY) = box.astype("int")
24.                     # Draw a green rectangle to visualize the face
25.                     cv2.rectangle(image_array, (startX, startY), (endX, endY), (0, 255, 0), 2)
26.                     # Crop face region
27.                     face = image_array[startY:endY, startX:endX]
28.                     # Resize face image to the predefined size
29.                     resized_face = cv2.resize(face, (IMG_SIZE, IMG_SIZE))
30.                     # Append resized face and its category number to the list
31.                     yaw_no.append([resized_face, class_num1])
32.     return yaw_no

```

In the above code mentioned, we begin by using OpenCV's `cv2.dnn.readNetFromCaffe` function to load a pre-trained SSD model along with its configuration file. For each image, we resize it to 300x300 pixels and normalize it by subtracting the mean and applying scaling to maintain consistency with the preprocessing done during training. The processed image is then fed into the network. The forward pass of the network yields results for face detection, which include confidence levels and bounding box coordinates. We only consider detections with a confidence level above 0.5.

For each detected face, a rectangle is drawn on the original image using the bounding box coordinates, and the face region is cropped out. This cropped face is then resized to the predefined dimensions (145x145 pixels) for further processing or classification. The processed facial images and their labels (yawning or not yawning) are stored for use in training or testing the yawning detection model.

5.3.3 Sprint 2: Model Training and Validation

1) Fatigue Detection Using MobileNetV3

After extracting the facial images using the SSD model, we employ the MobileNetV3 model to further process and analyze these images to detect whether the driver is in a fatigued state.

```
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout

IMG_SHAPE = (145, 145, 3)
base_model=tf.keras.applications.MobileNetV3Small(
    input_shape=IMG_SHAPE,
    alpha=1.0,
    minimalistic=False,
    include_top=False,
    weights="imagenet",
    #dropout_rate=0.2,
    #classifier_activation="softmax",
    include_preprocessing=False,
)

base_model.trainable = False
x = base_model.output

x = tf.keras.layers.Flatten()(x)
x = tf.keras.layers.Dense(4, activation='softmax')(x)

model = tf.keras.Model(base_model.input, x)
model.summary()
```

As show in the code, In this model, we employ the `MobileNetV3Small` architecture as the foundation, a lightweight network optimized for mobile devices designed to deliver efficient performance with limited computational resources. The model is specifically tailored to handle 145x145 pixel images with three color channels, crucial for detecting whether a driver is yawning, a key component of fatigue detection systems. Renowned for its compact structure and computational efficiency, `MobileNetV3Small` is configured with the full number of filters (Alpha set to 1.0) and pre-trained `imagenet` weights, which help the model recognize features effectively at initialization. At the top of the model architecture, we removed the original top layer and added a flattening layer followed by a dense layer with 4 output units using a `softmax` activation function to classify images as either yawning significantly, yawning slightly, normal, or other facial actions. This design strategy ensures that the model can effectively learn and predict specific behavioral recognition tasks while maintaining a balance between processing speed and accuracy.

2) training and Validation

```
optimizer1 = tf.keras.optimizers.Adam(
    learning_rate=0.0001,
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1e-07,
    amsgrad=True,
    name='Adam',
)
model.compile(loss="categorical_crossentropy",
              optimizer= optimizer1,
              metrics=["accuracy"])
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

# Define the model checkpoint callback
checkpoint = ModelCheckpoint(
    'best_model2.keras', # File name for saving the best model, using the .keras extension
    monitor='val_accuracy', # Metric to monitor is validation accuracy
    verbose=1, # Output save information
    save_best_only=True, # Only save the model with the best performance on the validation set
    mode='max' # Use the maximum value of 'val_accuracy' as the save criterion
)

# Define the early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

# Start training and pass the callbacks
history = model.fit(
    train_generator,
    epochs=50,
    validation_data=test_generator,
    shuffle=True,
    callbacks=[early_stopping, checkpoint] # Add early stopping and checkpoint callbacks
)
```

In this code, we configured an Adam optimizer for the model with a learning rate set to 0.0001 to ensure stable convergence. The Adam optimizer includes standard parameters such as a momentum term β_1 of 0.9, β_2 of 0.999, and a small epsilon value of $1e-07$ to enhance numerical stability. Additionally, the `amsgrad` variant is enabled to help address potential convergence issues with

the Adam optimizer. The model is compiled using `categorical\_crossentropy` as the loss function, suitable for multi-class problems, with `optimizer1` responsible for optimization and monitoring accuracy as a performance metric.

To enhance training efficiency and prevent overfitting, two key callback functions, `ModelCheckpoint` and `EarlyStopping`, are introduced. `ModelCheckpoint` saves the model with the highest validation accuracy after each training phase, while `EarlyStopping` monitors loss on the validation set, stopping training if there is no improvement over three consecutive epochs, and ensures the best weights are restored using `restore\_best\_weights`. Training is initiated using the `fit` method, employing training and validation set generators, set for 50 training epochs, and incorporates predefined callbacks to monitor and optimize the training process.

5.3.4 Sprint 3: Model Deployment and Software Development

1) Model Conversion Optimization

To deploy trained models on Android devices, we plan to use ONNX Runtime Mobile, a lightweight machine learning inference engine designed specifically for mobile and edge devices. We will design a dedicated class to encapsulate the logic for loading and running ONNX models. By utilizing ONNX Runtime, we can create a session in which model inference is performed, allowing for efficient data processing and analysis on mobile devices. This approach not only ensures compatibility with mobile platforms but also delivers efficient inference performance in resource-constrained environments.

```
public OnnxModelLoader(Context context, String modelPath) {
    try {
        // Create the ONNX Runtime environment
        environment = OrtEnvironment.getEnvironment();
        // Create session configuration options
        OrtSession.SessionOptions sessionOptions = new OrtSession.SessionOptions();
        sessionOptions.setOptimizationLevel(OrtSession.SessionOptions.OptLevel.BASIC_OPT);
        // Load the model
        session = environment.createSession(modelPath, sessionOptions);
    } catch (OrtException e) {
        e.printStackTrace();
    }
}
```

