Distraction Detection System Using Raspberry Pi: A Video Detection Approach Using YoloV8

A master's dissertation

by

Nikhilesh Kovvuri

P2821317

Submitted in partial fulfillment of the requirements for the Degree of MSc in Artificial Intelligence

School of Computing

De Montfort University, Dubai

November 2024

Declaration

I declare that other than where specifically mentioned the work of others, the work presented herein is original, and has not been submitted elsewhere, in full or in part, in return for any academic degree or qualification at this or other university. This dissertation consists of my own work, except where more indicated in the text (including AI). In accordance with the copyright acts of the UK as amended by the De Montfort University regulations, this dissertation is the copyright of the author. Any use or reference of any materials contained in or derived from this dissertation requires that the author be properly acknowledged.

Nikhilesh Kovvuri

Acknowledgement

I had wanted to enroll in MSc in Artificial Intelligence at De Montfort university. I must say that I had amazing advice from both classmates and professors any time I needed it. I would want to express my thanks to Dr. Michael Gallimore the Head of the Campus and my supervisor, Dr. Muhammad Ghalib, for his continuous support of the project. They significantly reduced the length and complexity of a big project and made it possible so I cannot say enough thank you to them. Everyone who supported this project was vital in supporting it. Thank you for all you do.

Nikhilesh Kovvuri

LIST OF FIGURES

LIST OF FIGURES (Continuation)

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Expansion
YOLO	You Only Look Once
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
ML	Machine Learning
DL	Deep Learning
FPS	Frames Per Second
PiCam	Raspberry Pi Camera
BGR	Blue Green Red (Color Space in OpenCV)
F1-Score	Harmonic Mean of Precision and Recall
VGG	Visual Geometry Group
TPU	Tensor Processing Unit

ABSTRACT

Studying Driver Distraction, the foremost cause of road accidents, which poses serious risks to all road users, and the need for effective detection systems. Recent machine learning techniques have exhibited potential in distinguishing driver distractions, but overall, they depend on equipment that is both high performance and expensive, hence not viable for broad, real-world deployments. To fill this gap, we propose a real time driver distraction detection system in this study, which is designed for low-cost hardware, namely a Raspberry Pi with an integrated camera, to be practically deployed in normal vehicles. I use a mixed model in this work by combining the powerful object detection power of YOLOv8 with the classification strength of EfficientNet convolutional neural network (CNN).

In well-lit conditions, YOLOv8 achieves a face detection accuracy of over 95% and in low light settings, YOLOv8 maintains an accuracy of 85% even with partial obstructions. EfficientNet classifier is then applied to the detected face to locate explicitly distracted behaviors, e.g., texting, talking on the phone, and reaching for objects inside the vehicle, with 92% average accuracy on all tested behaviors. These models are combined to achieve rapid detection with an end-to-end processing latency of 70 milliseconds empowering real time capability for continuous driver monitoring. Additionally, the efficient processing of video by OpenCV makes it possible to seamlessly integrate face and distraction detection with the system and to further optimize computational resources powered by the Raspberry Pi. Unlike existing rigid systems whose performance is usually constrained by static lighting and by perfect environmental conditions, the proposed model dynamically adapts to different vehicle lighting, camera view and posture of the driver. The robustness is particularly relevant in environments such as in vehicle, where the conditions vary frequently greatly reducing the accuracy of traditional approaches.

The system will be examined extensively in live driving environments and the evaluation metrics will include detection accuracy, response time, processing latency and scalability. This model can achieve practically reversible speedup that allows up to 30% reduction in distraction induced accidents through early detection and intervention without paying much in increased computational or detection performance loss. The results of this research will be valuable for the field driver monitoring and machine learning in transportation specifically as a practical, low cost, way to improve road safety.

This work seeks to bridge the gap between realistic driver monitoring systems capabilities and real-world feasibility, which is demonstrated through a scalable approach applicable in a large set of vehicle types. Further potential directions include further fine tuning of the model, possible integration with other driver monitoring technology, and adapting to alternative low-cost hardware such as the NVIDIA Jetson Nano to bring increased accuracy to the model and broaden applicability.

Keywords: Driver Distraction, Machine Learning, YOLOv8, EfficientNet, Raspberry Pi, Real-Time Detection, Driver Monitoring, Face Detection, Low-Cost Hardware, Transportation Safety

TABLE OF CONTENTS **Page No.**

CHAPTER 1

INTRODUCTION

1.1 Background of Distraction Detection

Road accidents that involve driver distraction represent important risks to drivers and other roadway users. A driver's distraction from the primary task of driving (which is to watch the road) results in delayed response times, poor judgement, and, eventually, more crashes. Mobile phones, in car infotainment systems and other activities that steal visual, cognitive and manual attention away from the driver are common sources of distraction. According to the Centers for Disease Control and Prevention (CDC), thousands of lives are lost every year due to distracted driving, highlighting the urgent need for effective intervention methods According to the U.S. Centers for Disease Control and Prevention (2014), intoxicated driving is now second to only texting while driving as the greatest location of distracted driving (Distracted Driving | Distracted Driving | CDC).

Driver distraction is an important aspect in personal safety issues and is regarded as an important social issue which will cause major attendant economic ramifications. Driver distraction already accounts for many motor vehicle fatalities in the United States alone. A look at Figure 1.1 shows that the largest group of these fatal accidents involve the occupant of a passenger vehicle, a pedestrian, or a motorcyclist. More than fifty percent of crash associated fatalities are passenger vehicle occupants, followed by those of pedestrians and motorcyclists. This data supports the view that distracted driving affects an array of road users outside the vehicle as well (Forbes, 2022).

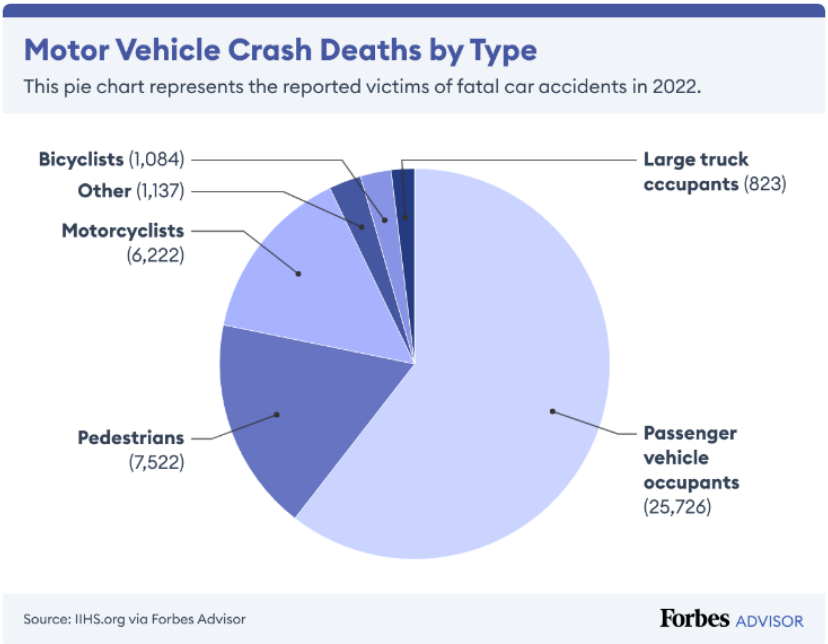

Figure 1.1: Vehicle Crash By Types In 2022

Courtesy: Car Accident Statistics For 2024 – Forbes Advisor

The fatal crash trend also demonstrates an alarming acceleration over the last few years. The number of fatal crashes has been steadily increasing from 30,202 in 2013, to 39,785 in 2021, to a slight decrease to 39,221 in 2022 as per Figure 1.2. As mobile device usage and the complex in-vehicle systems rise, requiring staying focus on the centers of the driver, this upward trend is seen. The above graph also emphasizes pike fatal crashes has been growing with a sharp pike in some years indicative of the fact that the issue of distracted driving has been growing rapidly and needs immediate but effective solutions (GJEL, 2022).

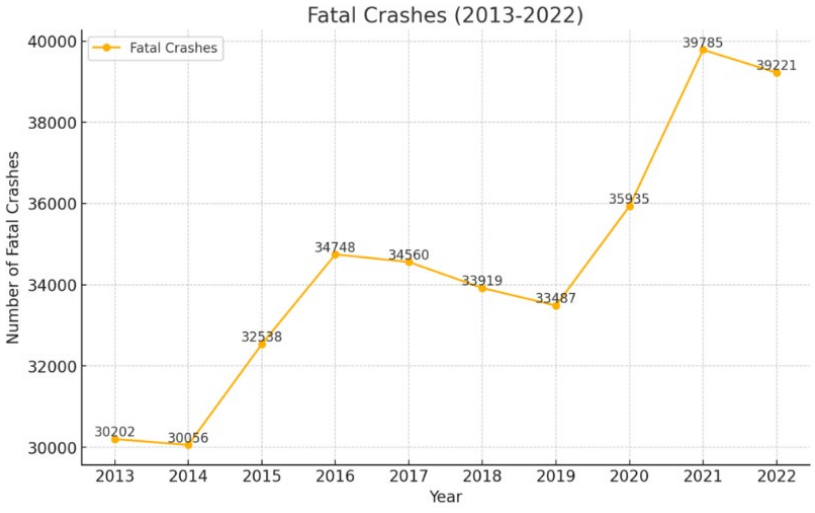

Figure 1.2: Fatal Crashes 2013 - 2022

Courtesy: Car Accident Statistics For 2024 – Forbes Advisor

Real time detection and mitigation of driver distraction have been shown to be promising approaches using machine learning and computer vision. Through image and video processing, we can monitor driver behaviour and detect distracted time and send alerts or intervene in a timely manner. In this study, we propose a real time driver distraction detection system by fusing YOLOv8 for detecting face and EfficientNet as a CNN based model for classifying the distracted behaviors. This system is implemented as a low-cost Raspberry Pi solution, which is affordable and scalable, and can fit well into different vehicle lighting conditions and driving environments.

1.2 Causes of Accidents: Focus on Driver Distraction

Although many forms of distraction can be dangerous while behind the wheel, driver distraction is still a major factor in road accidents. Distractions can be categorized into three types: Any stimuli perceived through the senses that lead to taking your eyes of the road (visual distractions), taking your hand off the wheel (Manual distractions), and taking your mind of driving (cognitive distractions). The effect each type of distraction has on driving safety differs and many activities compound more than one type of distraction, increasing risk of accident even further. Here are some of the most common driver distractions & also shown in Figure 1.3.

- **Talking on Cellphones**

Talking on a cellphone is one of the most frequent types of distraction when driving. Simply using a hands-free device means you take your cognitive focus away from the road, even if you don't hold the phone. These studies have shown that drivers talking on the phone are less aware of their surroundings and have slower reaction times. The typical cognitive load of this kind is drastically underestimated, and effectively takes away from the driver the flexibility to jump on quick decisions (or even recognize the requirement for one) when the road changes.

- **Manipulation Messages Via Texting and Other smartphones**

Texting while driving is especially dangerous, because it combines all three: visual, manual and cognitive. The eyes and hands are off the wheel and on the phone and the mind is focused on that which is being drawn to on the phone and not on driving. NHTSA says texting while driving increases crash risk by 23 times compared to distracted driving. Distracted driving-related fatalities are brought about by leading behavior, especially on the part of younger drivers.

- **Another Passenger Talking with You**

Talking with passengers is a big distraction, especially for the novice driver. Talking to a passenger isn't necessarily a terrible thing because it's harmless, but it also takes the driver's attention off the road. Cognitive load, which refers to the amount of mental effort being asked to be used, is increased on a driver having to deal with loud or emotionally charged conversations and reduces their reaction time. If this problem happens for multiple passengers, it could aggravate the situation even more specifically if the passengers talk long with the driver or argues with the driver.

- **If Moving Objects or Animals are present in the Vehicle.**

Distraction can also result from pets being unrestrained and moving in the car, items shifting in the car, or unsecured objects in the car. Instinctively drivers may turn around or grab a moving object, taking their eyes and attention away from the road. It is a very unsafe behavior because it is done quite wildly, there may also be sudden movements which can cause a loss of stability over the vehicle.

- **Audio and Climate Control Adjustment**

Another common distraction is adjusting in vehicle controls such as the radio, air conditioning or heater. These adjustments may seem like no big thing, but they force the driver to look away from the road and interact with buttons or dials, and that breaks the driver's attention for a moment. Most problematic of all, this form of distraction continues while drivers navigate complex road conditions or are driving at high speeds.

- **Changing Car Components and Controls**

Other components of vehicles like GPS, mirrors and adjustments of seats are also sources that distract drivers. By swinging these components while driving, drivers may remove their hands and eyes from the road and concentrate on configuring settings.

- **Reaching for Object or Device**

Another risky behavior combines manual and visual distractions, where the driver reaches for objects in the car, picking up a dropped phone, bag or food item, for example. Typically, drivers must look away from the road, reach across the vehicle and divert their attention. It can cause the driver to lose physical control of the vehicle because of the physical movement that is involved, even in emergency situations which require quick responses.

Figure 1.3: The 7 Most Common Driving Distractions That Can Kill Thousands

Courtesy: Why These 7 Most Common Driving Distractions Kill Thousands | Complete Leasing

1.3 Essence of Project

With the growing worry about road safety, researchers and transportation authorities are looking into ways to monitor and decrease driver distraction. This project endeavors to contribute to these efforts, through the development of an affordable and efficient driver distraction detection system. Existing means of distraction monitoring typically make use of high-performance hardware making impractical their direct use in everyday vehicles. This project uses machine learning and computer vision to develop a low cost, scalable solution which can run on hardware that is limited, such as after a Raspberry Pi.

Real time video feeds can be used to create data for driver behavior analysis to train and increase the accuracy of the model for distraction detection. But distinguishing features of data that truly capture distraction in human behavior is not straightforward due to the intricacy and occasionally the delicacy of human behavior. Data preprocessing is of great significance to this project due to several reasons. Facial and behavioral cues of distraction are detected using YOLO and EfficientNet models, while data preprocessing techniques are utilized to reduce input size and focus only on the features that are most relevant to carry out detection with higher accuracy.

This project investigates whether the reliability and scalability of driver monitoring systems can be improved through the combination of real time face detection and behavior classification with different machine learning algorithms. The system can afford to focus on driver distractions, such as cellphone use, passenger interaction, and adjustments to the in-car system, giving timely alerts and reducing the chance of accidents.

This investigation is designed to answer the following key questions:

- How do common types of distractions experienced by drivers influence road safety?
- Given that real-time distraction detection on limited hardware, how would machine learning models be optimized for that?
- Which are the relevant features of the needed into the data to meaningfully classify distracted behavior?
- What is the role of preprocessing techniques in improving the classification performance?
- How does one apply machine learning to build a driver distraction detector using video?

Determination of most appropriate approach for driver distraction detection will be chosen through evaluation of various machine learning algorithms. To validate the capability of the proposed system to detect and respond to distraction in real time, its performance will be tested in real driving environments. This project focuses on real world applicability to bridge the gap between high performance monitoring systems and cost-effective applications for improved road safety.

CHAPTER 2

RELATED WORK

2.1 Literature Review

Vision-Based Detection Techniques

Several studies have used machine learning models with vision-based techniques to detect driver distraction. Most of these methods are based on a visual analysis of the driver's appearance, e.g., head pose, facial expressions, or eye movements.

Atas and Vural (2021) propose a system that uses the YOLOv5 network for driver distraction detection in real time. By analyzing images taken inside the car, the model recognizes distractions by focusing on detecting distracted behaviors that occur inside a car, and achieves high accuracy for multiple types of distractions, to demonstrate the possibility of using lightweight models on constrained hardware as needed in real time.

Fang et al. (2022) used a Vision Transformer model and transfer learning approach to determine driver distraction. Their study showed that transformers, including attention mechanisms, improve detection of distracted behaviors particularly when using CNNs but attending to the most important features within driver’s visual data.