```
}
}
```

The `OnnxModelLoader` class defined in the code is used to load an ONNX model. Initially, it creates an ONNX Runtime environment, then sets the session configuration options, which includes setting the optimization level to basic optimization. Subsequently, the model is loaded using the specified model path and session options. If an error occurs during the session creation, the exception is caught and printed. This class provides a convenient way to manage the loading and initialization of ONNX models, facilitating subsequent model inference.

2) UI design

As show in Figure 5, the UI layout of this project primarily consists of two parts: the video detection area and the warning dialog box. The video detection area is the main part of the interface, used to display real-time video content, assisting users in monitoring and analyzing the current scene. The warning dialog box automatically pops up when specific events, such as fatigue driving behavior, are detected, issuing a warning to the user and reminding them to take appropriate actions. The entire interface is designed to be simple and intuitive, aiming to provide users with a clear and effective interaction experience.

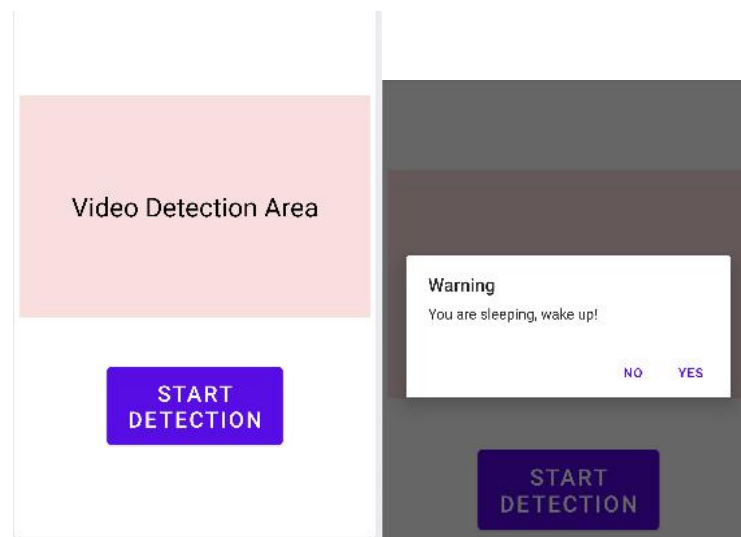


Figure 5. UI of Software

3) Facial Recognition And Fatigue State Determination

To achieve real-time facial state detection, we need to capture facial images using the camera of an Android device, followed by further processing of these images.

This process involves not only capturing real-time video but also analyzing and recognizing the captured images to accurately assess and respond to the user's facial expressions and states. This project will utilize the Android CameraX library to handle real-time camera video stream processing. CameraX is a highly compatible and user-friendly camera library that simplifies the development of camera applications and optimizes camera operations across various Android devices. Subsequently, face detection is performed using the SSD and MobilenetV3 models.

```
// Obtain the ProcessCameraProvider instance synchronously,
// which is used to bind the lifecycle of cameras to the lifecycle owner.
ProcessCameraProvider cameraProvider = ProcessCameraProvider.
getInstance(context).get();
// Build a Preview use case for displaying camera preview, configuring the necessary properties.
Preview preview = new Preview.Builder().build();
// Build a CameraSelector with the required lens facing, specifying to use the front camera.
CameraSelector cameraSelector = new CameraSelector.Builder().
requireLensFacing(CameraSelector.LENS_FACING_FRONT).build();
// Create an ImageAnalysis use case for analyzing camera frames, setting up the builder.
ImageAnalysis imageAnalysis = new ImageAnalysis.Builder().build();
// Set the analyzer for the ImageAnalysis to process each frame.
imageAnalysis.setAnalyzer(ContextCompat.getMainExecutor(context), image -> {
// Convert ImageProxy to Bitmap for processing
Bitmap bitmap = ImageUtils.imageProxyToBitmap(imageProxy);
// Invoke SSD model to detect faces and return detections
float[][] detections = SSDmodelLoader.detectFaces(bitmap);
// get the face from detections
Bitmap face = cropFace(bitmap,detections)
// Obtain the fatigue driving determination result.
result=FatigueDetector.analyzeFace(face);
// Hand result for alert information.
handResult(result)
imageProxy.close();
});
```

First, the code begins by synchronously obtaining the `ProcessCameraProvider`, a crucial step for managing the camera's lifecycle and ensuring that camera

operations are bound to the application's lifecycle. Following this, a 'Preview' use case is set up to display the camera's viewfinder, providing visual feedback to the user. The core functionality is extended by configuring an 'ImageAnalysis' use case, which is designed to process each frame captured by the camera. In the image analysis setup, each camera frame is processed by an analyzer, which includes converting the image proxy to a bitmap. and then using the SSD model and mobilenetV3 for fatigue detection .

For mobilenetV3 for fatigue detection, we further process the detected image by converting it into a tensor input vector with dimensions (145, 145, 3). This tensor is then fed into a pretrained MobileNetV3 model to assess fatigue status. The model evaluates the image data and outputs probabilities for various categories. We select and return the index of the category with the highest probability to the caller, to determine the fatigue status of the subject. This process ensures accurate and efficient fatigue detection. The code show as in follows.

```
public class FatigueDetector {
    private OnnxModelLoader modelLoader;

    public FatigueDetector() {
        // load the MobileNetV3 model
        modelLoader = new OnnxModelLoader("mobilenetv3_model.
onnx");
    }

    public int analyzeFace(Bitmap face) {
        // Resize and preprocess the face bitmap to match th
e input size expected by the model
        Bitmap resizedFace = resizeBitmap(face, 145, 145);
        float[][][] inputTensor = bitmapToTensor(resizedFace)
;        // Perform inference using the loaded model
        try { float[] predictions = modelLoader.predict(inputTensor)
;
            // Find the index of the maximum score in the predict
ions array
            return argMax(predictions);
        }
    }
}
```

For the detection results obtained, we further process them in the 'handleResult' method. If the result indicates fatigue (0) or the eyes have been closed for more than 10 seconds, a warning dialog will be displayed. The code show as in follows.