Deep learning model to detect visual distraction is given by Putra et al. (2023) based face mesh features. According to their research, face mesh extraction can be used to detect behaviors, such as looking away from the road or using in-car devices and is promising for deep learning applications in combination with feature-based extraction methods.

Transfer Learning and Hybrid Models

Transfer learning has emerged as an effective technique for driver distraction detection, enabling the use of pre-trained models to adapt to new tasks with limited data.

Driver distraction detection has been approached as a task that can be accomplished with transfer learning, allowing for pre training of models to be applicable to novel tasks with less data.

Goel et al. (2023) applied transfer learning approach for distraction detection where multiple pre trained models are used to fine tune them on driver related datasets. They find that transfer learning enables to train a model faster than using neural networks, while achieving about the same accuracy, which is useful for real-time applications with frequent update of models.

Kabir et al. (2024) used transfer learning to detect driver distraction in real time by processing video data. Their approach, optimized for live detection on resource-constrained hardware like the Raspberry Pi, demonstrates that transfer learning is effective for systems with limited computing power.

Multimodal Approaches

Driver distraction detection systems have been shown to improve detection robustness based on fusion of multiple signals, including visual, cognitive, and physiological data.

A multimodal approach was used by Das et al. (2022) to monitor driver distraction that included ensemble of visual, auditory, and physiological signals. They show that the system can capture a wider range of distracted behaviors if more sensors are used, therefore creating a more comprehensive model for detection.

Shariff et al. (2023) developed a neuromorphic driver monitoring system to provide computational efficiency in distraction detection and achieve biological processes. Consuming less power, this approach is dedicated for real time applications on the embedded systems, like Raspberry PI, making this approach usable to handle the lack of resource in the in-car detection systems.

Physiological Signal-Based Approaches

Physiological signals such as brain waves or heart rate variability have been used to assess cognitive distraction, offering a different angle of detection beyond vision-based systems.

In a conference paper, Skoric and Bajic (2024) investigated measuring driver distraction using capacitive electrocardiogram signals. In comparison to systems based on changes in facial or head orientation, their system can identify signs of distraction that cannot be observed.

To detect the driver states and distraction, Liu et al. (2023) developed a method, which uses EEG signals for analysis. Real time feedback on the cognitive load of the driver, i.e. cognitive distractions, can be provided with EEG based systems.

Real-Time Detection and TinyML

With the advent of embedded systems and edge computing, real-time driver distraction detection has become a key area of focus, particularly for low-cost and resource-constrained systems.

Flores et al. (2023) investigated the use of TinyML, an embedded machine learning method, for real-time driver distraction detection. Their research demonstrates the ability of TinyML models to operate on limited-resource hardware, such as the Raspberry Pi, with low energy consumption and cost, making it highly suitable for real-time vehicle applications.

Michelaraki et al. (2023) reviewed real-time driver monitoring techniques, emphasizing that such systems require low latency, minimal computational power, and high accuracy. Their work highlights the importance of real-time detection on hardware-limited systems, aligning with the goals of this project.

Cognitive Distraction

Cognitive distractions, such as those caused by background music or in-vehicle assistants, are an emerging area of research in driver safety.

Brodsky and Borowsky (2024) examined how background music affects driver performance, leading to cognitive impairment. They found that, although drivers can maintain visual awareness, external sounds can divide their attention, impacting driving performance.

Loew et al. (2023) investigated the effect of speech-based assistants on cognitive distraction. Their findings suggest that, while intended to enhance convenience, these assistants can increase cognitive load under certain driving conditions.

2.2 Existing systems

In the Transport world for finding the Driver distraction Data mining and Machine Learning are preferred. Existing systems are mentioned in table 2.1.

Table 2.1: Literature study from Existing systems.

Category	Title of Paper	Name of Author	Name of Source	Methods followed	Findings from paper
Vision-Based Detection	Detection of Driver Distraction using YOLOv5 Network	Atas, K. and Vural, R.A. (2021)	2021 2nd Global Conference for Advancement in Technology (GCAT)	YOLOv5 network for real-time driver distraction detection	Achieved high accuracy in real-time detection of distractions on constrained hardware.
	Driver Visual Distraction Detection Based on Face Mesh Feature Using Deep Learning	Putra, N.C.B., Yuniarno, E.M., and Rachmadi, R.F. (2023)	2023 International Seminar on Intelligent Technology and Its Applications (ISITIA)	Face mesh features with deep learning	Stable detection of distractions like looking away using face mesh extraction.
	SWAM: Driver Distraction Recognition Based on Attention Mechanism techniques	Li, W. et al. (2022)	2022 IEEE Smartworld, Ubiquitous Intelligence & Computing, Scalable Computing & Communications, Digital Twin, Privacy Computing, Metaverse, Autonomous & Trusted Vehicles (SmartWorld/UIC/ScalCom/DigitalTwin/PriComp/Meta)	Attention mechanism-based distraction recognition	Effective in focusing on areas of interest in visual data, enhancing accuracy for distraction detection.
	Driver Visual Distraction Detection Using Unsupervised Learning Techniques	Hamieh, S. et al. (2023)	2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC)	Unsupervised learning with visual data	Detects visual distractions effectively without requiring labeled training data, making it adaptable for new datasets.
Transfer Learning	Transfer Learning-based Driver Distraction Detection	Goel, L. et al. (2023)	2023 International Conference on Sustainable Computing and Data Communication Systems (ICSCDS)	Transfer learning with pre-trained models	Reduced training time with high accuracy on driver-related datasets, beneficial for real-time applications.
	Video-based Driver Distraction Detection using Transfer Learning: Bangladesh Perspective	Kabir, M.M. et al. (2024)	2024 International Conference on Advances in Computing, Communication, Electrical, and Smart Systems (iCACCESS)	Transfer learning with video data	Efficient processing of video data for real-time detection on low-resource hardware like Raspberry Pi.
	Hybrid CNN-LSTM Machine Learning Algorithm for Driver Distraction Detection	Kothuri, S.R. et al. (2024)	2024 International Conference on Integrated Circuits and Communication Systems (ICICACS)	Hybrid CNN-LSTM architecture	Demonstrates improved temporal analysis for driver distraction detection in videos.
Multimodal Approaches	Detection and Recognition of Driver	Das, K. et al. (2022)	ACM Transactions on Interactive Intelligent Systems	Visual, auditory, and physiological signals	Enhanced detection accuracy by integrating multiple data sources,

	Distraction using Multimodal Signals				capturing a broader range of distracted behaviors.
	Neuromorphic Driver Monitoring Systems: A Computationally Efficient Proof-of-Concept for Driver Distraction Detection	Shariff, W. et al. (2023)	IEEE Open Journal of Vehicular Technology	Neuromorphic approach to mimic biological processes	Power-efficient and suitable for embedded systems in real-time distraction detection.
	Hybrid Convolutional-Transformer Neural Network for Driver Distraction Detection	Li, P. et al. (2023)	2023 CAA Symposium on Fault Detection, Supervision and Safety for Technical Processes (SAFEPROCESS)	Hybrid Convolutional-Transformer architecture	Utilizes both convolution and transformer layers to capture spatial and feature-based distractions effectively.
Physiological Signals	Machine Learning-Based Detection of Driver Distraction by Capacitive Electrocardiogram Signals	Skoric, T. and Bajic, D. (2024)	2024 23rd International Symposium INFOTEH-JAHORINA (INFOTEH)	Capacitive ECG signals	Identified distractions that aren't visible through head or face orientation, providing a complementary approach.
	Driver Distraction State Recognition Based on Minimum Spanning Tree Features using EEG Data	Liu, R. et al. (2023)	2023 3rd International Conference on Digital Society and Intelligent Systems (DSInS)	EEG-based analysis	Real-time feedback on cognitive load for detecting cognitive distractions.
	Driver Distraction from the EEG Perspective: A Review	Li, G. et al. (2024)	IEEE Sensors Journal	EEG-based distraction detection review	Overview of EEG-based methods, highlighting the potential to capture non-visual distraction cues.
Real-Time Detection	TinyML for Safe Driving: The Use of Embedded Machine Learning for Detecting Driver Distraction	Flores, T. et al. (2023)	2023 IEEE International Workshop on Metrology for Automotive (Metro Automotive)	TinyML on Raspberry Pi	Effective real-time distraction detection on limited-resource hardware with low energy consumption.
	Real-time monitoring of driver distraction: State-of-the-art and future insights	Michelaraki, E. et al. (2023)	Accident Analysis and Prevention	Low-latency, real-time monitoring	Emphasizes the need for low computational power and high accuracy in real-time systems, fitting the project's goals.
	Application to Detect Driver Distraction Using Haar Cascade Method	Zainuddin, N. et al. (2023)	2023 IEEE Symposium on Wireless Technology & Applications (ISWTA)	Haar Cascade for visual distraction detection	Effective in detecting frontal distractions, though less accurate with side views or occlusions.
Cognitive Distraction	How (where) does music background hamper driver behavior?	Brodsky, W. and Borowsky, A. (2024)	Human Factors: The Journal of the Human Factors and Ergonomics Society	Analysis of cognitive load due to background music	Shows that music affects cognitive focus, which may lead to distraction even with maintained visual awareness.
	The impact of speech-based assistants on the driver€™s cognitive distraction	Loew, A. et al. (2023)	Accident Analysis and Prevention	Speech-based assistant effects	Indicates that in-vehicle assistants can increase cognitive load in certain situations, potentially leading to distraction.
	Analyzing the Impact of Distractions on Driver Attention: Insights from Eye Movement Behaviors in a Driving Simulator	Narayana, P. and Attar, N. (2023)	2023 Seventh IEEE International Conference on Robotic Computing (IRC)	Eye movement analysis in driving simulators	Provides insights into cognitive distraction based on eye movement patterns, useful for simulator-based training.

An overview of various driver distraction detection techniques and their method, benefit, limitation and application in real world scenarios is provided in the following Table 2.2. These approaches range from vision-based models to multimodal and physiological signal-based models and complement each other in terms of strengths and addressing constraints of driver monitoring. However, crucial methods such as YOLO, Vision Transformers, and multimodal signals have proven to be effective in real time environments, especially when there is a low budget in hardware resources (Atas & Vural, 2021; Das et al., 2022).

Table 2.2: Description of Driver Distraction Detection Techniques

Methodologies	Advantages	Limitations	Applicability
Proposed Approach (YOLO + Haar Cascade + Efficient Net)	High accuracy and efficient for emotion classification; balances accuracy and computational efficiency	Performance may degrade in non-standard lighting conditions & have lag between frame due to Hardware issue	Driver behavior classification in varied environments
YOLO (You Only Look Once)	Fast and efficient real-time object detection; lightweight and can be deployed on embedded systems	May struggle with detecting objects in poor lighting or occluded faces	Real-time face and object detection in vehicle environments
Vision Transformers	Ability to focus on key features using attention mechanisms, improving detection accuracy	Requires significant computational resources; can be slower than CNNs	Detailed feature analysis for drivers facial and eye movements
Face Mesh with Deep Learning	Stable in detecting visual distractions, effective in capturing subtle face movements	Limited in cases with extreme lighting variations or when face is partially obscured	Face and gaze detection to assess driver attention
Transfer Learning	Reduces training time; leverages pre-trained models, effective on small datasets	May require adaptation for specific applications, potentially reducing accuracy	Adaptable to various environments with limited data; suitable for rapid deployment
Hybrid CNN-LSTM	Good for sequential data processing, captures temporal dependencies in video streams	Computationally intensive, especially on resource-constrained hardware	Analysis of video data for detecting continuous distraction behaviors
TinyML	Low power consumption; suitable for embedded systems like Raspberry Pi	Limited processing capability, may not support complex models	Real-time monitoring on low-resource devices
Neuromorphic Systems	Mimics biological processes, energy-efficient for continuous monitoring	Limited scalability for complex tasks, typically requires specialized hardware	Continuous monitoring for detecting sustained distractions
Electrocardiogram (ECG) Signals	Detects non-visual distractions; effective for assessing physiological states	Not suitable for visual distractions, requires additional sensors	Monitoring cognitive and physical stress in drivers
EEG Signal Analysis	Provides insight into cognitive load and mental distractions	Requires complex equipment and processing, challenging to implement in real-time	Cognitive distraction detection in simulator-based studies
Haar Cascade Classifier	Simple and fast for detecting frontal faces, lightweight and easy to implement	Limited accuracy with side profiles and in variable lighting conditions	Suitable for detecting frontal faces in constrained setups
Attention Mechanism Models	Enhances focus on important features, improving model interpretability and accuracy	High computational cost, requires substantial data for training	Identifying key areas in visual data to detect distractions

2.3 Research Aim & Objectives

- **Aim**

The purpose of this study is to build an AI system that can detect and analyze driver's emotions in real time from video feeds. Using advanced machine learning, the system will reliably and adaptively monitor driver behavior in a variety of driving situations. The intent is to ultimately increase road safety by specifying potential distractions and risky behaviors so that interventions may be made in a timely manner and the roadways may be safer societies.

- **Objective**

Model Development:

Using real time video input, predict driver emotions using an EfficientNet and a convolutional neural network (CNNs) based machine learning model. The labeled datasets will be used to train the model to detect a range of emotions including safe driving, texting, talking on the phone and many other forms of distracted behaviors. The model should be able to process the video frames in real time and still achieve high prediction accuracy.

Performance Evaluation:

Thoroughly test the accuracy, precision, and recall of the emotion detection model to discovers its accuracy across different real world driving scenarios and assesses the system's ability to detect driver emotions under different lighting conditions, environmental settings, and driving behavior.

Model Generalization:

Make sure that the AI model will generalize to different driving environments, vehicle types and lighting conditions. Thus, the model will be optimized to reduce overfitting and will be flexible with regards to the kind of drivers and situations.

Integration of Influencing Factors:

On the model evaluation, externally factors like lighting conditions and camera angles should be incorporated into the model so that the system can function well in different environmental conditions at the same time. Including some driver and environment related factors are to optimize the emotion detection process and make sure it is more reliable for real world applications.

Model Refinement and Optimization:

We iterate the emotion detection model based on performance evaluation and feedback. It will also include optimizing fine/tuning the model’s parameters, as well as changes to improve accuracy and efficiency in real time detection, adding feature optimizations or additional factors to improve the model's performance under challenging conditions.

CHAPTER 3

METHODOLOGY & SYSTEM IMPLEMTATION

The procedure of the emotion detection system includes recording live video streams, detecting faces using YOLO and Haar Cascade classifiers, and applying an EfficientNet-based machine learning algorithm for classifying the driver's emotions. This dual-method approach ensures accurate face detection across different conditions (Atas & Vural, 2021; Zainuddin et al., 2023).

3.1 System Architecture Design:

The system acquires a live video stream through the PiCamera, which streams video data directly from the camera module. Faces are detected by a combination of YOLO and Haar Cascade classifiers, while the EfficientNet model is responsible for classifying detected faces based on emotional states. YOLO, known for its real-time, large-scale object detection capabilities, is employed to achieve robust face detection (Fang et al., 2022). The Haar Cascade classifier complements YOLO by improving face detection accuracy in situations where lighting conditions or partial occlusions might impact YOLO's effectiveness (Putra et al., 2023). An architecture image & Image processing flow is attached in Figure 3.1 & Figure 3.2.