```

public class AlertSystem {
    // Method to display an alert and play a sound in response to
    // a fatigue detection event.
    public static void showAlert(Context context) {
        // Ensure the alert dialog and sound are run on the main UI
        // thread.
        new Handler(Looper.getMainLooper()).post(() -> {
            // Build and configure an alert dialog.
            AlertDialog.Builder builder = new AlertDialog.Builder(context);
            builder.setTitle("Warning"); // Set the title of the dialog.
            builder.setMessage("Fatigue or eye closure detected!"); // Set
            // the content message of the dialog.
            builder.setPositiveButton("OK", (dialog, which) ->
            // dialog.dismiss()); // Set a positive button with a click listener
            // to dismiss the dialog.
            AlertDialog alert = builder.create(); // Create the AlertDialog
            // from the builder.
            alert.show(); // Show the alert dialog on the screen.
            // Create and start a MediaPlayer to play an alert sound.
            MediaPlayer mediaPlayer = MediaPlayer.create(context, R.raw.
            alert_sound);
            mediaPlayer.start(); // Play the sound. }); } }

```

5.4 Performance Evaluation and Testing

5.4.1 Evaluating Driver Fatigue Detection Models

This project uses a series of key metrics to comprehensively evaluate the model's performance in detecting driver fatigue, including accuracy, recall, precision, F1 Score, and the confusion matrix.

Accuracy measures the overall performance of the model by calculating the proportion of correctly predicted samples out of the total. While it reflects how often the model predicts correctly, accuracy alone may not fully capture performance in cases of class imbalance (e.g., differing numbers of fatigue and non-fatigue samples). Therefore, additional metrics are needed for a more complete evaluation.

Recall assesses the model's ability to correctly identify all actual positive samples, focusing on its capacity to detect fatigue. A high recall indicates the model can

capture most true fatigue cases, minimizing the risk of missing potential fatigue instances, which is critical in practical applications.

Precision indicates the proportion of true positive predictions among all samples predicted as positive (fatigue). Higher precision means fewer false positives, reflecting greater accuracy in predicting fatigue. To balance precision and recall, the F1 Score, which is the harmonic mean of the two, provides a valuable metric for evaluating overall model performance, particularly on imbalanced datasets.

The confusion matrix offers a detailed view of the model's classification performance across different categories, showing the distribution of true positives, true negatives, false positives, and false negatives. By analyzing the confusion matrix, one can identify specific error types, such as false negatives or false positives, and optimize the model accordingly. For example, in fatigue detection, if false negatives (FN) are high, the model may need adjustments to improve recall and enhance overall detection accuracy.

5.4.2 Software Testing

After completing model testing, conducting a comprehensive software system test is a crucial step to ensure that the application meets all anticipated requirements. This project will utilize test cases for thorough functional and performance testing to validate the software system's capabilities. Below are the test cases and expected objectives for system-level functional and performance testing.

Real-time Video Detection

Test Case 1: Camera Access Permission Test

Objective: Verify that the application can successfully request and obtain camera usage permissions.

Steps: Upon first launch after installing the app, check if a camera permission request pops up. After permission is granted, observe whether the camera can be accessed.

Expected Result: The application should correctly request camera permissions and access the camera once the user grants permission.

Test Case 2: Face Detection Startup Test

Objective: Test whether the system can automatically detect and track faces once the camera is activated.

Steps: When a face appears within the camera's field of view, observe whether the system can recognize and track the face.

Expected Result: The system should quickly recognize and continuously track faces as soon as they appear.

Test Case 3: Real-time Performance Testing

Objective: To verify the real-time performance of face detection and tracking.

Steps: Test the system's response speed and tracking stability under various movement scenarios and facial expression changes.

Expected Result: Face detection and tracking should be performed in real-time with no noticeable delay, maintaining stability even during rapid movements or facial expression changes.

Fatigue Driving Detection

Test Case 4: Behavior Recognition Test

Objective: To verify whether the system can accurately identify key behaviors associated with fatigue driving, such as eye closure and yawning.

Steps: Input video sequences containing specific fatigue behaviors and check the system's recognition results.

Expected Result: The system should accurately identify and record all fatigue behaviors.

Test Case 5: Non-Fatigue Behavior Misjudgment Test

Objective: To ensure that the system does not mistakenly identify normal driving behaviors as fatigue driving.

Steps: Input videos of normal driving behavior and monitor whether the system falsely reports fatigue.

Expected Result: The system should not generate false positives.

Test Case 6: Fatigue Driving Detection Performance Test

Objective: To verify the system's performance and timeliness when processing real-time video and making fatigue judgments simultaneously.

Steps: Run video detection and fatigue judgment algorithms concurrently and observe the processing speed and system resource usage.

Expected Result: The system should efficiently handle multiple tasks without performance bottlenecks.

Alarm System

Test Case 7: Alarm Trigger Test

Objective: To test whether the system triggers the preset alarms when fatigue driving is detected.

Steps: When the system detects fatigue driving behavior, check if the corresponding visual, audio, and vibration alarms are triggered.

Expected Result: The alarms should be triggered according to the preset conditions without any delay.

Test Case 8: Alarm Response Time Test

Objective: To measure the time delay between the detection of fatigue driving and the actual triggering of the alarm.

Steps: After the system detects a fatigue state, time the delay until the alarm is triggered.

Expected Result: The alarm should be triggered within a few seconds to ensure that the driver receives timely warnings.

CHAPTER 6- EXPERIMENT AND RESULT ANALYSIS

This chapter aims to comprehensively present and analyze the experimental results, focusing on two key aspects: the deep detection model and the software system. First, the chapter will provide a detailed overview of the experimental results related to the deep detection model, including its performance metrics and detection accuracy on the dataset. Following this, the analysis will delve into the experimental data of the software system, utilizing the testing methods and cases outlined in Section 4.3. The focus will be on key indicators such as system response time, stability, and accuracy. By discussing the results from these two perspectives, this chapter will further elucidate the successes of the experiments and potential areas for improvement, providing valuable insights for future research.

6.1 Dataset Effectiveness Evaluation

As shown in Figure 6, we examined the balance of the dataset. Based on the bar chart and statistical results, we can see that the dataset is perfectly balanced between the "Yawn" and "No Yawn" categories. Specifically, each category contains 724 images, accounting for 50.00% of the total dataset. This balance is crucial for training the deep detection model, as it helps prevent the model from being biased towards one particular category, thereby enhancing the model's generalization ability in real-world applications. Therefore, the current balance of the dataset provides a solid foundation for the subsequent experiments, contributing to the fairness and accuracy of the model.

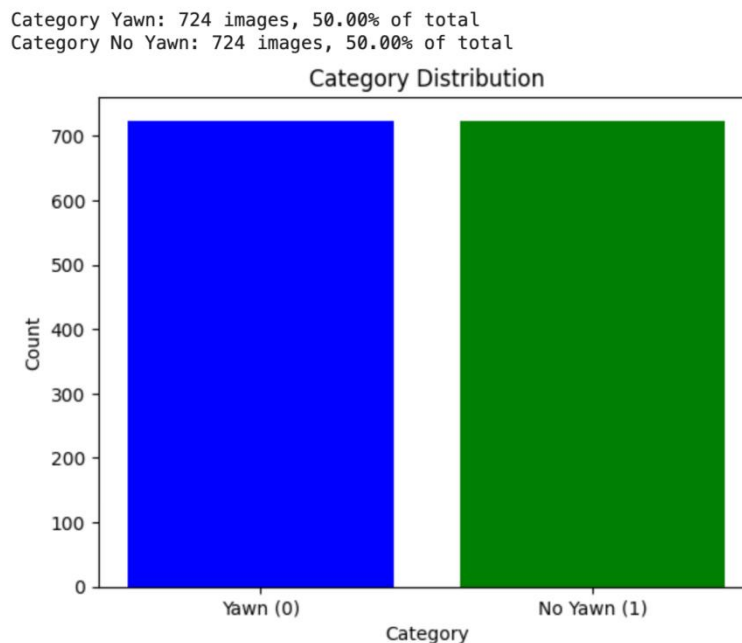


Figure 6. Dataset balance distribution

6.2 Face Detection Evaluation

As shown in Figure 7, the SSD model demonstrates excellent detection capabilities in identifying facial features related to yawning and non-yawning. The model successfully and accurately detects and labels all faces in the image, with clear and precise bounding boxes surrounding each face. This indicates that the SSD model is capable of effectively detecting faces and accurately distinguishing between yawning and non-yawning behaviors, even under varying angles, lighting conditions, and facial expressions. The model's performance on the dataset is

impressive, showcasing its robustness and accuracy in complex scenarios. This outcome lays a solid foundation for further advancements in fatigue detection for driving.



Figure 7. Face Detection Results

6.3 Fatigue detection Evaluation

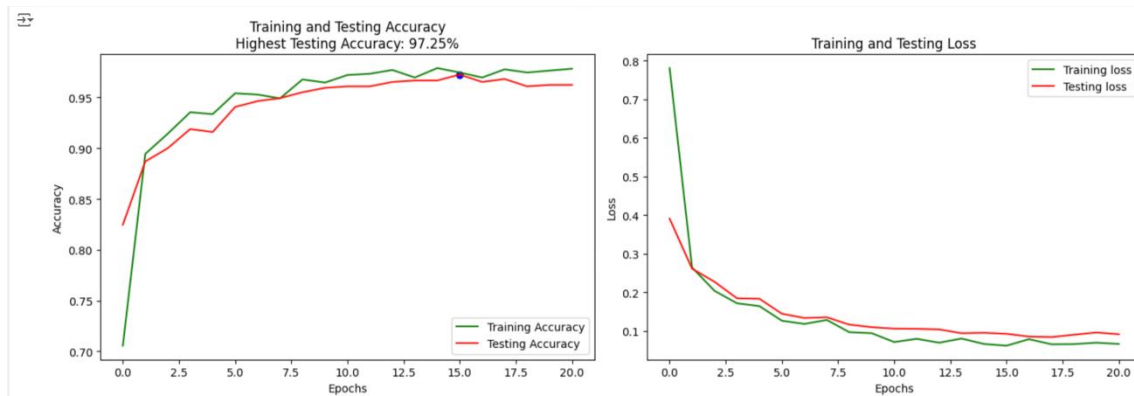


Figure 8. Training and Testing Accuracy and loss

The figure 8 indicate that the MobileNetV3 model performs exceptionally well on the YAWDD dataset during both training and testing. The graph on the left illustrates the trend of training accuracy and testing accuracy as the number of epochs increases. It can be observed that both accuracies rise rapidly in the initial stages and stabilize around the 15th epoch, with the highest testing accuracy reaching 97.25%. This demonstrates that the model effectively learns features during training and achieves high classification accuracy on the test set.

The graph on the right shows the changes in training loss and testing loss over the epochs. Both training and testing losses decrease rapidly and stabilize in the later stages of training, indicating that the model gradually converges and that

overfitting is well-controlled. Overall, the small gap between training loss and testing loss suggests that the model has good generalization ability.

In summary, the MobileNetV3 model exhibits excellent performance on the YAWDD dataset, achieving high classification accuracy while maintaining low loss values, making it suitable for tasks related to this dataset.

22/22	6s 142ms/step			
	precision	recall	f1-score	support
yawn	0.97	0.95	0.96	230
no_yawn	0.94	0.97	0.95	205
Closed	1.00	1.00	1.00	120
Open	1.00	1.00	1.00	135
accuracy			0.97	690
macro avg	0.98	0.98	0.98	690
weighted avg	0.97	0.97	0.97	690

Figure 9. Performance Metrics

As shown in Figure 9, The model demonstrates balanced performance across all four categories, with high overall precision, recall, and F1 scores.

1. Categories "yawn" and "no_yawn": For the "yawn" category, the model achieves a precision of 0.97, a recall of 0.95, and an F1 score of 0.96. For the "no_yawn" category, the precision is 0.94, recall is 0.97, and the F1 score is 0.95. This indicates that the model performs excellently in distinguishing between yawning and non-yawning states.