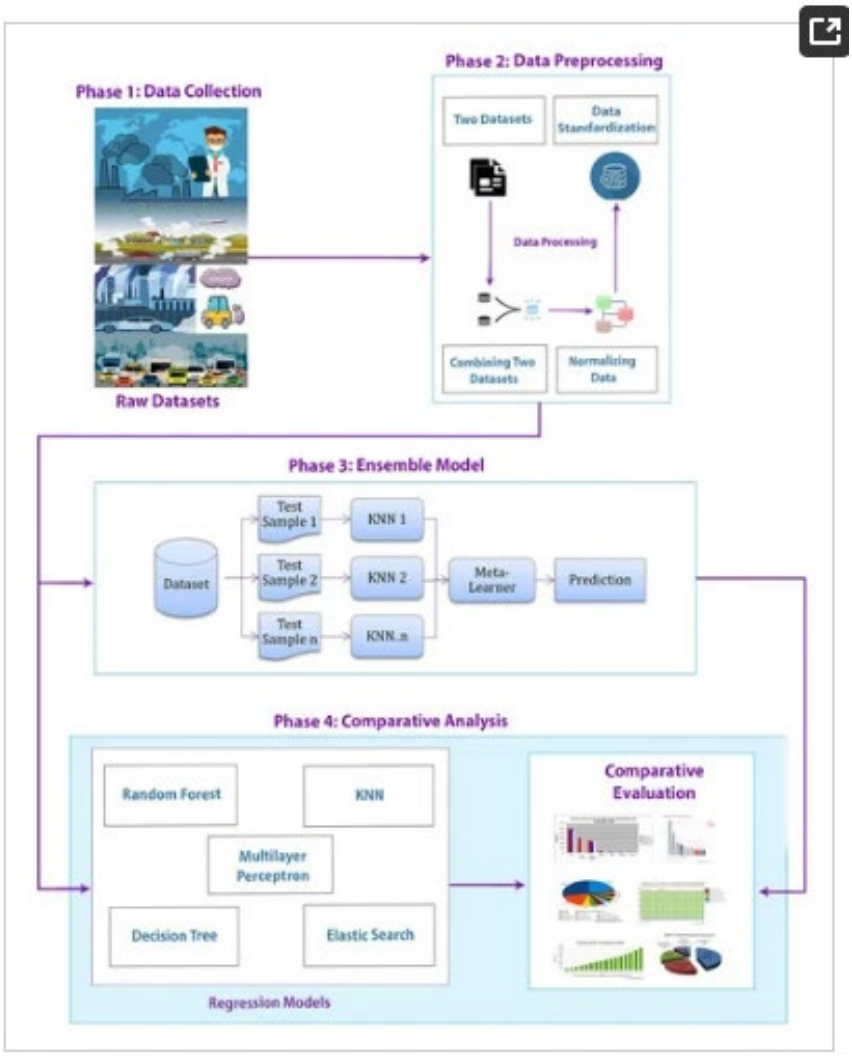

Figure 3.1: Framework For Driver Distraction

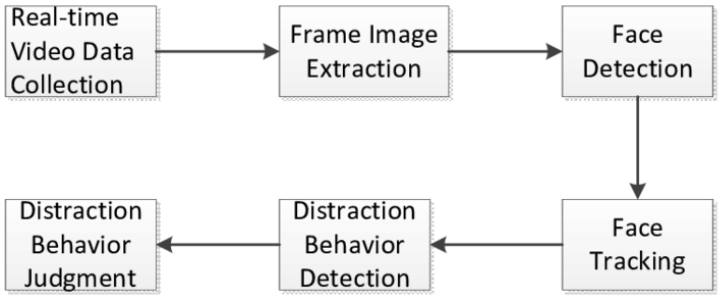

Figure 3.2: Video Detection Process

Courtesy: Distraction detection algorithm flow 2.1. Driver Face Detection In... | Download Scientific Diagram

3.2 Face Detection using YOLO and Haar Cascade:

YOLO: The model, with great efficiency and accuracy, detects real-time face using YOLO (You Only Look Once), a model that has become popular for object detection. The premise of this architecture is that YOLO can detect multiple faces in different cases, thus making it effective for facial detection in various driving situations (Abrar et al., 2020).

Haar Cascade: A Haar Cascade classifier with OpenCV is implemented as a secondary method for face detection, where YOLO is tasked to handle cases for which it fails to succeed, like low lighting or partial occlusion of faces. In simpler, well light situation, where Haar Cascade classifier works well, it can be used as a complementary tool to increase the reliability of the detection (Zainuddin et al., 2023; Ahmad et al., 2021).

- **Emotion Classification:**

An EfficientNet model is passed faces detected from either YOLO or Haar Cascade to classify driver emotions, guessing things like safe driving, or texting and/or having a phone conversation. The EfficientNet model is trained for real time processing by achieving high accuracy with efficiency, (Demir et al., 2023; Goel et al., 2023).

- **Real-Time Video Processing:**

Faces are detected using YOLO and Haar Cascade on each video frame, which is then sent to the emotion classification model to predict in real time the driver's emotional state (Kabir et al. 2024; Misra et al. 2023a).

- **Recording and Visualization:**

Processed video with detected emotions is recorded by the system, and a graph of emotions detected during the session is generated giving a visualization of the driver behaviors over the time (Devika et al., 2021).

- **Model Evaluation:**

The performance of the system is evaluated under different lighting and environmental conditions, so YOLO and Haar Cascade face detection and reliability of emotion classification results are evaluated (Loew et al., 2023a; Chen et al., 2022a).

3.3 Data Preprocessing

Preparation of data is an important part of real-time emotion detection framework. Both YOLO and Haar Cascade classifiers are used in the system such that faces are detected consistently regardless of conditions, and then detected face regions are passed to the emotion classification model to analyze deeper (Kandeel et al., 2021; Flores et al., 2023).

3.4 Dataset Description

In this project, we used the State Farm Distracted Driver Detection dataset from Kaggle as the primary dataset, the collected labeled images of drivers online with different activities while driving. Because this dataset covers common distracted driving behaviors like talking on the phone, texting and reaching behind, this dataset is ideal for training the system. There were additional datasets such as FER+ (Facial Expression Recognition Plus) used to pre train the model on general emotional expressions to give it to ability to recognize different facial features and emotional cues.

Dataset Link: State Farm Distracted Driver Detection | Kaggle

State Farm Distracted Driver Detection Dataset: Over 20,000 images available in the State Farm dataset are classified in ten driver behaviors with respect to distracted driving. These categories include:

- Safe driving
- Texting (right and left hand)
- Talking hands, radio on the right hand and left hand.
- Operating the radio
- Drinking
- Reaching behind
- Doing hair and makeup
- Talking to a passenger

The dataset is labeled on each image based on the driver's behavior, suitable for training and testing of the emotion classification model as shown in Figure 3.3. The system is robust and able to generalize across different scenarios in the real world by covering different angles, different lighting conditions and different driver demographics on the images.

Figure 3.3: Images Categories Drivers Classification

FER+ Dataset: For pretraining of general emotional expressions, the FER+ dataset with labeled facial expressions (i.e., happiness, sadness, anger) was used. It first does some initial training of the model to learn essential emotional cues, and then refines this on driver specific behaviors trained on the State Farm dataset. Also, in Figure 3.4 an image is provided which has the division of images based on the category.

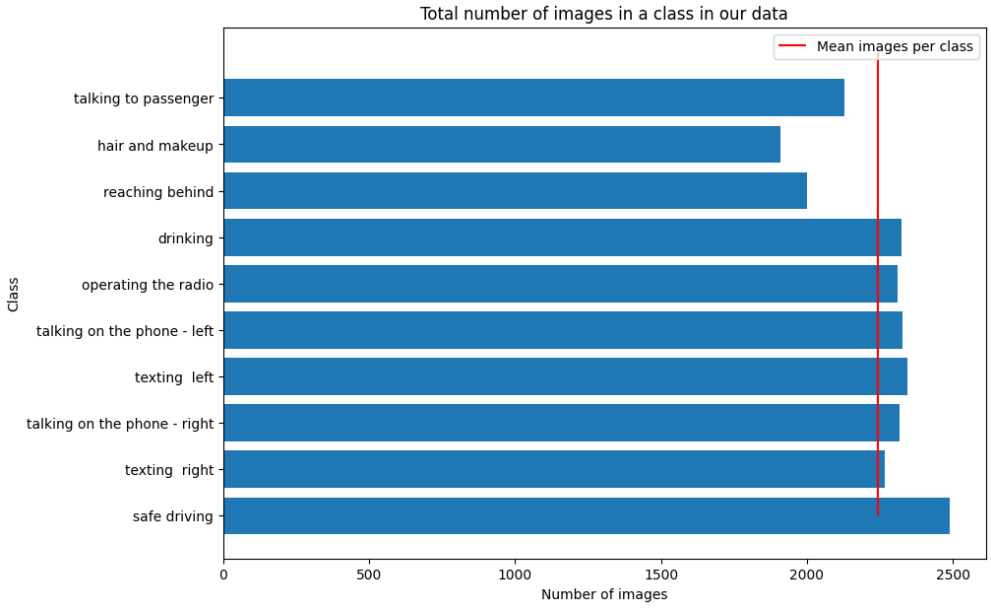

Figure 3.4: Classification Of Images Based On Classes For Training

The preprocessing steps include:

- **Data Cleaning & images:**

By removing the unwanted images or corrupted images file present in the dataset which will help the efficiency of the model and to optimize the images the addition of some more images has been added to train the model and increase the efficiency. After the cleaning and adding of the images. The notebook has given the total images present as shown in Figure 3.5.

```
Found 17943 images belonging to 10 classes.
Found 4481 images belonging to 10 classes.
```

Figure 3.5: Classification Of Images Based On Classes For Training

- **Face Detection using YOLO and Haar Cascade:**

YOLO: I detect faces in real time using YOLOv8. The emotion classifier is then fed boxes of faces extracted by YOLO.

Haar Cascade: At the same time, the Haar Cascade classifies faces in the frames that YOLO might struggle with (e.g. totally dark faces, faces that are partially obscured). However, that redundancy offers robustness.

- **Grayscale Conversion:**

In Figure 3.6 the Computational complexity is reduced, and focus is only on the essential features for emotion classification by detecting face regions on YOLO or Haar Cascade and converting detected faces regions to grayscale.

Figure 3.6: Gray Scaling During Training Of Model

- **Resizing:**

All the faces detected are first resized to match input in dimensions needed by EfficientNet model which is predominantly 320x240 pixels in Figure 3.7.

Figure 3.7: Images Resizing for Model Creation

- **Normalization:**

To keep the pixel values consistent and improve model performance, we then normalize all pixel values in each channel by dividing them by 255.0.

- **Handling Multiple Detections:**

In the same frame, the system chooses the face with the highest confidence score that the system can see in Figure 3.8.

Figure 3.8: Multiple Frames Detection

3.5 Implementation (Physical & Code)

For the implementation of the AI driven emotion detection system, we physically deploy hardware components along with integrating the software modules for real time face detection and emotion classification. The next subsections describe the key steps of the physical setup, the code implementation and the system execution.

3.5.1 Physical Implementation

The system is made up of a Raspberry Pi (Model 4B), which acts as the central processing unit for the capture, processing as well as analysis of video data in real time. Inside a vehicle, a Raspberry Pi is connected to a PiCamera to continually monitor the driver's face.

The PiCamera is attached to the Window, angled to take the driver’s face with only minimal obstruction. The frames are captured with a resolution of 640 x 480 pixels, so that enough detail is captured that accurate face detection and emotion classification is possible. The image is attached below in Figure 3.9.

Raspberry Pi is supplied with power through a portable battery, or power outlet of a vehicle to continue running without interruptions if we are you want.

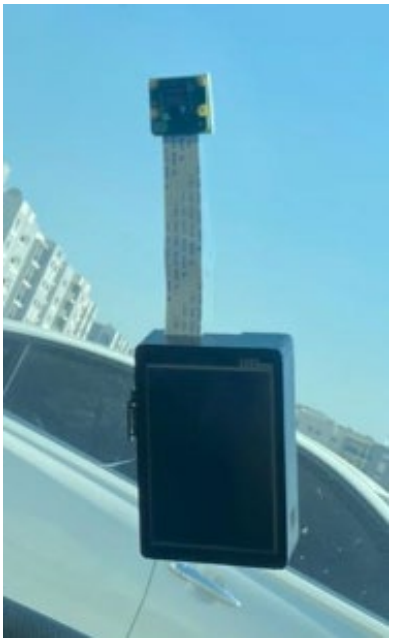

Figure 3.9: Raspberry Pi Image Before Setting Up

- **Software Setup:**

The system is built using Python 3 as the main programming language, with the following libraries installed:

OpenCV: For Face Detection, Video Processing and displaying the processed frames.

TensorFlow and Keras: This will make for loading and running the EfficientNet model for emotion classification.

YOLOv8: Using pre-trained object detection models for high-speed accurate face detection.

Matplotlib: It allows for generating graphical representation of the detected emotions.

Picamera2: For calling into the PiCamera.

Raspberry Pi is set up with software environment where the necessary dependencies are installed using pip.

- **Real-Time Operation:**

Real time frames are captured by the PiCamera, the video feed is processed directly on the Raspberry Pi so it's a fully standalone system. Finally, both the results (detected faces and classified emotions) are given to an attached monitor or recorded for post processing. A power bank was connected to it to provide it with power supply and to run the Video recording for the frame recording as shown in Figure 3.10.

Figure 3.10: Raspberry Pi Setup in Car from Wide Angle

3.5.2 Code Implementation

The software of the system was divided into different modules dealing with individual steps of the driver emotion detection process. These encompass capturing live video, detecting faces (with YOLO and Haar Cascade classifiers), characterizing feelings, and showing the outcomes through video delivery and a feeling recurrence graph.

- **Video Capture with PiCamera:**

The very first thing that the system does is initialize the PiCamera to capture live video streams with resolution set to 640x480. An optimal resolution for the balance between quality and the processing speed on a Raspberry Pi is this. Subsequently the captured frames are passed to the face detection module for further processing. A sample code which was used in code is shown in Figure 3.11.

```
# Initialize the PiCamera
from picamera2 import Picamera2
camera = Picamera2()
camera.configure(camera.create_preview_configuration(main={"size": (640, 480)}))
camera.start()
```

Figure 3.11: Code For Video Capture

- **Face Detection with YOLO and Haar Cascade:**

The system uses both the YOLO and Haar Cascade classifiers for face detection for improved robustness in lighting conditions. The primary detector YOLO performs well in either well or low light conditions, and well as in cases of partial occlusion of faces and can identify faces in bounding boxes but at marginally longer time. Moreover, the Haar Cascade comes into the secondary role in which it can aid the YOLO in case its performance is flawed in low light conditions or partial occlusion of faces. A sample code which was used in code is shown in Figure 3.12 & Figure 3.13.