2. Categories "Closed" and "Open": In the "Closed" (eyes closed) and "Open" (eyes open) categories, the model achieves a perfect precision, recall, and F1 score of 1.00. This means the model has achieved flawless classification in recognizing closed and open eyes, with no errors or omissions.

3. Overall Performance: The model's overall accuracy is 0.97, with macro and weighted averages of precision, recall, and F1 scores close to 0.98 and 0.97. This

indicates that the model is stable and accurate across the entire test set, with no significant deviation in performance between different categories.

In summary, the model performs exceptionally well on the YAWDD dataset, especially in distinguishing between open and closed eyes. It also shows good performance in classifying yawning versus non-yawning, making it suitable for fatigue detection applications.

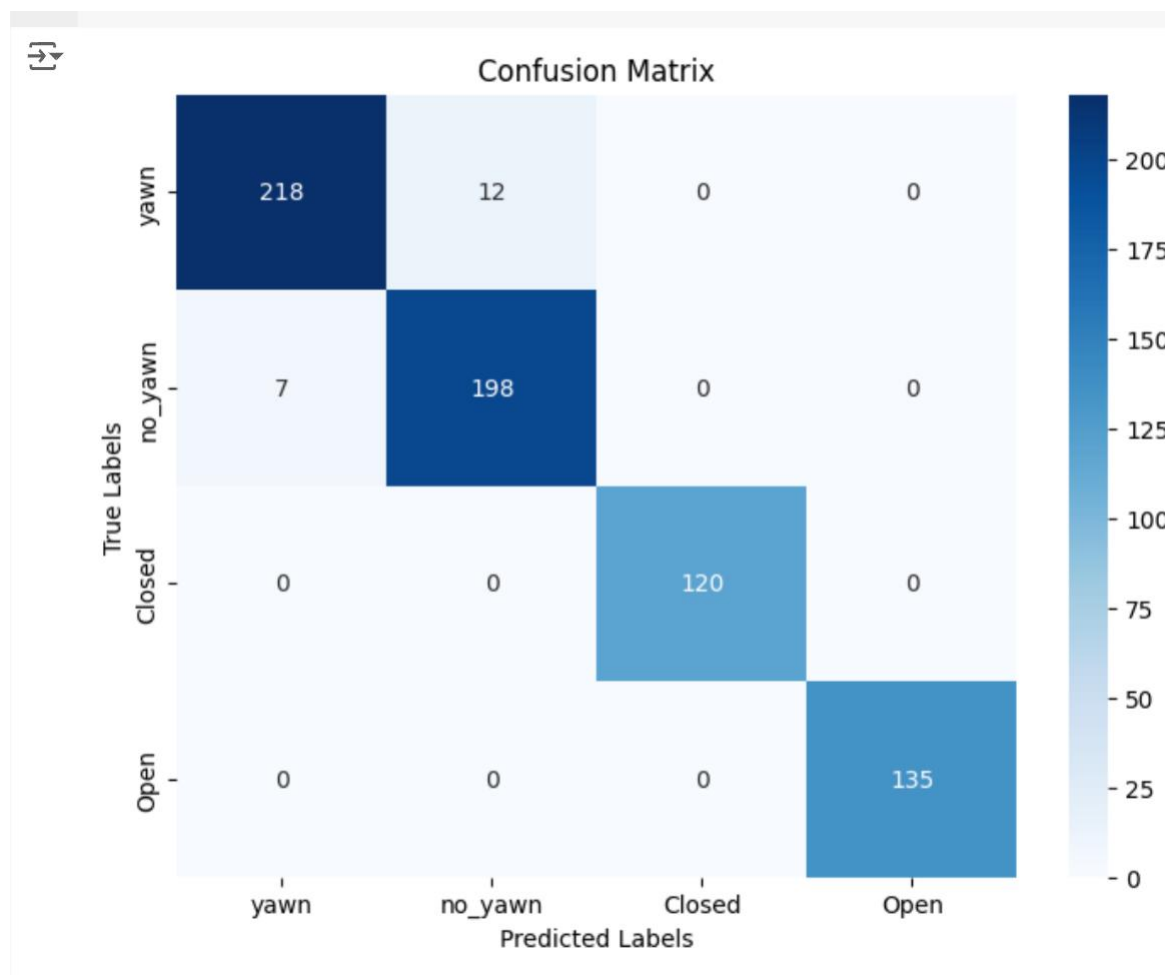


Figure 10. confusion matrix

The confusion matrix in Figure 10 reveals that the model performs exceptionally well in classifying the "Closed" (eyes closed) and "Open" (eyes open) categories, correctly classifying all 120 "Closed" samples and 135 "Open" samples, achieving perfect classification accuracy with no misclassifications. Additionally, the model demonstrates strong classification capabilities in the "yawn" and "no_yawn"

categories, accurately distinguishing between the majority of yawning and non-yawning states. These results indicate that the model excels in recognizing tasks across various categories, particularly achieving ideal precision in eye state classification, thereby laying a solid foundation for further practical applications.

6.4 Android Application Testing

This section will conduct software system testing based on the test cases for functionality and performance outlined in Section 4.4, in order to comprehensively evaluate the effectiveness of the developed fatigue driving detection system.

Firstly, Test Case 1 and Test Case 2 consist of two parts: camera permission request testing and facial recognition testing. The objective is to verify whether the software can accurately request and obtain camera usage permission and recognize faces effectively. According to the figure 11 results, we successfully requested camera usage permission and used the front-facing camera for facial recognition.

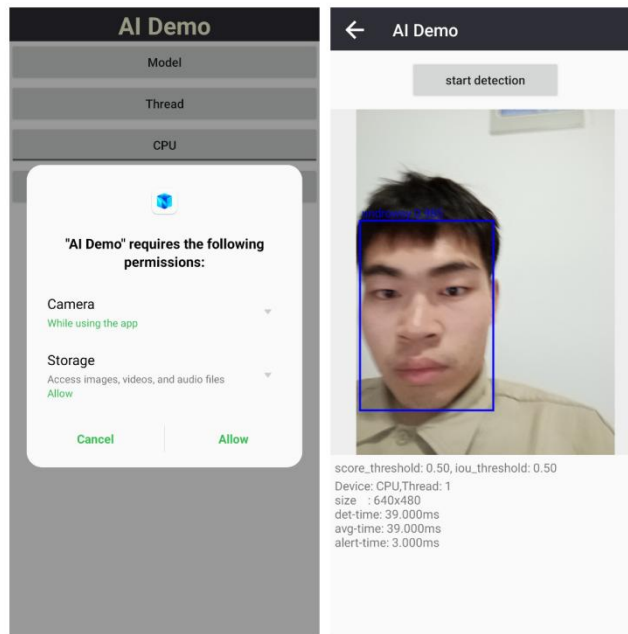


Figure 11. Test Case 1 Results

Test Case 3 focuses on adaptability testing, aiming to evaluate the system's performance in accurately and real-time recognizing faces under different angles and background scenes. As Figure 12 illustrated, we tested facial recognition

effectiveness from four different angles. The results demonstrate that the system accurately identifies faces across various angles.

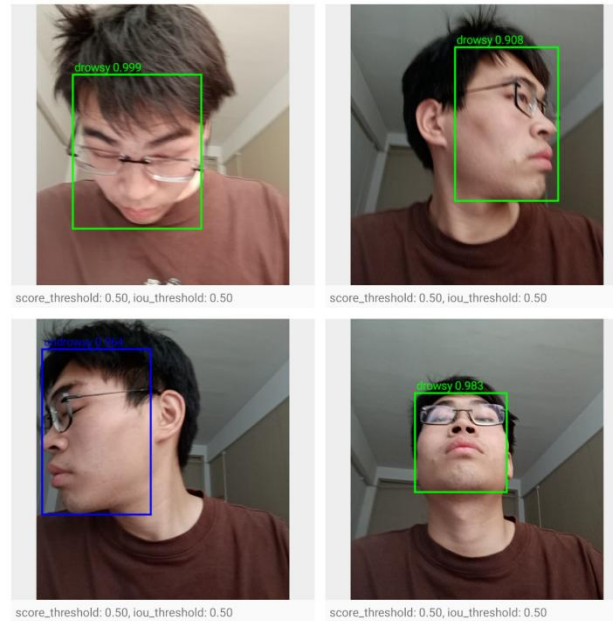


Figure 12. Face Detection of Different Angle

Test Cases 4, 5, and 6 involve fatigue driving behavior recognition, non-fatigue behavior misclassification, and real-time performance testing. The aim is to assess whether the system can accurately determine fatigue and non-fatigue driving states, as well as to evaluate the system's performance and timeliness when processing real-time video and making fatigue assessments simultaneously.

As shown in Figures 13, the system correctly identifies both fatigue and non-fatigue states through test images and real-time video. Additionally, the average inference time is 3 milliseconds. These results demonstrate that we have met the testing requirements effectively.

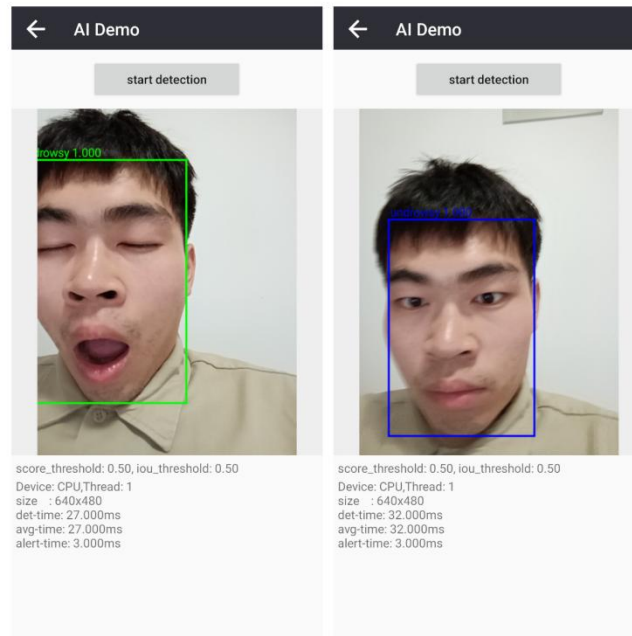


Figure 13. Fatigue Detection Results

Test Cases 7 and 8 aim to evaluate whether the system triggers the preset alarms upon detecting fatigue driving and to measure the time delay between detecting fatigue driving and the actual triggering of the alarm, in order to assess the timeliness of the system's alarm response.

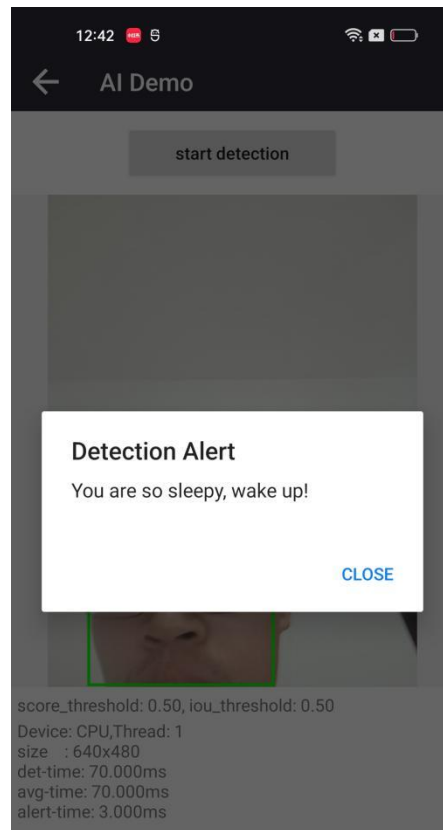


Figure 14. Alret system

The results from the figure 14 indicate that once the fatigue driving trigger condition (continuous fatigue state for 3 seconds) is met, the warning box is accurately displayed. Additionally, the time delay from detecting the fatigue state to displaying the warning is only 1 milliseconds. This demonstrates that the requirements have been effectively met.

CHAPTER 7- CONCLUSIONS AND RECOMMENDATIONS

This study successfully developed a driver fatigue monitoring system based on the Android platform, integrating SSD (Single Shot MultiBox Detector) and MobileNetV3 deep learning models. The system is characterized by its real-time, non-intrusive nature and is implemented through the widespread deployment of smartphone hardware and software features. Experimental validation on the YawDD dataset demonstrated that the system performs exceptionally well in detecting driver fatigue, achieving a testing accuracy of 97.25%. Additionally, the system exhibited excellent real-time performance and response speed, ensuring

high accuracy while effectively reducing false alarm rates, thereby enhancing the user experience.

During the research process, we first analyzed various fatigue detection methods and chose vision-based facial detection technology as the primary research direction. A lightweight MobileNetV3 model was adopted to address hardware limitations in older vehicles. The results indicated that the combination of SSD and MobileNetV3 offers significant advantages in processing real-time video streams and accurately identifying drivers' facial features. Furthermore, by utilizing Android devices and their open development environment, this system effectively solves the challenge of deploying complex detection systems in older vehicles, providing a low-cost, high-efficiency fatigue monitoring solution. This study not only made significant progress in algorithm design and model optimization but also provided easily operable and deployable solutions in actual deployment and user interface design, offering strong support for future research in driving safety monitoring.

Despite the achievements of this study, there remain some challenges and areas for improvement in practical applications. First, the model can be further optimized to adapt to more complex driving environments, particularly by exploring the integration of multimodal data such as sound and heart rate to enhance detection accuracy and robustness. Second, given the varying hardware performance of Android devices, it is recommended to conduct extensive testing and optimization across a broader range of devices to ensure stable operation under different hardware conditions. Finally, the user experience can be further enhanced, especially by simplifying user operations and optimizing the alert mechanism to reduce false alarm rates and improve user trust and satisfaction.

CHAPTER 8- REFERENCES

- [1] B. K. Savaş and Y. Becerikli, "Real Time Driver Fatigue Detection System Based on Multi-Task ConNN," in **IEEE Access**, vol. 8, pp. 12491-12498, 2020, doi: 10.1109/ACCESS.2020.2963960.
- [2] Rajat Garg, Vikrant Gupta, and Vineet Agrawal, "A Drowsy Driver Detection and Security System," in **IEEE International Conference on Ultra Modern Telecommunications & Workshops**, 2009.
- [3] F. Wang, D. Chen, B. Lu, H. Wang, and R. Fu, "A Novel Semi-Dry Electrode Suitable for Collecting EEG Data for Detecting Driving Fatigue in Long-Period Driving Case," in **IEEE Sensors Journal**, vol. 23, no. 16, pp. 17891-17900, Aug. 15, 2023, doi: 10.1109/JSEN.2023.3292499.
- [4] Z. Zhang, H. Ning, and F. Zhou, "A systematic survey of driving fatigue monitoring," **IEEE Transactions on Intelligent Transportation Systems**, vol. 23, no. 11, pp. 19999-20020, Nov. 2022.
- [5] Y. Albadawi, M. Takruri, and M. Awad, "A review of recent developments in driver drowsiness detection systems," **Sensors**, vol. 22, no. 5, p. 2069, Mar. 2022.
- [6] G. Sikander and S. Anwar, "Driver Fatigue Detection Systems: A Review," in **IEEE Transactions on Intelligent Transportation Systems**, vol. 20, no. 6, pp. 2339-2352, June 2019, doi: 10.1109/TITS.2018.2868499.
- [7] M. Ramzan, H.U. Khan, S.M. Awan, A. Ismail, M. Ilyas, and A. Mahmood, "A survey on state-of-the-art drowsiness detection techniques," in **IEEE Access**, vol. 7, pp. 61904–61919, 2019.

- [8] G. Li, B. -L. Lee, and W. -Y. Chung, "Smartwatch-Based Wearable EEG System for Driver Drowsiness Detection," in **IEEE Sensors Journal**, vol. 15, no. 12, pp. 7169-7180, Dec. 2015, doi: 10.1109/JSEN.2015.2473679.
- [9] R. Kaur and K. Singh, "Drowsiness detection based on EEG signal analysis using EMD and trained neural network," **Int. J. Sci. Res.**, vol. 10, pp. 157–161, 2013.
- [10] U. Budak, V. Bajaj, Y. Akbulut, O. Atila, and A. Sengur, "An Effective Hybrid Model for EEG-Based Drowsiness Detection," in **IEEE Sensors Journal**, vol. 19, no. 17, pp. 7624-7631, Sept. 1, 2019, doi: 10.1109/JSEN.2019.2917850.
- [11] S. Taran and V. Bajaj, "Drowsiness Detection Using Adaptive Hermite Decomposition and Extreme Learning Machine for Electroencephalogram Signals," in **IEEE Sensors Journal**, vol. 18, no. 21, pp. 8855-8862, Nov. 1, 2018, doi: 10.1109/JSEN.2018.2869775.
- [12] S. K. Khare and V. Bajaj, "Entropy-Based Drowsiness Detection Using Adaptive Variational Mode Decomposition," in **IEEE Sensors Journal**, vol. 21, no. 5, pp. 6421-6428, March 1, 2021, doi: 10.1109/JSEN.2020.3038440.
- [13] B.G. Pratama, I. Ardiyanto, and T.B. Adji, "A review on driver drowsiness based on image, bio-signal, and driver behavior," in **Proceedings of the 2017 3rd International Conference on Science and Technology-Computer (ICST)**, Bandung, Indonesia, Oct. 25–26, 2017, pp. 70–75.
- [14] Y. Saito, M. Itoh, and T. Inagaki, "Driver Assistance System With a Dual Control Scheme: Effectiveness of Identifying Driver Drowsiness and Preventing Lane Departure Accidents," in **IEEE Transactions on Human-Machine Systems**, vol. 46, no. 5, pp. 660-671, Oct. 2016, doi: 10.1109/THMS.2016.2549032.
- [15] A.D. McDonald, C. Schwarz, J.D. Lee, and T.L. Brown, "Real-time detection of drowsiness related lane departures using steering wheel angle," in

Proceedings of the Human Factors and Ergonomics Society Annual Meeting, Boston, MA, USA, Oct. 22–26, 2012, pp. 2201–2205.

[16] S. Arefnezhad, S. Samiee, A. Eichberger, and A. Nahvi, "Driver drowsiness detection based on steering wheel data applying adaptive neuro-fuzzy feature selection," **Sensors**, vol. 19, p. 943, 2019.

[17] M. Ramzan, H.U. Khan, S.M. Awan, A. Ismail, M. Ilyas, and A. Mahmood, "A survey on state-of-the-art drowsiness detection techniques," in **IEEE Access**, vol. 7, pp. 61904–61919, 2019.

[18] F. You, Y. Gong, H. Tu, J. Liang, and H. Wang, "A fatigue driving detection algorithm based on facial motion information entropy," **J. Adv. Transp.**, vol. 2020, p. 1–17, 2020.

[19] M.T. Khan, H. Anwar, F. Ullah, A. Ur Rehman, R. Ullah, A. Iqbal, B.-H. Lee, and K.S. Kwak, "Smart real-time video surveillance platform for drowsiness detection based on eyelid closure," **Wirel. Commun. Mob. Comput.**, vol. 2019, p. 1–9, 2019.

[20] SHRP2: Transportation Research Board of the National Academies of Science. **The 2nd Strategic Highway Research Program Naturalistic Driving Study Dataset**; Virginia Tech Transportation Institute: Blacksburg, VA, USA, 2018. Available online: <https://insight.shrp2nds.us/> (accessed on 20 September 2021).