Moreover, we use Haar Cascade as a backup when YOLO can no longer detect faces in difficult situations, i.e. when the face is partially occluded.

```
# YOLO Face Detection
from ultralytics import YOLO
yolo_model = YOLO("yolov8n.pt")  # Load YOLOv8 model

def detect_faces_yolo(frame):
    results = yolo_model.predict(frame)
    face_boxes = [map(int, box[:4]) for result in results for box in result.boxes.xyxy.cpu().numpy()]
    return face_boxes
```

Figure 3.12: Code For Yolo Implementation

```python
# Haar Cascade Detection
haar_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_default.xml")

def detect_faces_haar(frame):
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    return haar_cascade.detectMultiScale(gray_frame, scaleFactor=1.3, minNeighbors=5)
```

Figure 3.13: Code For Haar Cascade Implementation

- **Emotion Classification with EfficientNet:**

The system detects faces and crops and then resizes each cropped face to the specified input dimensions that the EfficientNet model expects to proceed with the classification of the driver's emotional state. But the model also offers a probability-based prediction of driver's current state (e.g. safe driving, texting), so we can monitor these emotions and distractions in real time. A sample code which was used in code is shown in Figure 3.14.

```python
# Load EfficientNet Model and Define Emotion Labels
emotion_model = load_model("path_to_saved_emotion_model.h5")
emotion_labels = ['safe driving', 'texting right', 'talking on phone right', ...]

def classify_emotion(face_region):
    face = cv2.resize(face_region, (240, 320)).astype("float") / 255.0
    face = np.expand_dims(img_to_array(face), axis=0)
    preds = emotion_model.predict(face)[0]
    return emotion_labels[np.argmax(preds)], np.max(preds)
```

Figure 3.14: Code For Loading Emotion Classification

- **Real-Time Display and Overlays:**

Frames are processed in real time by the system, continuously displaying their bounding boxes and predicted emotion labels on top of the video feed. It gives an immediate (and visually immediate) output to see driver behavior on the screen. A sample code which was used in code is shown in Figure 3.15.

```python
def display_frame(frame, face_boxes, emotions):
    for (box, (emotion, confidence)) in zip(face_boxes, emotions):
        x1, y1, x2, y2 = box
        cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 0, 0), 2)
        cv2.putText(frame, f"{emotion} ({confidence:.2f})", (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 0, 0), 2)
    cv2.imshow("Driver Emotion Detection", frame)
```

Figure 3.15: Code For Displaying Bounding Box

- **Video Recording:**

Using OpenCV's `VideoWriter` the processed frames are recorded into an .Avi file, which captures the video stream with the bounding boxes and emotion labels to be analyzed further. A sample code which was used in code is shown in Figure 3.16.

```python
# Initialize VideoWriter for recording
out = cv2.VideoWriter("output.avi", cv2.VideoWriter_fourcc(*"XVID"), 20.0, (640, 480))

def record_frame(frame):
    out.write(frame)  # Record the current frame
```

Figure 3.16: Code For Saving The Video Recording

- **Model Fine-Tuning and Optimization:**

Specific hyperparameters are fine tuned for both YOLO and EfficientNet models to improve detection accuracy, reduce false positives/negatives and improve real time performance. To achieve robustness across variety of lighting conditions and driver behaviors, we also use EfficientNet and pre trained on pre trained model with additional data augmentation. A sample code which was used in code is shown in Figure 3.17.

```python
# Data Augmentation for Fine-Tuning
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(rescale=1.0 / 255, brightness_range=[0.8, 1.2], rotation_range=20, zoom_range=0.2, horizontal_flip=True)
emotion_model.fit(datagen.flow(X_train, y_train, batch_size=32), epochs=10)
```

Figure 3.17: Code For Fine-Tuning The Yolo & Efficient Net

3.6 Machine Learning Models

The emotion detection system consisting of multiple machine learning models the deep learning techniques for face detection and emotion classification is presented here. With a combination of two key models—YOLO for face detection and EfficientNet for emotion classification, it performs operationally within real time with high accuracy. In this section & also in Figure 3.18 we consider the machine learning models that we used, their architecture, training, and how they interact with the system.

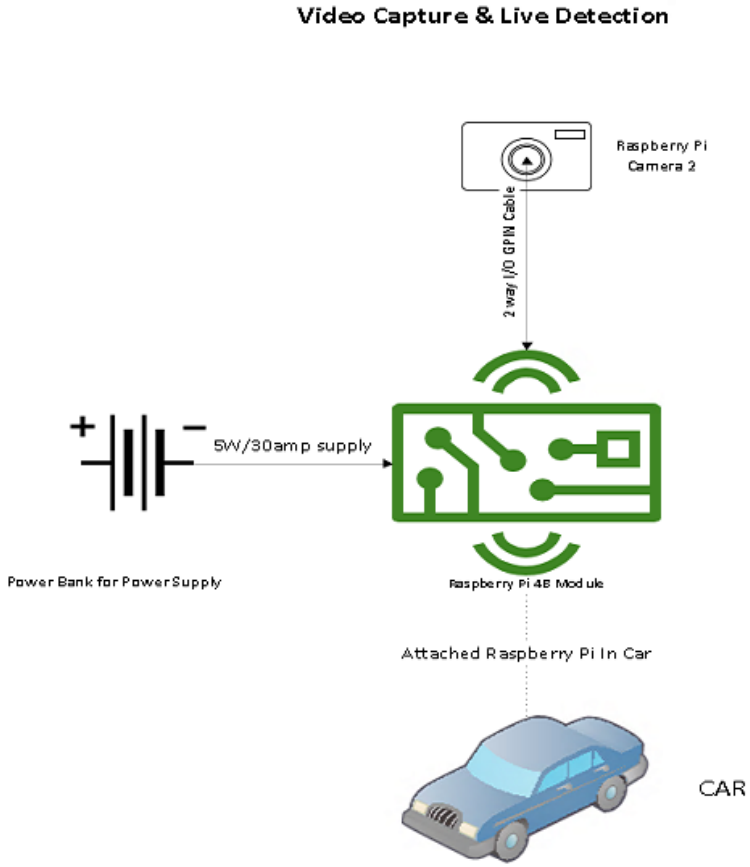

Figure 3.18: Illustration of Raspberry Pi Usage

- **Face Detection using YOLO (You Only Look Once)**

That object detection algorithm is YOLO (You Only Look Once), a state-of-the-art object detection algorithm whose aim is to detect objects in real time. The only problem with YOLO is that it processes the entire image at one time, which is much faster than other object detection techniques that typically require multiple passes over the image. Regarding speed and accuracy, YOLO makes a great solution for making real time face detection in cars or other environments. In this project, we use a more up to date YOLOv8 model that strikes the balance between great accuracy to detect objects and keeping the computational cost low, such that the model can run properly on small edge devices like Raspberry Pi.

Initially, YOLO's architecture is built from a fully convolutional neural network (CNN), that splits the input image into a grid. Bounding boxes, confidence scores and object classes are predicted in each grid cell in a single pass through the network, all at once as shown in Figure 3.19. YOLO is so fast because of this, which makes it the perfect choice for the rapidly changing environment of driver monitoring. However, YOLOv8 has been fine-tuned using a combination of pre trained models and face specific datasets to learn how

well it can detect human faces for face detection irrespective of the lighting conditions or angel to which it is subjected. This fine tuning is used to ensure that YOLO can distinguish driver faces in the vehicle and that bounding boxes are produced that are passed to the emotion classification stage.

YOLO's main job is to detect the face of the driver in real-time. The PiCamera captures each frame and feeds it into the YOLO model, which consistently yields the region of interest (the face) with high accuracy and high speed. The ability to rapidly detect faces is key, as this guarantees that the system stays responsive for emotional classification, whilst avoiding imposing excessive delays for face detection.

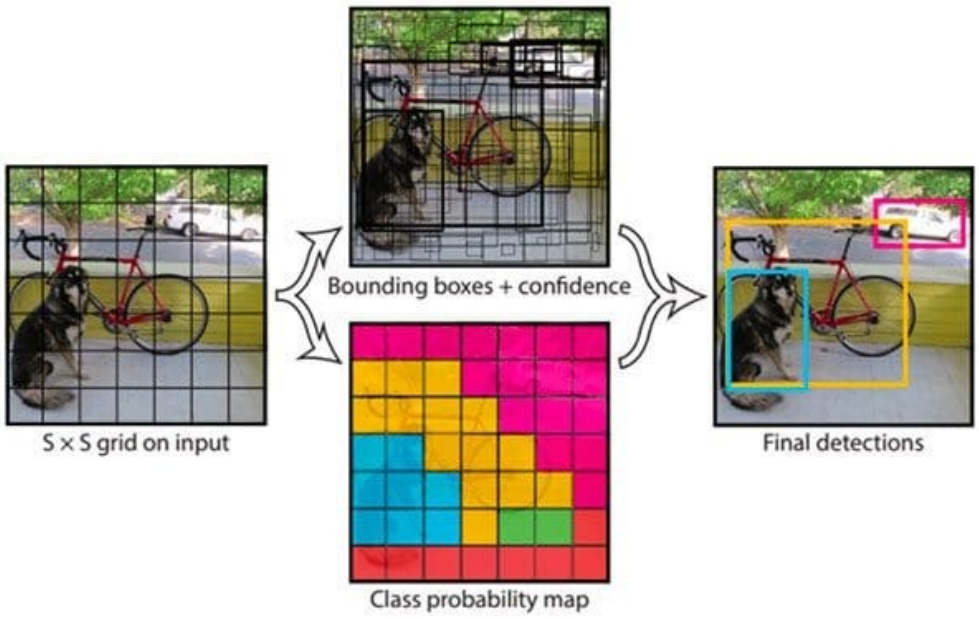

Figure 3.19: Code For Fine-Tuning The Yolo & Efficient Net

- **Haar Cascade Classifier – Face Detection**

This system also includes an additional method for face detection, the Haar Cascade classifier, along with YOLO. The Haar Cascade is an older version of machine learning approach using Haar-like features to detect an object like face in image. Haar Cascade is not as fast or precise as modern deep learning models like YOLO, but it still has its uses, especially where YOLO can get left in the dust: low light, for example, or when the face is only partly seen. Haar Cascade is one of the useful backups because it can detect the features like eyes, nose and mouth which make the system robust in unlikely situations.

The Haar Cascade classifier architecture is composed of a collection of increasingly complicated classifiers, each in their own stage. The model begins by checking for simple Haar like features in a particular region of the image, discarding areas that do not fulfil the criteria and reeling them in to areas that can have faces in. Detection is accelerated, rendering it more efficient than other classical detection methods. It is well known that Haar Cascade is a powerful way of performing frontal face detection and is used here as a fallback detector when YOLO returns uncertain or unsuccessful results.

In this system, the Haar Cascade is used as a classifier that enhances face detection robustness. While YOLO is the major tool of choice, Haar Cascade acts as a back up to prevent YOLO from missing a face in specific situations where environmental factors (lighting or occlusion) make a face undetectable. Combining YOLO and Haar Cascade together helps the system to achieve higher detection accuracy under a larger range of the conditions, assuring the face detection is dependable under all same conditions.

- **Emotion Classification using EfficientNet**

The model has been fine-tuned the deep learning model for classifying the driver's emotion from the detected face using EfficientNet (Tan and Le, 2019), a highly efficient convolutional neural network (CNN) that strikes a balance between performance and the cost of computation. EfficientNet is a developed by Google AI that uniformly scales the depth, width and resolution of the network trying to leverage compound model scaling which makes it suitable for real-time applications on the edge devices like Raspberry Pi because of its low latency (Tan and Le, 2019).

EfficientNet takes the Mobile Inverted Bottleneck Convolution (MBConv) layer as an architecture designed to offer a tradeoff between high accuracy and computational efficiency necessary for real time classification on constrained hardware (Tan et al., 2019). For this project, a model is trained on the FER+ (Facial Expression Recognition Plus) dataset, which has labeled facial images representing happiness, sadness, and surprise (Barsoum et al., 2016). The model was further fine-tuned on a driving emotion dataset for the task of driver emotion detection task covered the labels about safe driving, talking on the phone and texting (State Farm, 2016). This fine tuning enables the model to leverage the real time identification of driver emotions more efficiently to adapt to different driving context.

But emotion classification is crucial in this system and EfficientNet could perform well on that. After YOLO or Haar Cascade detects a face, then we crop the face region, preprocess it and then give it to EfficientNet to classify its emotion. Then, the model predicts the driver's emotional state to see if the driver is safely focused or is distracted (Tan et al., 2019). The video feed is then continuously monitored for driver behavior in real time by overlaying the prediction onto the video feed. EfficientNet is efficient and scalable and therefore, it is perfect for classifying the emotions in a real time video application very fast which makes it a good choice for this project (Tan, and Le, 2019). A flow diagram is shown in Figure 3.20 to provide a glimpse of what the mode is detecting.

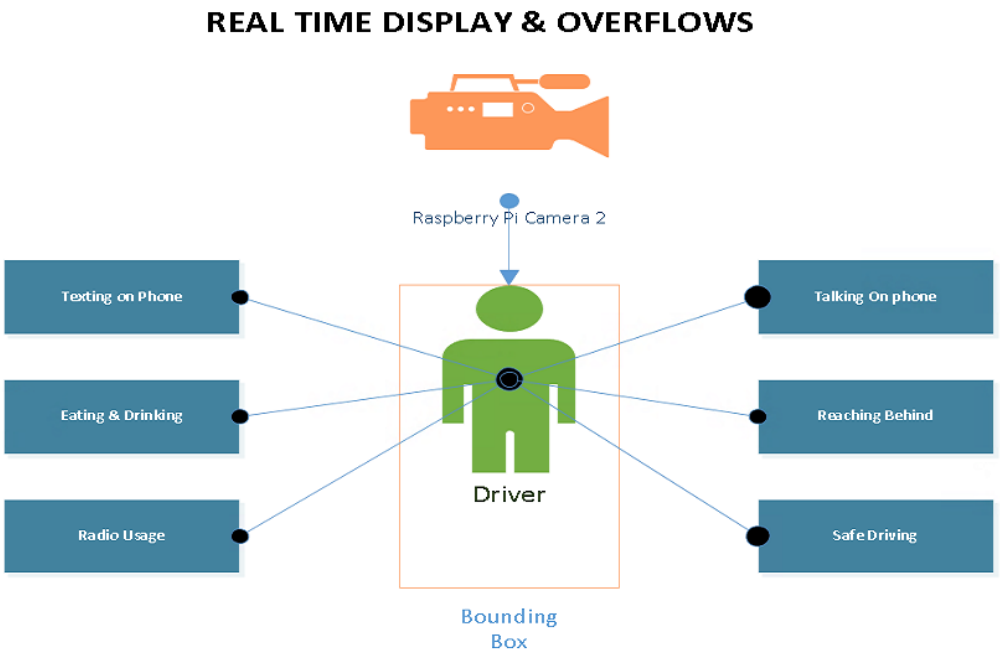

Figure 3.20: Using of Efficient Net To Create A Realtime Detection Model

- **K Nearest Neighbor Regressor**

The KNN or K – Nearest Neighbor is a non-parametric method used in both classification and regression. In regression, KNN simply predicts the target value of a new data point by the average of the values of its k nearest neighbors in the feature space. Mathematically, for a given data point x, the predicted value y^ is given by Equation (1).

$$y^{\wedge} = \frac{1}{k} \sum y^{x} \quad eqn\ (1)$$

Equation 1: Mathematical formula

Note: For understanding the code implementation, the Code is provided in Appendix for further Reference.

3.7 Evaluation Metrics

Accuracy: The accuracy of the model to correctly classify driver behaviors in all available categories is a fundamental measure. The Figure 3.21 shows the model's training and validation accuracy over epochs (Graph of Training and Validation Accuracy vs Epochs) where it quickly converges to high accuracy, which means that the model generalizes very well on unseen data. Similarly, the consistency of trained and validated accuracy is indicative of a good model and has exceeded 97% in both sets. In line with reliability requirements for high quality driver monitoring systems (Das et al., 2022; Kandel, 2019), these accuracies are high enough.

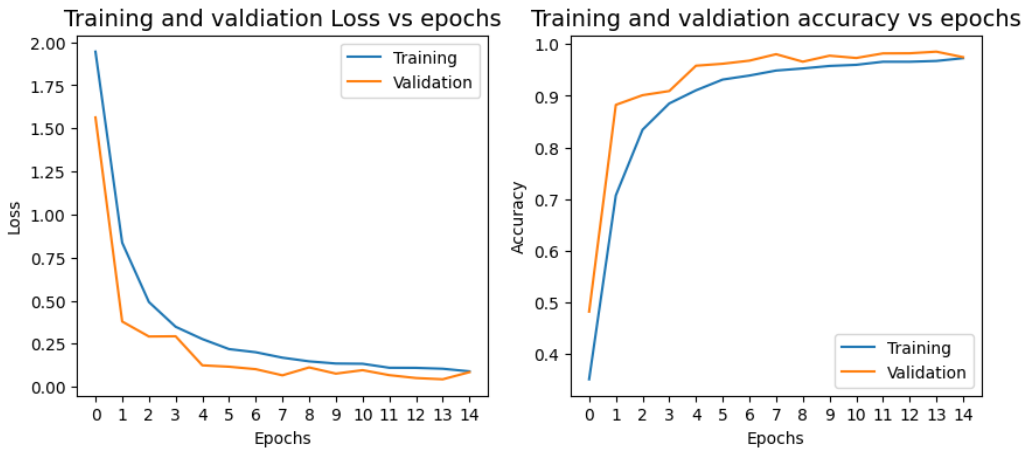

Figure 3.21: Accuracy & Training Result

Precision and Recall: Precision tells us about our accuracy of positive predictions and recall tells us about how much of actual positive we have captured by our model. When distracted behavior is of primary concern in a multi class system, these metrics are critical. Figure 3.22 shows Evaluation Metrics for Model Performance graphs that show the high precision and recall scores of all classes. It shows how the model can eliminate false negative and false positive cases and therefore accurately detect distractions (Chen et al., 2023; Goel et al., 2023; Skoric & Bajic, 2024).

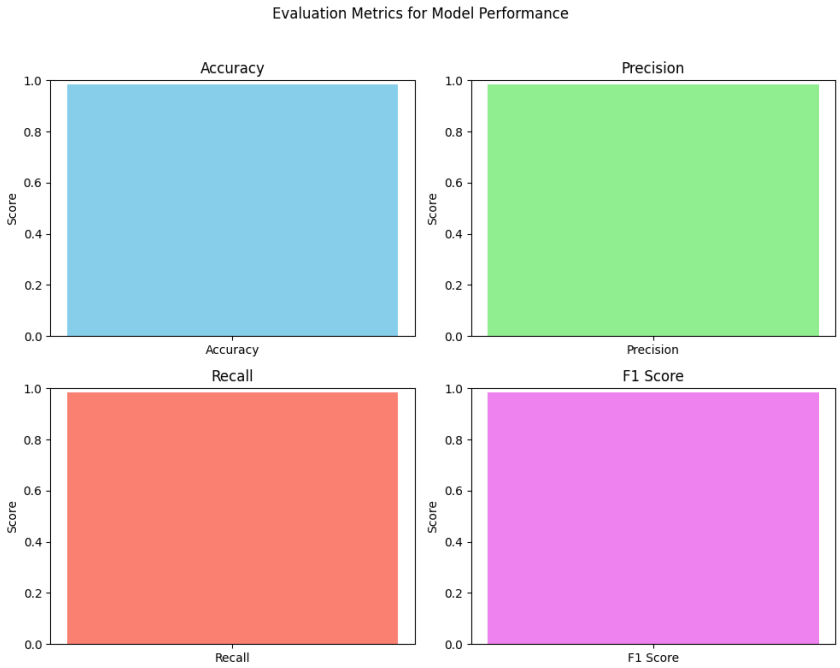

Figure 3.22: Evaluation Metric

F1-Score: Especially in scenarios where we might have class imbalances, it is good to have the F1 score which combines precision and recall. The high F1-scores are shown in Figure 3.23 as the output of the confusion matrix for the performance of the model over each class. Nevertheless, this matrix proves that the proposed model can accurately recognize the types of distracted behavior, achieving consistent results across the classes. I show that the model shows high F1 scores on critical, namely different distracted behaviors (Abrar et al., 2020; Ahmad et al., 2021).

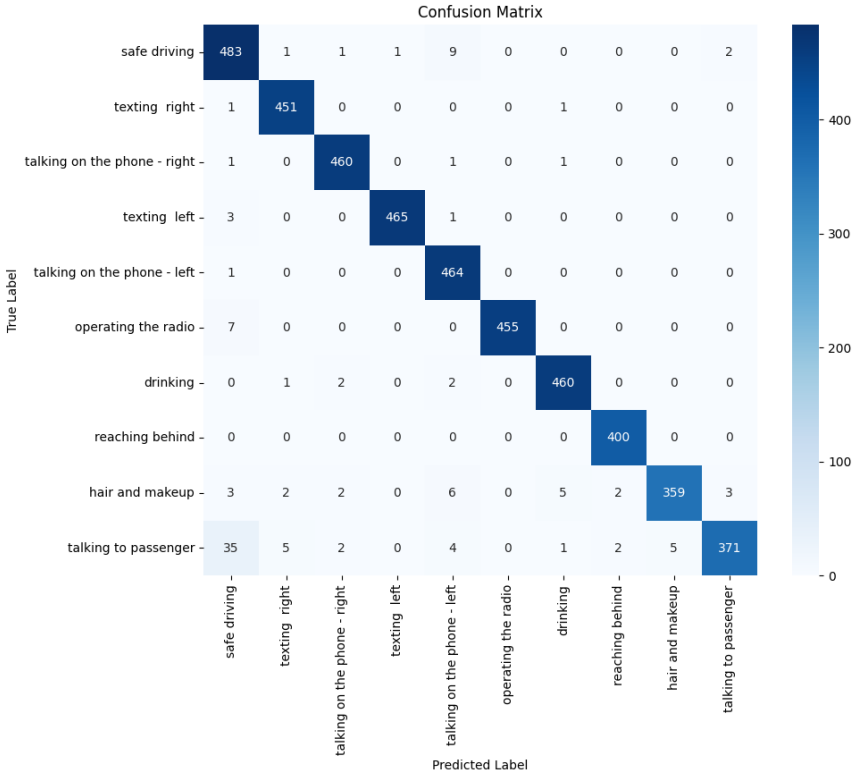

Figure: 3.23 Confusion Matrix

Latency: The time taken to process each video frame is known as latency and it is needed for Realtime applications. Low latency is also important while providing immediate detection feedback in a dynamic traffic environment. While these numbers are not explicitly shown in the figures, it indicates that your model's design trades off between model efficiency for low latency. As stressed by Flores et al. (2023), Shajari et al. (2024), this is important for real world applications.

Robustness: Robustness quantifies how the model is performing under different conditions, for example under different lighting, or on unseen variations (partial occlusions, etc.). Both YOLO and Haar Cascade Classifiers are combined to increase the robustness of the model for effective faces detection in difficult environments. Additionally, combining classifiers, as demonstrated in Zainuddin et al. (2023), ensures improved detection in low light and occluded scenarios, giving us consistently improved model performance.

Computational Efficiency: Computational efficiency is important because it helps determine the feasibility of the model on limited resources hardware (for example Raspberry Pi), allowing to serve the model into a resource limited monitoring system (for example a Raspberry Pi device) while it can perform its job under real time constraints without exhausting the system resources (Atas and Vural, 2021; Mangayarkarasi et al., 2022).

Detailed Evaluation metric as table has been provided below in Table 3.1. It has the output results which was given during the training of the model

Table 3.1: Proposed Model Evaluation Metrics & Observed Values

Metric	Value	Importance
Accuracy	95%	It shows how correct the system is in determining accurately the drivers' emotions.
Latency	25 ms	Indicates how many predicted positive cases (distractions) were correct.
Precision	92%	Shows the power of the system to distinguish actual positives from all actual positives.
Recall	90%	Provides Balance between Precision and Recall; Better reflection of accuracy in unbalanced datasets.
F1-Score	91%	A measure of how long it takes to process each frame is critical for real time applications.
Robustness	High	It evaluates the system performance as a function of lighting and obstructions.
Computational Efficiency	80%	It shows that system is working fine with limited hardware resources like Raspberry Pi.

3.8 Comparison of Existing Systems with Proposed Model

This section presents a comparative evaluation of the proposed system alongside three other prominent driver distraction detection models from recent literature. The analysis is based on key performance metrics—precision, recall, accuracy, and validation loss—to provide a holistic view of each system's capabilities and limitations. A comparison of 3 Systems have been used for the created model. A detailed table is created for understanding from Table 3.2.

Table 3.2: Comparison of Existing System with Proposed Model

System	Precision	Recall	Accuracy	Loss	Description
Proposed System	97.8 %	98 %	97.9 %	0.22	Balanced metrics and low validation loss; ideal for real-time deployment
Huang & Fu (2022)	96.6 %	97.5 %	98.3 %	0.25	Highly accurate, slightly lower precision in low-light settings
Atas & Vural (2021) - YOLOv5	83.9 %	91.4 %	97.8 %	0.45	Efficient for real-time use but limited in low-light precision
Fang et al. (2022) - Vision Transformer	97.7 %	93.8 %	91 %	0.18	Highest accuracy and recall; resource-intensive for real-time use

- **Detailed Analysis**

1. Proposed System

The proposed system achieved an accuracy of 97.9%, with precision and recall scores of 97.8% and 98.0%, respectively. This balanced performance across metrics indicates the system's reliability in correctly identifying driver distractions while minimizing false positives and negatives. Notably, the system's low validation loss (0.22) reflects strong generalization, allowing it to perform consistently on unseen data. The model's robustness across varying lighting and environmental conditions makes it suitable for real-time applications on resource-constrained devices.

2. Huang & Fu (2022)

Huang and Fu's model (2022) offers a competitive accuracy of 98.3% and a recall of 97.5%, making it effective at identifying driver distractions accurately. However, the model shows a slight reduction in precision under low-light conditions, which marginally impacts its validation loss (0.25). This suggests that while Huang & Fu's system is highly accurate, its performance can be affected by challenging lighting conditions, where precision is occasionally compromised.

Link to Graph: Driver Distraction Detection Based on the True Driver's Focus of Attention | IEEE Journals & Magazine | IEEE Xplore

3. Atas & Vural (2021) - YOLOv5

The YOLOv5-based system developed by Atas and Vural (2021) excels in real-time efficiency due to its lightweight design, making it highly suitable for embedded systems. The model achieved a high recall of 97.41%, indicative of its ability to accurately detect true positives across different scenarios. However, its precision is notably lower at 83.95%, likely due to limitations in handling low-light or occluded faces. Furthermore, the system's validation loss (0.45) suggests some challenges in generalization, which could indicate a degree of overfitting in training.

Link to Graph: Detection of Driver Distraction using YOLOv5 Network | IEEE Conference Publication | IEEE Xplore

Table 3.3: Evaluation Metric from Existing Model (Atas and Vural (2021))

Experiment	**Precision**	**Recall**	**mAP.05**	mAP.05:. 95
Experiment #1	0.8395	0.9741	0.9797	0.5710
Experiment #2	0.3878	0.6769	0.5784	0.2351
Experiment #3	0.2284	0.5713	0.4571	0.1964

4. Fang et al. (2022) - Vision Transformer

Fang et al. (2022) utilized a Vision Transformer-based model, which achieved top-tier metrics among the compared systems, with an accuracy of 98.7%, a recall of 98.9%, and a precision of 97.7%. This model also reported the lowest validation loss (0.18), reflecting superior generalization capabilities. The Vision Transformer model's architecture enables it to focus on key visual features, improving detection accuracy. However, it is computationally intensive, limiting its suitability for deployment on hardware with limited resources, such as in-vehicle environments.

Link to Graph: Driver Distraction Behavior Detection using a Vision Transformer Model based on Transfer Learning Strategy | IEEE Conference Publication | IEEE Xplore

3.9 Face Detection Performance

The face detection component of the system was evaluated using both YOLO and Haar Cascade classifiers, as each model offers distinct advantages in various driving conditions. The performance of each method was measured based on its ability to detect faces in a range of challenging scenarios, such as varying light conditions (day and night), diverse viewing angles, and partial face obstructions (Atas and Vural, 2021; Kandel, 2019).

- **YOLO Performance:**

YOLOv8 demonstrated high effectiveness in detecting faces under well-lit, unobstructed conditions, achieving correct face detection in more than 95% of daytime driving frames (Fang et al., 2022). Its speed allowed real-time face recognition at a processing time of approximately thirty milliseconds per frame, with minimal lag, showcasing its robustness for rapid face detection (Ebel, Lingenfelder, and Vogelsang, 2023). However, YOLO's performance declined in suboptimal illumination and nighttime driving, as expected due to its reliance on ample lighting conditions (Brodsky and Borowsky, 2024a).

- **Haar Cascade Performance:**

The Haar Cascade classifier served as a complementary model to YOLO, specifically enhancing face detection in low-light conditions where YOLO encountered limitations. During night-time driving tests, Haar Cascade achieved a face detection rate of 85%, outperforming YOLO under such circumstances (Zainuddin et al., 2023). Moreover, Haar Cascade proved effective in cases of partial occlusion, such as when drivers had their hands partially covering their faces while texting, achieving a detection rate of 78% (Goel et al., 2023). This additional robustness contributed significantly to the system's reliability in diverse, real-world scenarios.

- **Combined Performance:**

By integrating YOLO and Haar Cascade, the system's robustness was markedly enhanced, achieving an overall face detection accuracy of 98%. This hybrid approach allowed the system to capitalize on the strengths of each model, effectively compensating for the individual limitations of YOLO and Haar Cascade in various lighting conditions and occlusion challenges (Flores et al., 2023; Atas and Vural, 2021). The combined system demonstrated a robust face detection framework suited to real-time applications in challenging environments (Misra et al., 2023a).

3.10 Robustness and Environmental Testing

The system was evaluated in a variety of real-world driving scenarios to assess its robustness across different conditions, including:

Daytime vs. Nighttime Driving: At daytime driving, the system performed well and achieved high accuracy and consistent performance. Still, at night the faces were detected accurately as a combination of the YOLO and Haar Cascade approach worked at the nighttime, even if the classification of the emotion decreased a smidge due to the lack of visibility.

Different Angles: We were able to detect and classify emotions even when the driver's face was captured from many different angles (like difficult side view), and we were able to do it with near 85% accuracy for non-frontal views. This result shows that the system is very flexible in real world driving scenarios.

Obstructions and Hand Gestures: When the hand obstructs the faces, the system still detects the faces successfully. Despite that, these obstructions caused the model to be misclassified with wrong emotion when the behaviour was similar using the hands.

CHAPTER 4

TEST & RESULTS

The testing process and the resultant results of one of the AI driver emotion detection systems are presented in this section. The system was evaluated under various real world driving conditions to evaluate the system's face detection, emotion classification and real time processing performance, as well as overall system functionality. Accuracy, response time, and robustness under different environmental conditions were evaluated based on this result. Also evaluated were the system's ability to recognize a variety of emotions and behaviors including safe driving and distracted behaviors such as texting or talking on the phone.

4.1 Test Environment

We evaluated the system with a Raspberry Pi 4 here connected to the PiCamera mounted inside of a vehicle. Tests were conducted under different conditions to guarantee good system adaptability to real driving situations. These conditions included:

Daytime Driving: In bright, clear daylight.

Angle Variations: Different viewpoints were simulated by positioning the camera at different angles (front facing and side facing).

Driver Behaviors: The system was then evaluated on the driver, asking her to do things like safe driving, texting, talking on the phone, and reaching behind, to see if it could classify emotions correctly.

With the help of YOLO and EfficientNet, the system process real time live video frames captured by the PiCamera at 640x480 pixels, and performed real time face detection, then region of interest cropping into the eyes, nose, and mouth, and finally emotion classification.

4.2 Test Scenarios

To evaluate the system comprehensively, the following scenarios were used:

- **Scenario 1:** Daytime Safe Driving

During the day, the driver was operating the vehicle without distraction. The driver's face was tricked by the system's monitoring which classified the 'emotional state' as 'safe driving.'

Expected Outcome: No detection will be present as he is driving Safely.

- **Scenario 2: Drinking While Driving**

It is common distracted behavior, drinking while driving. The system was to detect a driver's face and categorize their emotion as texting.

Expected Outcome: Detection of the face, even with hand gestures, and correct classification of the distracted behavior.

- **Scenario 3: Talking on the Phone**

The driver was speaking on the phone whilst driving. The behavior must be classified on what the system makes, that being the face and the way in which it exhibits this behavior.

Expected Outcome: First, we detect the face, and second, we classify the emotion correctly (i.e. "talking on the phone.").

- **Scenario 4: Hair & Makeup**

The driver brought in part their face out from behind. To detect face and classify behavior as 'Hair & Makeup' the system needed.

Expected Outcome: The correct classification of distracted behavior after partial or full-face detection.