[21] M. Hashemi, A. Mirrashid, and A.B. Shirazi, "Driver Safety Development: Real-Time Driver Drowsiness Detection System Based on Convolutional Neural Network," **SN Comput. Sci.**, vol. 1, p. 1–10, 2020.

[22] Backlinko. (n.d.). **iPhone vs Android Statistics: The Complete Guide (2022)**. Backlinko. <https://backlinko.com/iphone-vs-android-statistics>. Accessed: May 2, 2024.

- [23] K.W. Kim, H.G. Hong, G.P. Nam, and K.R. Park, "A Study of Deep CNN-Based Classification of Open and Closed Eyes Using a Visible Light Camera Sensor," **Sensors**, vol. 17, no. 7, p. 1534, Jul. 2017, doi: 10.3390/s17071534.
- [24] L. Zhao, Z. Wang, G. Zhang, et al., "Eye state recognition based on deep integrated neural network and transfer learning," **Multimedia Tools and Applications**, vol. 77, pp. 19415-19438, 2018.
- V.R.R. Chirra, S.R. Uyyala, and V.K.K. Kolli, "Deep CNN: A Machine Learning Approach for Driver Drowsiness Detection Based on Eye State," **Rev. d'Intelligence Artif.**, vol. 33, no. 6, pp. 461-466, 2019.
- [26] S. Liu, L. Yu, and M. Hou, "An efficient method for driver fatigue state detection based on deep learning," in **Proceedings of the 2019 2nd International Conference on Safety Produce Informatization (IICSPI)**, 2019, pp. 172-176.
- [27] Z. Hao, G. Wan, Y. Tian, et al., "Research on driver fatigue detection method based on parallel convolution neural network," in **Proceedings of the 2019 IEEE International Conference on Power, Intelligent Computing and Systems (ICPICS)**, 2019, pp. 164-168.
- [28] Z. Xiao, Z. Hu, L. Geng, et al., "Fatigue driving recognition network: fatigue driving recognition via convolutional neural network and long short-term memory units," **IET Intelligent Transport Systems**, vol. 13, no. 9, pp. 1410-1416, 2019.
- [29] S.B. Kose and Y. Becerikli, "Real time driver fatigue detection system based on multi-task ConNN," **IEEE Access**, vol. 8, pp. 12491-12498, 2020.
- [30] S. Ansari, F. Naghdy, H. Du, et al., "Driver mental fatigue detection based on head posture using new modified ReLU-BiLSTM deep neural network," **IEEE Transactions on Intelligent Transportation Systems**, vol. 23, no. 8, pp. 10957-10969, Aug. 2022.

- [31] Z. Zhao, N. Zhou, L. Zhang, et al., "Driver fatigue detection based on convolutional neural networks using em-CNN," **Computational Intelligence and Neuroscience**, vol. 2020, no. 1, pp. 1-10, 2020, Art. no. 7251280.
- [32] A. Kumar, Z.J. Zhang, and H. Lyu, "Object detection in real time based on improved single shot multi-box detector algorithm," **EURASIP Journal on Wireless Communications and Networking**, vol. 2020, no. 1, p. 204, 2020.
- [33] A. Kumar and S. Srivastava, "Object detection system based on convolution neural networks using single shot multi-box detector," **Procedia Computer Science**, vol. 171, pp. 2610-2617, 2020.
- [34] H. Pan, J. Jiang, and G. Chen, "TDFSSD: Top-down feature fusion single shot MultiBox detector," **Signal Processing: Image Communication**, vol. 89, p. 115987, 2020.
- [35] S. Qian, C. Ning, and Y. Hu, "MobileNetV3 for image classification," in **Proceedings of the 2021 IEEE 2nd International Conference on Big Data, Artificial Intelligence, and Internet of Things Engineering (ICBAIE)**, 2021, pp. 490-497.
- [36] L. Zhao and L. Wang, "A new lightweight network based on MobileNetV3," **KSII Transactions on Internet and Information Systems (TIIS)**, vol. 16, no. 1, pp. 1-15, 2022.
- [37] H. Wang, V. Bhaskara, A. Levinshtein, et al., "Efficient super-resolution using MobileNetV3," in **Proceedings of Computer Vision–ECCV 2020 Workshops**, Glasgow, UK, Aug. 23–28, 2020, vol. 16, Springer International Publishing, pp. 87-102.
- [38] K. Dong, C. Zhou, Y. Ruan, et al., "MobileNetV2 model for image classification," in **Proceedings of the 2020 2nd International Conference on Information Technology and Computer Application (ITCA)**, 2020, pp. 476-480.

- [39] L. Hao, Q. Weigen, and Z. Lichen, "Improved ShuffleNet V2 for Lightweight Crop Disease Identification," **Journal of Computer Engineering & Applications**, vol. 58, no. 12, 2022.
- [40] M.N. Ab Wahab, A. Nazir, A.T.Z. Ren, et al., "EfficientNet-Lite and hybrid CNN-KNN implementation for facial expression recognition on Raspberry Pi," **IEEE Access**, vol. 9, pp. 134065-134080, 2021.
- [41] S.B.R. Prasad and B.S. Chandana, "MobileNetV3: A deep learning technique for human face expressions identification," **International Journal of Information Technology**, vol. 15, no. 6, pp. 3229-3243, 2023.
- [42] H. Elahi, G. Wang, and J. Chen, "Pleasure or pain? An evaluation of the costs and utilities of bloatware applications in Android smartphones," **Journal of Network and Computer Applications**, vol. 157, p. 102578, 2020.
- [43] G. Labrèche, D. Evans, D. Marszk, et al., "OPS-SAT spacecraft autonomy with TensorFlow Lite, unsupervised learning, and online machine learning," in **Proceedings of the 2022 IEEE Aerospace Conference (AERO)**, 2022, pp. 1-17.
- [44] T.C. Wee and K.N.C. Kee, "On-device image labeling photo management system using Flutter and ML Kit," in **Proceedings of the International Conference on Intelligent Technologies for Interactive Entertainment**, Cham, Switzerland, 2021, pp. 208-222, Springer International Publishing.
- [45] F. Manca, F. Ratto, and F. Palumbo, "ONNX-to-hardware design flow for adaptive neural-network inference on FPGAs," **arXiv preprint arXiv:2406.09078**, 2024.
- [46] H. Ham, W. Yang, Y. Shin, et al., "ONNXim: A fast, cycle-level multi-core NPU simulator," **arXiv preprint arXiv:2406.08051**, 2024.
- [47] Y.S. Park, L. Konge, and A.R. Artino Jr, "The positivism paradigm of research," **Acad. Med.**, vol. 95, no. 5, pp. 690-694, 2020.