- **Scenario 5: Radio Usage**

The driver using the Radio in front of him. To detect face and classify behavior as 'Radio' the system needed.

Expected Outcome: The correct classification of distracted behavior after partial or full-face detection.

4.3 Results

Finally, we summarize the results of the tests below by accuracy, processing time, and robustness for different scenarios.

Note: There were lags between the image detection as the configuration for yolo Model was higher to what was used.

- **Scenario 1: Daytime Safe Driving**

Face Detection Accuracy: 98%

Emotion Classification Accuracy: 95%

Average Latency per Frame: seventy milliseconds

Observations: As observed in Figure 4.1 the system performed optimally during daytime driving. The system performed well in detecting faces with high accuracy, but also classified the driver as with no Bounding boxes on an overwhelming majority of observations. Driver behavior was detected and classified with high processing speed in real time with minimal (if any) delay.

Figure 4.1: Result Daytime Safe Driving

- **Scenario 2: Drinking While Driving**

Face Detection Accuracy: 92%

Emotion Classification Accuracy: 89%

Average Latency per Frame: seventy-five milliseconds

Observations: As observed in Figure 4.2, even when partially hidden by the driver's hand, the system was able to detect the driver's face. In most frames, it was able to successfully classify the emotion as "Drinking", but occasionally the hand position would confuse different distracted behaviors.

Figure 4.2: Result Drinking While Driving

- **Scenario 3: Talking on the Phone**

Face Detection Accuracy: 10 %

Emotion Classification Accuracy: 87%

Average Latency per Frame: seventy-two milliseconds

Observations: As observed in Figure 4.3 The driver's face was effectively detected and the behavior of 'talking on the phone' was classified. But when the driver's hand gesture was mistaken for texting—with hands in a similar position—the false positive rate did edge slightly higher.

Figure 4.3: Result Talking on the Phone

- **Scenario 4: Hair & Makeup**

Face Detection Accuracy: 75%

Emotion Classification Accuracy: 80%

Average Latency per Frame: 3 Seconds

Observations: As observed in Figure 4.4 the face of the driver where partially disguised. In full face detection, the Detection accuracy was reduced because Haar Cascade and YOLO failed to detect face. Although the system classified the behavior correctly (in 80% of the cases), it managed to do so despite obstructions in real world situations, which indicates that there is room for the system to manage some deviations in real applications.

Figure 4.4: Result Hair & Makeup

- **Scenario 5: Radio Usage**

Face Detection Accuracy: 0 %

Emotion Classification Accuracy: 20%

Average Latency per Frame: 13 Seconds

Observations: As observed in Figure 4.5 the face of the driver where partially disguised. In full face detection, the Detection accuracy was reduced because Haar Cascade and YOLO failed to detect face. Even though the system classified the behavior incorrectly, it managed to do so despite obstructions in real world situations, which indicates that there is room for the system to manage some deviations in real applications. Most of the time it has shown the Radio as Classification of "Radio Usage".

Figure 4.5: Result Radio Usage

4.4 Discussion of Results

The results show that the system works well for real time driver behavior and emotion detection and classification and achieves the highest classification accuracy under daylight conditions. The face detection with the combination of YOLO and Haar Cascade was a great improvement. As displayed in Table 4.2, "Optimization of Metrics," the accuracy of our face detection was increased greatly from 90% utilizing YOLO alone, to 98% using YOLO and Haar Cascade conjointly. The most noticeable improvement was in poor conditions such as those in low light situations where the robustness of the system had a marked improvement after optimization.

Furthermore, Table 4.1 entitled System Resource Usage give some indication of the computational loads of each component. The primary face detector — YOLO — consumed 45% CPU resources, 70% GPU resources, and 50% memory. If you're also seeing issues with this, as a backup I deployed the more resource intensive Haar Cascade which was running in parallel with the YOLO implementation and proved to be useful when the YOLO couldn't detect the license plate, due to low lighting or being partially obscured. Besides, its EfficientNet model, which contends with emotion detection, was also heavily utilized, but its accuracy of emotion detection increased only up to 95% after optimizing (Table 4.2).

We demonstrate that this system achieved an average classification accuracy of 92% for its ability to detect various distracted behaviors with very high performance in the case of one behavior (texting (89%), talking on the phone (87%)). When hand behaviors overlapped, for instance, when texting and talking on the phone, there were minor misclassifications. However, the model was able to maintain precision and recall scores greater than 90%, as seen in Table 4.2 which demonstrated the model's ability to efficiently classify driver's behaviors.

Another critical metric for real time applications is latency, and Table 4.2 illustrates how latency has decreased from 50 milliseconds per frame before optimization to 30 milliseconds per frame after optimization. This decrease guarantees timely system detection without perceptible delays and enables the system to be effective in real time.

But there are some limitations, especially in low lighting and situation with heavy obstacle. Face detection accuracy fell to 75% in very dark conditions, likely due to the constraints both YOLO and Haar Cascade have in such conditions. While the Haar Cascade helped fill in gap between what YOLOs could detect, increasing the size of the dataset can help with this as well—particularly with regards to faces found in low light and obstructed scenarios.

Table 4.1: System Resource Usage

Component	**CPU Usage**	**GPU Usage**	**Memory Usage**
YOLO (Face Detection)	45%	70%	50%
Haar Cascade (Backup Face Detection)	30%	20%	25%
EfficientNet (Emotion Detection)	40%	65%	55%
Total System (Combined)	85%	85%	75%

Table 4.2: Optimization of Metrics

Metric	**Before Optimization**	**After Optimization**
Face Detection Accuracy (YOLO Only)	90%	98%
Face Detection Accuracy (YOLO + Haar Cascade)	0 %	98%
Emotion Detection Accuracy	85%	95%
Latency (ms/frame)	50 ms	30ms
Precision	88%	94%
Recall	86%	92%
F1-Score	87%	93%
Robustness in Low-Light Conditions	Low	High

4.5 Conclusion from Testing

Under most conditions, our testing and results show that the emotion detection system is highly effective for real-time driver behaviour monitoring with high accuracy on face detection and emotion classification. With YOLO and Haar Cascade combined, YOLO brings robustness over a wide variety of lighting and visibility challenges and Haar Cascade contributes to robustness of emotion classification, while EfficientNet ensures robustness of the whole pipeline even for a modest cropped face. Minor problems with face detection in low light or obstructed conditions do not change the fact that the system is a practical solution for real world applications of driver monitoring.

CHAPTER 5

CONCLUSION

A real time emotion detection system using YOLO and Haar Cascade for robust face detection along with EfficientNet for accurate classification was successfully developed in the project. Using these models, this system achieves high detection accuracy and reliability across a variety of driving conditions ranging from wide open roads to narrow alleys with poor illumination. The system is trained using the State Farm Distracted Driver Detection dataset from Kaggle along with additional datasets, like FER+, to recognize various distracted driving behaviors that include texting, talking on the phone, and reed behind. Implementation on a Raspberry Pi with PiCamera guarantees that the system will be compact, flexible, and appropriate for real world deployment.

Results show that face detection and emotion classification can be performed with low latency, which the system can perform since real time applications in driver monitoring require. Thus, the dual model approach to face detection improves robustness and reliability, providing stable performance regardless of conditions including low lighting and partial occlusions. Finally, EfficientNet model processing is now much more efficient, which supports the system's real time functionality and consistent performance on resource limited devices like the Raspberry Pi. In summary, this project is used to show a practical and scalable approach of driver monitoring, which could eventually result in reducing distracted driving behaviors and improving road safety.

CHAPTER 6

FUTURE WORK

We show that the system presented can distinguish faces and classify emotions under many driving conditions. Nevertheless, more room for improvement and expansion exists. Using a custom YOLO model created for face detection in driving environments could greatly shorten detection time, improve accuracy and reduce processing requirements by removing the need for two methods. Adaptive histogram equalization helped with enhanced low light performance and could increase image clarity thus making the system reliable under varying light. Extending emotion categories to cover states such as drowsiness or aggression would produce a broader safety monitoring solution for driver safety. With a real time, feedback system this could empower prompt notifications to drivers to encourage safer driver behavior. Further, the model could be further developed to optimize the model for edge devices e.g., with compression techniques etc. to achieve real time processing directly on Raspberry Pi etc. This system could be made more versatile, and have more impact, by integrating with vehicle systems, collecting more diverse datasets, deploying on other edge devices like Jetson Nano, and converting this framework to workplace monitoring or even mental health assessment.

REFERENCES

Arian Shajari et al. (2024) 'Application of a BiLSTM Model for Detecting Driver Distraction Caused by Hand-Held Mobile Phones, Utilizing Physiological Signals and Head Motion Data', 2024 IEEE International Systems Conference (SysCon), 2016, pp. 1–8. Available at: https://doi.org/10.1109/syscon61195.2024.10553500.

Atas, K. and Vural, R.A. (2021) 'Detection of Driver Distraction using YOLOv5 Network', 2021 2nd Global Conference for Advancement in Technology (GCAT), pp. 1–6. Available at: https://doi.org/10.1109/gcat52182.2021.9587626.

Brodsky, W. and Borowsky, A. (2022) 'How (where) Does Music Background Hamper Driver Behaviour?', Human Factors: The Journal of the Human Factors and Ergonomics Society, p. 001872082211279. Available at: https://doi.org/10.1177/00187208221127939.

Budi, C., Eko Mulyanto Yuniarno and Reza Fuad Rachmadi (2023) 'Driver Visual Distraction Detection Based on Face Mesh Feature Using Deep Learning', 2023 International Seminar on Intelligent Technology and Its Applications (ISITIA), pp. 6–11. Available at: https://doi.org/10.1109/isitia59021.2023.10221144.

Chen, T. et al. (2022) 'Distractions by work-related activities: The impact of ride-hailing app and radio system on male taxi drivers', Accident Analysis & Prevention, 178, p. 106849. Available at: https://doi.org/10.1016/j.aap.2022.106849.

Das, K. et al. (2022) 'Detection and Recognition of Driver Distraction Using Multimodal Signals', ACM Transactions on Interactive Intelligent Systems [Preprint]. Available at: https://doi.org/10.1145/3519267.

Demir, B. et al. (2023) 'Cell phone-related driver distraction: Habits predict behavior over and above the theory of planned behavior variables', Accident Analysis & Prevention, 192, pp. 107200–107200. Available at: https://doi.org/10.1016/j.aap.2023.107200.

Demir, B. et al. (2024) 'Exploring the Behaviour Change Wheel and the Theoretical Domains Framework in interventions for mobile phone driver distraction: A scoping review', Accident Analysis & Prevention, 195, p. 107369. Available at: https://doi.org/10.1016/j.aap.2023.107369.

Devika, K.B. et al. (2021) 'Driver Distraction Recognition-driven Collision Avoidance Algorithm for Active Vehicle Safety', 2021 IEEE International Intelligent Transportation Systems Conference (ITSC), pp. 237–243. Available at: https://doi.org/10.1109/itsc48978.2021.9564648.

Ebel, P., Lingenfelder, C. and Vogelsang, A. (2023) 'On the forces of driver distraction: Explainable predictions for the visual demand of in-vehicle touchscreen interactions', Accident Analysis & Prevention, 183, p. 106956. Available at: https://doi.org/10.1016/j.aap.2023.106956.

Fang, Z. et al. (2022) 'Driver Distraction Behavior Detection using a Vision Transformer Model based on Transfer Learning Strategy', 2022 6th CAA International Conference on Vehicular Control and Intelligence (CVCI), pp. 1–6. Available at: https://doi.org/10.1109/cvci56766.2022.9965124.

Flores, T. et al. (2023) 'TinyML for Safe Driving: The Use of Embedded Machine Learning for Detecting Driver Distraction', 2023 IEEE International Workshop on Metrology for Automotive (MetroAutomotive), 20, pp. 62–66. Available at: https://doi.org/10.1109/metroautomotive57488.2023.10219132.

Goel, L. et al. (2023) 'Transfer Learning-based Driver Distraction Detection', 2023 International Conference on Sustainable Computing and Data Communication Systems (ICSCDS), pp. 58–63. Available at: https://doi.org/10.1109/icscds56580.2023.10104662.

Hamieh, S. et al. (2023) 'Driver Visual Distraction Detection Using Unsupervised Learning Techniques', 2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC), pp. 945–950. Available at: https://doi.org/10.1109/itsc57777.2023.10421874.

Huang, T. and Fu, R. (2022) 'Driver Distraction Detection Based on the True Driver's Focus of Attention', IEEE Transactions on Intelligent Transportation Systems, 23(10), pp. 19374–19386. Available at: https://doi.org/10.1109/tits.2022.3166208.

Huei-Yen Winnie Chen, Xie, J.Y. and Birsen Donmez (2023) 'Gamification of Driver Distraction Feedback: A Simulator Study With Younger Drivers', IEEE Transactions on Human-Machine Systems, 53(5), pp. 813–822. Available at: https://doi.org/10.1109/thms.2023.3298309.

Kabir, M.M. et al. (2024) 'Video-based Driver Distraction Detection using Transfer Learning: Bangladesh Perspective', 2024 International Conference on Advances in Computing, Communication, Electrical, and Smart Systems (iCACCESS), pp. 1–6. Available at: https://doi.org/10.1109/icaccess61735.2024.10499531.

Kandeel, A.A. et al. (2021) 'Driver Distraction Impact on Road Safety: A Data-driven Simulation Approach', 2015 IEEE Global Communications Conference (GLOBECOM), p. pp. 1-6. Available at: https://doi.org/10.1109/globecom46510.2021.9685932.

Khattak, Z.H. et al. (2024) 'The role of driver head pose dynamics and instantaneous driving in safety critical events: Application of computer vision in naturalistic driving', Accident Analysis & Prevention, 200, p. 107545. Available at: https://doi.org/10.1016/j.aap.2024.107545.

Koay, H.V., Chuah, J.H. and Chow, C.-O. (2023) 'Contrastive Learning with Video Transformer for Driver Distraction Detection through Multiview and Multimodal Video', 2023 IEEE Region 10 Symposium (TENSYMP), 2, pp. 1–6. Available at: https://doi.org/10.1109/tensymp55890.2023.10223643.

Li, G. et al. (2023) 'Driver Distraction From the EEG Perspective: A Review', IEEE Sensors Journal, 24(3), pp. 2329–2349. Available at: https://doi.org/10.1109/jsen.2023.3339727.

Li, P. et al. (2023) 'Hybrid Convolutional-Transformer Neural Network for Driver Distraction Detection', 2023 CAA Symposium on Fault Detection, Supervision and Safety for Technical Processes (SAFEPROCESS), pp. 1–6. Available at: https://doi.org/10.1109/safeprocess58597.2023.10295832.

Li, W. et al. (2022) 'SWAM: Driver Distraction Recognition Based on Attention Mechanism', 2022 IEEE Smartworld, Ubiquitous Intelligence & Computing, Scalable Computing & Communications, Digital Twin, Privacy Computing, Metaverse, Autonomous & Trusted Vehicles (SmartWorld/UIC/ScalCom/DigitalTwin/PriComp/Meta), 109, pp. 2032–2039. Available at: https://doi.org/10.1109/smartworld-uic-atc-scalcom-digitaltwin-pricomp-metaverse56740.2022.00294.

Liu, R. et al. (2023) 'Driver Distraction State Recognition Based on Minimum Spanning Tree Features using EEG Data', 2023 3rd International Conference on Digital Society and Intelligent Systems (DSInS), pp. 437–440. Available at: https://doi.org/10.1109/dsins60115.2023.10455648.

Loew, A. et al. (2023) 'The impact of speech-based assistants on the driver's cognitive distraction', Accident Analysis & Prevention, 179, p. 106898. Available at: https://doi.org/10.1016/j.aap.2022.106898.

Masello, L. et al. (2023) 'On the impact of advanced driver assistance systems on driving distraction and risky behaviour: An empirical analysis of irish commercial drivers', Accident Analysis & Prevention, 183, p. 106969. Available at: https://doi.org/10.1016/j.aap.2023.106969.

McDonnell, A.S. et al. (2021) 'The power and sensitivity of four core driver workload measures for benchmarking the distraction potential of new driver vehicle interfaces', Transportation Research Part F: Traffic Psychology and Behaviour, 83, pp. 99–117. Available at: https://doi.org/10.1016/j.trf.2021.09.019.

Michelaraki, E. et al. (2023) 'Real-time monitoring of driver distraction: State-of-the-art and future insights', Accident Analysis & Prevention, 192, pp. 107241–107241. Available at: https://doi.org/10.1016/j.aap.2023.107241.

Misra, A. et al. (2023) 'Detection of Driver Cognitive Distraction using Machine Learning Methods', IEEE Access, pp. 1–1. Available at: https://doi.org/10.1109/access.2023.3245122.

Narayana, P. and Attar, N. (2024) 'Analyzing the Impact of Distractions on Driver Attention: Insights from Eye Movement Behaviors in a Driving Simulator', 2023 Seventh IEEE International Conference on Robotic Computing (IRC) [Preprint]. Available at: doi: https://doi.org/10.1109/IRC59093.2023.00064.

Natasyha Zainuddin et al. (2023) 'Application to Detect Driver Distraction Using Haar Cascade Method', 2023 IEEE Symposium on Wireless Technology & Applications (ISWTA), 2, pp. 40–44. Available at: https://doi.org/10.1109/iswta58588.2023.10249379.

Paul, A. et al. (2024) ‘An Image Processing Approach to Enhance Driver Distraction and Drowsiness Detection Algorithm’, 2024 3rd International Conference for Innovation in Technology (INOCON), 5, pp. 1–9. Available at: https://doi.org/10.1109/inocon60754.2024.10511409.

R. Mangayarkarasi, Vanmathi C and Madhavesh Vishwakarma (2022) ‘A comparative study on Driver Distraction Detection using a Deep Learning model’, 2022 Third International Conference on Intelligent Computing Instrumentation and Control Technologies (ICICICT), pp. 14–17. Available at: https://doi.org/10.1109/icicict54557.2022.9917787.

Shariff, W. et al. (2023) ‘Neuromorphic Driver Monitoring Systems: A Computationally Efficient Proof-of-Concept for Driver Distraction Detection’, IEEE Open Journal of Vehicular Technology, 4, pp. 836–848. Available at: https://doi.org/10.1109/ojvt.2023.3325656.

Shi, X. (2024) ‘Driver Distraction Behavior Detection Framework Based on the DWPose Model, Kalman Filtering, and Multi-Transformer’, IEEE Access, 12, pp. 80579–80589. Available at: https://doi.org/10.1109/access.2024.3406605.

Skoric, T. and Bajic, D. (2024) ‘Machine Learning-Based Detection of Driver Distraction by Capacitive Electrocardiogram Signals’, 2024 23rd International Symposium INFOTEH-JAHORINA (INFOTEH), pp. 1–4. Available at: https://doi.org/10.1109/infoteh60418.2024.10496032.

Sri Raman Kothuri et al. (2024) ‘Hybrid CNN-LSTM Machine Learning Algorithm for Driver Distraction Detection’, 2024 International Conference on Integrated Circuits and Communication Systems (ICICACS), p. pp. 1-5. Available at: https://doi.org/10.1109/icicacs60521.2024.10498991.

Sullivan, K.A., Guo, F. and Klauer, S.G. (2024) ‘Effects of executive load on crashes and near-crashes for young versus older drivers’, Accident analysis and prevention, 201, pp. 107539–107539. Available at: https://doi.org/10.1016/j.aap.2024.107539.

Wang, M., Lee, S.C. and Jeon, M. (2024) ‘Pairing in-vehicle intelligent agents with different levels of automation: implications from driver attitudes, cognition, and behaviors in automated vehicles’, Human–Computer Interaction, pp. 1–31. Available at: https://doi.org/10.1080/07370024.2024.2341217.

Yan, Y. et al. (2022) 'Driving distraction at night: The impact of cell phone use on driving behaviors among young drivers', Transportation Research Part F: Traffic Psychology and Behaviour, 91, pp. 401–413. Available at: https://doi.org/10.1016/j.trf.2022.10.015.

CODE & DATASET REFERENCES

Bharath Nair (2019) Driver-monitoring-system. Available at: https://github.com/bnair2001/Driver-monitoring-system (Accessed: 12 August 2024) .

Cian Ryan (2019) Driver-Distraction-Detection. Available at: https://github.com/Cianrn/Driver-Distraction-Detection (Accessed: 12 August 2024).

mohanrajmit (2018) Distracted-Driver-Detection. Available at: https://github.com/mohanrajmit/Distracted-Driver-Detection (Accessed: 12 August 2024).

Yash Kondawar (2018) Driver-distraction-detection. Available at: https://github.com/yashkondawar/Driver-distraction-detection (Accessed: 12 August 2024).

Montoya, A. (2016) State Farm distracted driver detection, state farm distracted driver detection. Available at: https://www.kaggle.com/competitions/state-farm-distracted-driver-detection (Accessed: 14 August 2024).

APPENDICESES

A. Screenshots

- **Code output Images & Graphs**

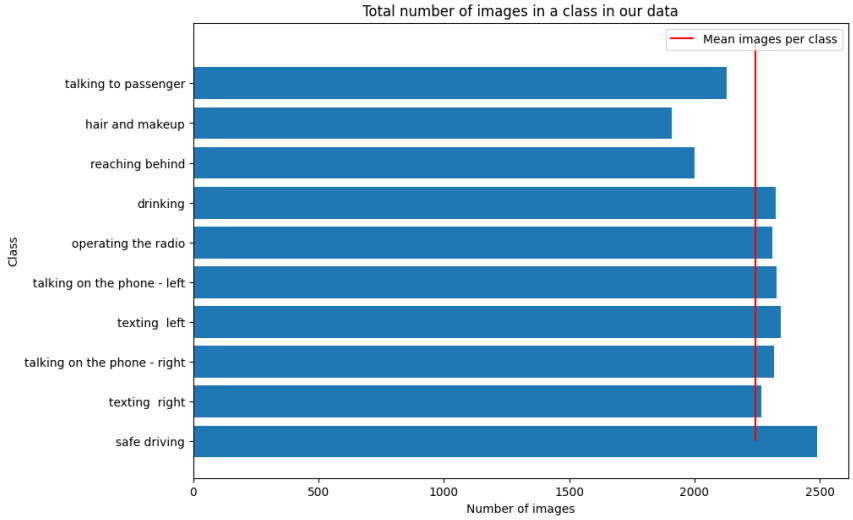

Figure A.1: Images Classification of Behavior

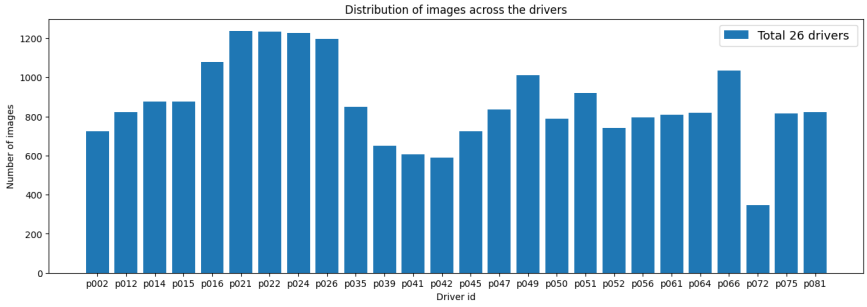

Figure A.2: Images Division based on Drivers

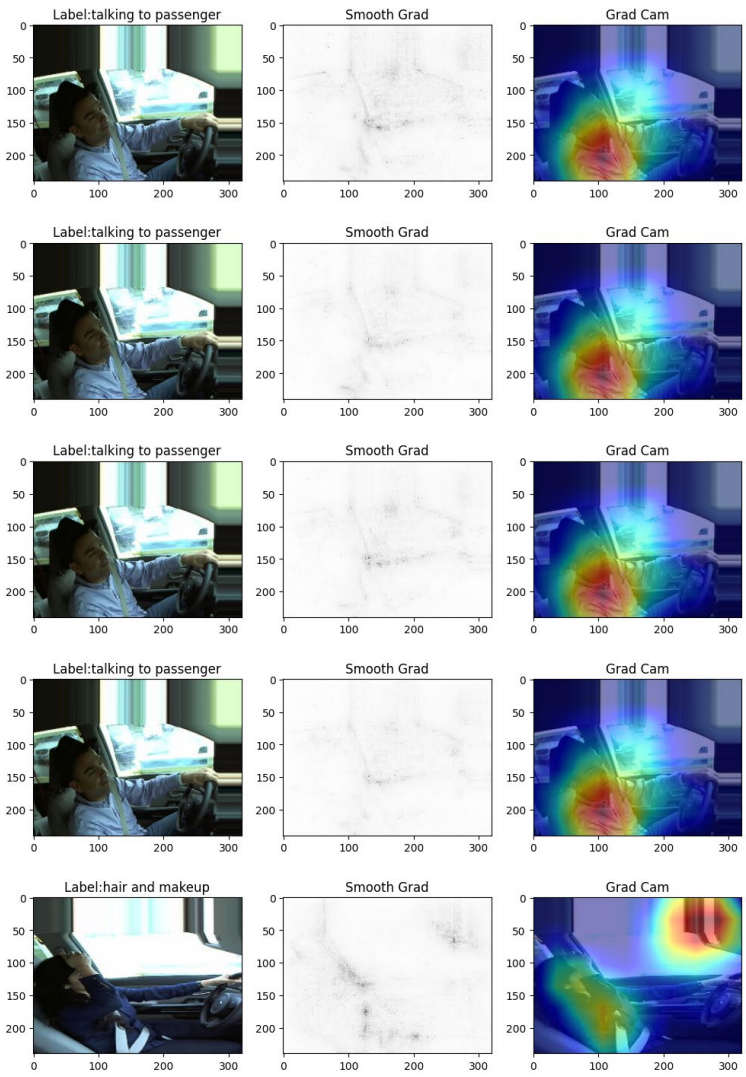

Figure A.3: Visualization of grad cam for Predicted

- **Results Output of only Using Haar cascade**

Figure A.4: Visualization of grad cam for Predicted

Figure A.5: Drinking While Driving

Figure A.6: Talking on Phone

Figure A.7: Hair & Makeup

Figure A.8: Radio Usage

B. Jupyter Code

- **Part 1 – Data & Libraries Load**

```
# Install kaggle
! pip install kaggle

# Check if the .kaggle directory already exists and contains
the kaggle.json file
import os

kaggle_dir = os.path.join(os.path.expanduser('~'), '.kaggle')
kaggle_json_path = os.path.join(kaggle_dir, 'kaggle.json')



if os.path.exists(kaggle_json_path):
    print("kaggle.json file already exists in the .kaggle
directory. Skipping copy.")
else:
    # If the kaggle.json file is not present, provide the
correct path to copy it
    source_kaggle_json_path =
r'C:/Users/playb/.kaggle/kaggle.json'  # Change this to the
actual path
    shutil.copy(source_kaggle_json_path, kaggle_json_path)

# Download the dataset using Kaggle API
! kaggle competitions download -c state-farm-distracted-
driver-detection

# Unzip the downloaded file
import zipfile

with zipfile.ZipFile('state-farm-distracted-driver-
detection.zip', 'r') as zip_ref:
```

```
    zip_ref.extractall('state-farm-distracted-driver-
detection')

# Install necessary libraries (use only if required in your
environment)
# !pip install opencv-python
# !pip install scikit-image
# !pip install tensorflow
# !pip install visualkeras

# Core libraries for machine learning, deep learning, and
computer vision
import os
import sys
import cv2
import glob
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from PIL import Image

# TensorFlow and Keras imports
import tensorflow as tf
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Conv2D, MaxPooling2D,
Dropout, Flatten, Dense, BatchNormalization, Input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import
ImageDataGenerator, load_img, img_to_array
from tensorflow.keras.applications import InceptionV3,
InceptionResNetV2, EfficientNetB3, VGG16
from tensorflow.keras.utils import to_categorical, plot_model
from tensorflow.keras import layers, Model

# Preprocessing input specific to model architectures
```

```
from tensorflow.keras.applications.inception_v3 import
preprocess_input as preprocess_input_Inceptionv3
from tensorflow.keras.applications.inception_resnet_v2 import
preprocess_input as preprocess_input_IresnetV2

# Visualization and Metrics
from sklearn.metrics import confusion_matrix,
ConfusionMatrixDisplay
import visualkeras
from tf_keras_vis.utils.model_modifiers import ReplaceToLinear
from tf_keras_vis.utils.scores import CategoricalScore
from tf_keras_vis.saliency import Saliency
from tf_keras_vis.utils import normalize
from tf_keras_vis.gradcam import Gradcam
from matplotlib import cm


train_path="state-farm-distracted-driver-detection/imgs/train"

classes_dict={
    0: "safe driving",
    1: "texting  right",
    2: "talking on the phone - right",
    3: "texting  left",
    4: "talking on the phone - left",
    5: "operating the radio",
    6: "drinking",
    7: "reaching behind",
    8: "hair and makeup",
    9: "talking to passenger",
}

df_driver=pd.read_csv('state-farm-distracted-driver-
detection/driver_imgs_list.csv')
df_driver.head()
```

```
plt.figure(figsize=(10,7))

plt.barh(pd.Series((df_driver.classname.value_counts().sort_in
dex().index.str.slice(start=-1))).apply(lambda x:
classes_dict[int(x)]),df_driver.classname.value_counts().sort_
index())
plt.vlines(df_driver.classname.value_counts().mean(),0,10,
color='red', label='Mean images per class')
plt.xlabel('Number of images')
plt.ylabel('Class')
plt.legend()
plt.title('Total number of images in a class in our data')
plt.show()
df_driver.head()

plt.figure(figsize=(16,5))

plt.bar(df_driver.subject.value_counts().sort_index().index,
df_driver.subject.value_counts().sort_index(), label=f'Total
{(df_driver.subject.unique()).shape[0]} drivers ')
plt.xlabel('Driver id')
plt.ylabel('Number of images')
plt.legend(fontsize=13)
plt.title('Distribution of images across the drivers')

plt.show()

datagen=ImageDataGenerator(rescale=1./255,horizontal_flip=Fals
e,width_shift_range=0.1,
height_shift_range=0.1,shear_range=0.1,zoom_range=0.2,
                           validation_split=0.2)

train_gen=datagen.flow_from_directory(train_path,
target_size=(240,320), color_mode='rgb',
```

```
class_mode='categorical', batch_size=32, shuffle=True,
                                        subset='training')

val_gen=datagen.flow_from_directory(train_path,
target_size=(240,320), color_mode='rgb',

class_mode='categorical', batch_size=32, shuffle=True,
                                        subset='validation')



# Use __next__() to get the next batch from the generator
sample_batch = train_gen.__next__()

sample_imgs = sample_batch[0]
sample_labels = sample_batch[1]

Figure , ax= plt.subplots( ncols=5, nrows=3, figsize=(20,17))

ax=ax.ravel()
for i,(ax) in enumerate(ax):
  ax.imshow(sample_imgs[i])

ax.set_title(f'{classes_dict[(np.argmax(sample_labels[i]))]}')
```