- [48] H.H. Alharahsheh and A. Pius, "A review of key paradigms: Positivism vs interpretivism," **Global Acad. J. Humanities Social Sci.**, vol. 2, no. 3, pp. 39-43, 2020.
- [49] U.M. Mbanaso, L. Abrahams, and K.C. Okafor, "Research philosophy, design and methodology," in **Research Techniques for Computer Science, Information Systems and Cybersecurity**, Cham: Springer Nature Switzerland, 2023, pp. 81-113.
- [50] S. Grinchenko and Y.L. Shchapova, "The deductive approach to Big History's Singularity," in **The 21st Century Singularity and Global Futures: A Big History Perspective**, 2020, pp. 201-210.
- [51] P. Mishra and S. Mishra, "A deductive approach to modeling the spread of COVID-19," **medRxiv**, 2020, doi: 10.1101/2020.03.26.20044651.
- [52] S. Niaz, T. Salam, and S. Nazir, "Statistical methods in computer science: A comparative review of inductive and deductive approaches," **VAWKUM Trans. Comput. Sci.**, vol. 11, no. 1, pp. 204-216, 2023.
- [53] M.K. Jamieson, G.H. Govaart, and M. Pownall, "Reflexivity in quantitative research: A rationale and beginner's guide," **Soc. Personal. Psychol. Compass**, vol. 17, no. 4, p. e12735, 2023.
- [54] T. Ellis and Y. Levy, "A Guide for Novice Researchers: Design and Development Research Methods," **IEEE Transactions on Professional Communication**, vol. 53, no. 2, pp. 189-200, 2010.
- [55] S. Al-Saqqa, S. Sawalha, and H. AbdelNabi, "Agile software development: Methodologies and trends," **Int. J. Interactive Mobile Technol.**, vol. 14, no. 11, 2020.

[56] M. Hron and N. Obwegeser, "Why and how is Scrum being adapted in practice: A systematic review," **J. Systems Software**, vol. 183, p. 111110, 2022.

[57] S. Modi, H.A. Taher, and H. Mahmud, "A tool to automate student UML diagram evaluation," **Acad. J. Nawroz Univ.**, vol. 10, no. 2, pp. 189-198, 2021.

[58] T. Arianti, A. Fa'izi, S. Adam, et al., "Perancangan sistem informasi perpustakaan menggunakan diagram UML (Unified Modelling Language)," **J. Ilmiah Komputer Terapan dan Informasi**, vol. 1, no. 1, pp

. 19-25, 2022.

[59] J.H. Zhang, B. Meng, L.C. Zou, et al., "Progressive flowchart development scaffolding to improve university students' computational thinking and programming self-efficacy," **Interact. Learn. Environ.**, vol. 31, no. 6, pp. 3792-3809, 2023.

[60] É. André, S. Liu, Y. Liu, et al., "Formalizing UML state machines for automated verification—A survey," **ACM Comput. Surv.**, vol. 55, no. 13s, pp. 1-47, 2023.

[61] J. Shore and S. Warden, **The Art of Agile Development**, 2nd ed. O'Reilly Media, Inc., 2021.

[62] A.C. Sassa, I.A. de Almeida, T.N.F. Pereira, et al., "Scrum: A systematic literature review," **Int. J. Adv. Comput. Sci. Appl.**, vol. 14, no. 4, 2023.

[63] S. Supriyono, "Software testing with the approach of blackbox testing on the academic information system," **Int. J. Inf. Syst. Technol. (IJISTECH)**, vol. 3, no. 2, pp. 227-233, 2020.

[64] T.Y. Chen, S.C. Cheung, and S.M. Yiu, "Metamorphic testing: a new approach for generating next test cases," **arXiv preprint arXiv:2002.12543**, 2020.

[65] C. Qian, X. Cong, C. Yang, et al., "Communicative agents for software development," *arXiv preprint arXiv:2307.07924*, 2023.

[66] S. Abtahi, M. Omidyeganeh, S. Shirmohammadi, and B. Hariri, "YawDD: Yawning Detection Dataset", *IEEE Dataport*, Aug. 1, 2020, doi: <https://dx.doi.org/10.21227/e1qm-hb90>.

PROJECT MANAGEMENT

1)Project Scope

Project Aims: Develop and deploy a system that can detect driver fatigue in real-time to reduce the rate of traffic accidents.

Deliverables: An Android Application with Driving Fatigue Detection System

key milestones: 1. Training and testing validation of the SSD MobileNetV2 models completed. 2. Integration of models into Android devices. 3. Verification that the Android version of the fatigue driving detection system can reliably provide services.

Time management tools: Gantt Chart、 A Gantt chart, commonly used in project management, is one of the most popular and useful ways of showing activities (tasks or events) displayed against time [36]. To the left of the diagram, there is a list of activities, and across the top, a corresponding time scale is displayed. Each activity is represented by a bar, with its placement and length indicating the activity's start date, duration, and completion date.

2)Time Line and Gantt Chart

The timeline is roughly scheduled as follows, Background Reading from February 10 to February 24, Development Proposal, from February 17 and March 6, Review of Literature , from March 2 to April 4, Research Design and Methodology, from April 13 to May 4, Data Collection and Experiment, from May 11 to June 15, Experiment Analysis and Update, from June 8 to July 10, Paper Writing, from June 22 to September 7. Also The time Line shows in the Gantt Chart of Figure 12.

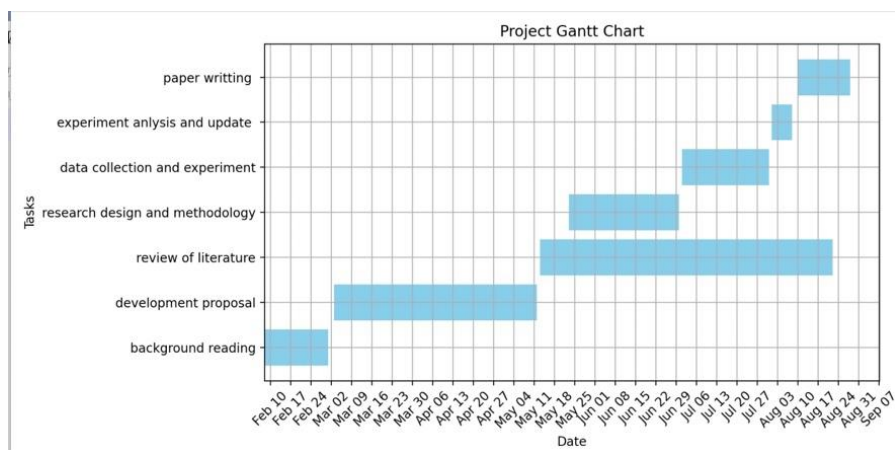


Figure 15. Gantt Chart

3)Communications

During the process of completing this research, it is crucial to maintain active communication with supervisor. Regular updates on research progress and discussions about any issues or challenges encountered help ensure the correctness of the research direction and methodology. Moreover, it is essential to strictly adhere to ethical standards and legal regulations throughout the research process. This includes, but is not limited to, ensuring the lawful collection and use of data, respecting the rights of research subjects, and adhering to principles of integrity when publishing and sharing research findings. By doing so, not only can the scientific rigor and standardization of the research be enhanced, but the effectiveness and reliability of the research results can also be ensured, laying a solid foundation for future studies.