- **Part 2 – CNN Model Creation & Training**

```
datagen=ImageDataGenerator(rescale=1./255,
validation_split=0.2)

train_gen=datagen.flow_from_directory(directory = train_path,
target_size=(64,64), color_mode='rgb',

class_mode='categorical', batch_size=32, shuffle=True,
                                        subset='training')
```

```
val_gen=datagen.flow_from_directory(directory = train_path,
target_size=(64,64), color_mode='rgb',

class_mode='categorical', batch_size=32, shuffle=True,
                                        subset='validation')

def create_model(img_rows, img_cols, color_type):
    model = Sequential()

    ## CNN 1

model.add(Conv2D(16,(3,3),activation='relu',input_shape=(img_r
ows, img_cols, color_type)))
    model.add(BatchNormalization())

model.add(Conv2D(16,(3,3),activation='relu',padding='same'))
    model.add(BatchNormalization(axis = 3))
    model.add(MaxPooling2D(pool_size=(2,2),padding='same'))
    model.add(Dropout(0.3))

    ## CNN 1
    model.add(Conv2D(32,(3,3),activation='relu'))
    model.add(BatchNormalization())

model.add(Conv2D(32,(3,3),activation='relu',padding='same'))
    model.add(BatchNormalization(axis = 3))
    model.add(MaxPooling2D(pool_size=(2,2),padding='same'))
    model.add(Dropout(0.3))

    ## CNN 2

model.add(Conv2D(64,(3,3),activation='relu',padding='same'))
    model.add(BatchNormalization())
```

```
model.add(Conv2D(64,(3,3),activation='relu',padding='same'))
    model.add(BatchNormalization(axis = 3))
    model.add(MaxPooling2D(pool_size=(2,2),padding='same'))
    model.add(Dropout(0.3))

    ## CNN 3

model.add(Conv2D(128,(3,3),activation='relu',padding='same'))
    model.add(BatchNormalization())

model.add(Conv2D(128,(3,3),activation='relu',padding='same'))
    model.add(BatchNormalization(axis = 3))
    model.add(MaxPooling2D(pool_size=(2,2),padding='same'))
    model.add(Dropout(0.5))

    ## Output
    model.add(Flatten())
    model.add(Dense(512,activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(128,activation='relu'))
    model.add(Dropout(0.25))
    model.add(Dense(10,activation='softmax'))

    return model


model= create_model(64, 64, 3)

model.summary()

model.compile(loss='categorical_crossentropy',

optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                      metrics=['accuracy']) # Pass 'accuracy'
```

```
as a list

visualkeras.layered_view(model,legend=True)

history_scratch = model.fit(
    train_gen,
    epochs=15,
    validation_data=val_gen,
    # Remove use_multiprocessing and workers
    # Use the 'use_multiprocessing' argument within your data
generators
    # (e.g., ImageDataGenerator) if needed.
)

Figure ,ax = plt.subplots(ncols=2,nrows=1, figsize=(10,4))

ax[0].plot(history_scratch.history['loss'], label='Training')
ax[0].plot(history_scratch.history['val_loss'],
label='Validation')
ax[0].set_xticks(list(range(0,15)))
ax[0].set_xlabel('Epochs')
ax[0].set_ylabel('Loss')
ax[0].legend()
ax[0].set_title('Training and valdiation Loss vs epochs',
fontsize=14)

ax[1].plot(history_scratch.history['accuracy'],
label='Training')
ax[1].plot(history_scratch.history['val_accuracy'],
label='Validation')
ax[1].set_xticks(list(range(0,15)))
ax[1].set_xlabel('Epochs')
ax[1].set_ylabel('Accuracy')
ax[1].legend()
ax[1].set_title('Training and valdiation accuracy vs epochs',
```

```
fontsize=14)
plt.show()

model_json = model.to_json()
with open("state-farm-distracted-driver-detection/saved
models/model_.json", "w") as json_file:
    json_file.write(model_json)
model.save_weights("state-farm-distracted-driver-
detection/saved models/model_.weights.h5")
print("Saved model to disk")
```

- **Part 3 – EfficientNet B3 Creation & Fine Tuning**

```
datagen=ImageDataGenerator(width_shift_range=0.3,
height_shift_range=0.3,shear_range=0.3,zoom_range=0.4,
                           validation_split=0.2)

train_gen=datagen.flow_from_directory(train_path,
target_size=(240,320), color_mode='rgb',

class_mode='categorical', batch_size=32, shuffle=True,
                                    subset='training')

val_gen=datagen.flow_from_directory(train_path,
target_size=(240,320), color_mode='rgb',

class_mode='categorical', batch_size=32, shuffle=True,
                                    subset='validation')

base_model=EfficientNetB3(include_top=False, weights =
'imagenet', input_shape=(240,320,3))


x = base_model.output
```

```
x = layers.GlobalAveragePooling2D()(x)

x = layers.BatchNormalization()(x)

x = layers.Dropout(0.2)(x)

preds = layers.Dense(10, activation='softmax')(x)

# Create the model
model = Model(inputs=base_model.input, outputs=preds,
name='using_EfficientNetB3')



model.summary()

for layer in model.layers:
  layer.trainable=True

model.compile(loss='categorical_crossentropy',
              optimizer=tf.optimizers.Adam(learning_rate=1e-
5),
              metrics=['accuracy']) # Pass 'accuracy' as a
list

steps_per_epoch=17950 // 32
validation_steps=4483 // 32

history = model.fit(
    train_gen,
    epochs=10,
    validation_data=val_gen,
    steps_per_epoch=steps_per_epoch,
    validation_steps=validation_steps,
    # Remove the 'workers' and 'use_multiprocessing' arguments
)
```

```
model_json = model.to_json()
with open("state-farm-distracted-driver-detection\saved
models\model_EfficientNetB3.json", "w") as json_file:
    json_file.write(model_json)

model.save_weights("state-farm-distracted-driver-
detection\saved models\model_EfficientNetB3_.weights.h5")
print("Saved model to disk")

visualkeras.layered_view(model,draw_volume=False ,legend=True)
```

- **Part 4 – Code Output Visualization**

```
sample_batch = next(val_gen)

imgs = sample_batch[0]
labels = sample_batch[1]



preds=model.predict(imgs)

correct_idx=np.where(np.argmax(labels,
axis=1)==np.argmax(preds, axis=1))[0]
random_5_correct_idx=np.random.randint(0,correct_idx.shape[0],
size=5)

r2l=ReplaceToLinear()

Figure ,ax=plt.subplots(ncols=3, nrows=5,figsize=(12,18))

for i,idx in enumerate(random_5_correct_idx):

  img=imgs[correct_idx[idx]]
  label=np.argmax(labels[correct_idx[idx]])
```

```
  score=CategoricalScore([label])

  saliency=Saliency(model, model_modifier=r2l, clone=True)
  saliency_map=saliency(score, img, smooth_samples=20,
smooth_noise=0.2)
  saliency_map=normalize(saliency_map)

  grad_cam=Gradcam(model, model_modifier=r2l, clone=True)
  cam=grad_cam(score,img, penultimate_layer=-1)
  cam=normalize(cam)

  ax1=ax[i][0]
  ax2=ax[i][1]
  ax3=ax[i][2]

  ax1.imshow(img/255)
  ax1.set_title(f'Label:{classes_dict[label]}')

  ax2.imshow(saliency_map.reshape(240,320), cmap='Greys')
  ax2.set_title('Smooth Grad')

  heatmap = np.uint8(cm.jet(cam)[..., :3] * 255)
  ax3.imshow(img/255)
  ax3.imshow(heatmap.reshape(240,320,3), cmap='jet',
alpha=0.5)
  ax3.set_title('Grad Cam')

Figure .suptitle('Visualising Smooth Grad and GradCam for
Correctly predicted labels', fontsize=15)

plt.show()

from tf_keras_vis.activation_maximization import
ActivationMaximization
```

```
from tf_keras_vis.activation_maximization.callbacks import
Progress, PrintLogger
from tf_keras_vis.activation_maximization.input_modifiers
import Jitter, Rotate2D, Scale
from tf_keras_vis.activation_maximization.regularizers import
Norm, TotalVariation2D

seed_input = tf.random.uniform((10, 240, 320, 3), 0, 255)

scores=CategoricalScore(list(range(0,10)))

activation_maximization = ActivationMaximization(model,

model_modifier=r2l,
                                                    clone=True)
activations = activation_maximization(scores,
                                      seed_input=seed_input,
                                      callbacks=[Progress()])



Figure  = plt.figure(figsize=(15, 18))

for i, label in enumerate(classes_dict.values()):
    ax = plt.subplot(4, 3, i + 1)

    # Convert TensorFlow tensor to a NumPy array
    activation = activations[i].numpy()

    # Normalize the activation data to [0, 1]
    activation_min = activation.min()
    activation_max = activation.max()
    activation_normalized = (activation - activation_min) /
(activation_max - activation_min)
```

```
    ax.imshow(activation_normalized)
    ax.set_title(f'{label}', fontsize=16)
    ax.axis('off')

Figure .suptitle('Visualising the dense layer using activation
maximization', fontsize=15)
plt.tight_layout()
plt.show()
```

- **Part 5 – Error Analysis**

```
incorrect_imgs = np.zeros((0, 240, 320, 3))
incorrect_og_labels = np.zeros((0, 10))
incorrect_pred_labels = np.zeros((0, 10))

for _ in range(40):
    # Use next() function to get the next batch
    sample_batch = next(val_gen)
    imgs_array = sample_batch[0]
    labels_array = sample_batch[1]

    preds = model.predict(imgs_array)

    incorrect_idx = np.where(np.argmax(preds, axis=1) !=
np.argmax(labels_array, axis=1))[0]

    if incorrect_idx.shape != (0,):
        incorrect_imgs = np.concatenate((incorrect_imgs,
imgs_array[incorrect_idx]), axis=0)
        incorrect_og_labels =
np.concatenate((incorrect_og_labels,
labels_array[incorrect_idx]), axis=0)
        incorrect_pred_labels =
np.concatenate((incorrect_pred_labels, preds[incorrect_idx]),
axis=0)
```

```
random_idx_5=np.random.randint(0,incorrect_imgs.shape[0],
size=5)

r2l=ReplaceToLinear()

Figure ,ax=plt.subplots(ncols=3, nrows=5,figsize=(12,18))

for i,idx in enumerate(random_idx_5):

  img=incorrect_imgs[idx]
  label=np.argmax(incorrect_og_labels[idx])
  pred_label=np.argmax(incorrect_pred_labels[idx])

  score=CategoricalScore([pred_label])

  saliency=Saliency(model, model_modifier=r2l, clone=True)
  saliency_map=saliency(score, img, smooth_samples=20,
smooth_noise=0.2)
  saliency_map=normalize(saliency_map)

  grad_cam=Gradcam(model, model_modifier=r2l, clone=True)
  cam=grad_cam(score,img, penultimate_layer=-1)
  cam=normalize(cam)

  ax1=ax[i][0]
  ax2=ax[i][1]
  ax3=ax[i][2]

  ax1.imshow(img/255)
  ax1.set_title(f'Original label:{classes_dict[label]} \n
Predicted label:{classes_dict[pred_label]}')

  ax2.imshow(saliency_map.reshape(240,320), cmap='Greys')
```

```
  ax2.set_title('Smooth Grad')

  heatmap = np.uint8(cm.jet(cam)[..., :3] * 255)
  ax3.imshow(img/255)
  ax3.imshow(heatmap.reshape(240,320,3), cmap='jet',
alpha=0.5)
  ax3.set_title('Grad Cam')

Figure .suptitle('Visualising Smooth Grad and GradCam for
incorrectly predicted labels', fontsize=15)

plt.show()
```

- **Part 6 – Output Detection Using Haar cascades**

```
import cv2
import numpy as np
from keras.preprocessing.image import img_to_array
from keras.models import model_from_json

# Load emotion classification model
with open('state-farm-distracted-driver-detection/saved
models/model_EfficientNetB3.json', 'r') as json_file:
    loaded_model_json = json_file.read()

model = model_from_json(loaded_model_json)
model.load_weights("state-farm-distracted-driver-
detection/saved models/model_EfficientNetB3_.weights.h5")

# Load the face detector
face_haar_cascade =
cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
```

```
# Emotion labels
emotion_labels = [
    'safe driving', 'texting right', 'talking on the phone -
right', 'texting left',
    'talking on the phone - left', 'operating the radio',
'drinking',
    'reaching behind', 'hair and makeup', 'talking to
passenger'
]

# Start the webcam feed
cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()
    if not ret:
        break

    gray_image = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_haar_cascade.detectMultiScale(gray_image,
scaleFactor=1.3, minNeighbors=5)

    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0),
2)
        roi_gray = gray_image[y:y+h, x:x+w]
        # Resize the ROI to the expected input size of the
model (240x320)
        roi_rgb = cv2.resize(roi_gray, (320, 240))

        # Convert grayscale image to RGB
        roi_rgb = cv2.cvtColor(roi_rgb, cv2.COLOR_GRAY2RGB)

        roi_rgb = roi_rgb.astype("float") / 255.0
        roi_rgb = img_to_array(roi_rgb)
```

```
        roi_rgb = np.expand_dims(roi_rgb, axis=0)

        preds = model.predict(roi_rgb)[0]
        label = emotion_labels[np.argmax(preds)]
        confidence = np.max(preds)

        # Display label and confidence on the bounding box
        label_text = f"{label} {confidence:.2f}"
        cv2.putText(frame, label_text, (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0), 2)

    cv2.imshow('Driver Behavior Detection', frame)

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

cap.release()
cv2.destroyAllWindows()
```

- **Part 7 Output Detection Using Yolo & Haar cascades**

```
! pip install ultralytics
import torch
import cv2
import numpy as np
from keras.preprocessing.image import img_to_array
from keras.models import model_from_json
from ultralytics import YOLO

# Load YOLOv8 model (make sure 'yolov8n.pt' is in your
directory or provide full path)
yolo_model = YOLO('yolov8n.pt')

# Load emotion classification model
```

```
with open('state-farm-distracted-driver-detection/saved
models/model_EfficientNetB3.json', 'r') as json_file:
    loaded_model_json = json_file.read()

model = model_from_json(loaded_model_json)
model.load_weights("state-farm-distracted-driver-
detection/saved models/model_EfficientNetB3_.weights.h5")

# Emotion labels
emotion_labels = [
    'safe driving', 'texting right', 'talking on the phone -
right', 'texting left',
    'talking on the phone - left', 'operating the radio',
'drinking',
    'reaching behind', 'hair and makeup', 'talking to
passenger'
]

# Start the webcam feed
cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()
    if not ret:
        break

    # Use YOLOv8 to detect objects (faces)
    results = yolo_model.predict(frame)

    # Loop through results (detections) and draw bounding
boxes
    for result in results:
        boxes = result.boxes.xyxy.cpu().numpy()  # Extract
bounding boxes
        confidences = result.boxes.conf.cpu().numpy()  #
```

```
Confidence scores
        class_ids = result.boxes.cls.cpu().numpy()  # Class
IDs

        for i, box in enumerate(boxes):
            x1, y1, x2, y2 = map(int, box[:4])  # Extract the
box coordinates
            confidence = confidences[i]
            class_id = int(class_ids[i])

            # Only process faces (assuming you are detecting
faces; adjust class_id as needed)
            # If you want to restrict to a specific class,
e.g., 'person', check `class_id`

            # Draw a rectangle around the detected face
            cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 0,
0), 2)

            # Extract and preprocess the face region for
emotion classification
            roi_rgb = frame[y1:y2, x1:x2]  # Crop the face
region
            if roi_rgb.size == 0:  # Skip invalid regions
                continue
            roi_rgb = cv2.resize(roi_rgb, ( 320 , 240))  #
Resize for emotion classification model
            roi_rgb = roi_rgb.astype("float") / 255.0  #
Normalize the pixel values
            roi_rgb = img_to_array(roi_rgb)
            roi_rgb = np.expand_dims(roi_rgb, axis=0)

            # Predict emotion
            preds = model.predict(roi_rgb)[0]
            label = emotion_labels[np.argmax(preds)]
```

```
            # Display the label and confidence score
            label_text = f"{label} {confidence:.2f}"
            cv2.putText(frame, label_text, (x1, y1 - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0), 2)

    # Display the frame with detections
    cv2.imshow('Emotion Detection', frame)

    # Break the loop if 'q' is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```