Analyzing the Efficacy of Convolutional Neural Network Architectures for real-time Driver Behavior Detection: An In-depth Study of VGG16, ResNet50, InceptionV3, and Xception

By
ARAFAT SAHIN AFRIDI
ID: 211-15-3971

MD ARAFATH KAFY
ID: 211-15-3946

FINAL YEAR DESIGN PROJECT REPORT

This Report Presented in Partial Fulfillment of the Requirements for the **Degree of Bachelor of Science in Computer Science and Engineering**

Supervised by

MS. NAZMUN NESSA MOON
Associate Professor
Department of Computer Science and Engineering Daffodil International University

Co-Supervised by

MD. SHAHRIAR SHAKIL
Lecturer
Department of Computer Science and Engineering Daffodil International University

DAFFODIL INTERNATIONAL UNIVERSITY
Dhaka, Bangladesh

January 12, 2025

APPROVAL

This Project titled **"Analyzing the Efficacy of Convolutional Neural Network Architectures for real-time Driver Behavior Detection: An In-depth Study of VGG16, ResNet50, InceptionV3, and Xception"**, submitted by **Arafat Sahin Afridi, 211-15-3971** and **Md. Arafath Kafy, 211-15-3946** to the Department of Computer Science and Engineering, Daffodil International University has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Engineering and approved as to its style and contents. The presentation has been held on 12 January, 2025.

BOARD OF EXAMINERS

Dr. Md. Fokhray Hossain (MFH) — **Chairman**
Professor
Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University

Md. Shah Md. Tanvir Siddiquee (SMTS) — **Internal Examiner**
Assistant Professor
Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University

Mr. Md Umaid Hasan (MUH) — **Internal Examiner**
Senior Lecturer
Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University

Nazibur Rahman — **External Examiner**
Technical Lead – Database Administrator
Telenor - Grameen Phone Account

©Daffodil International University

DECLARATION

We hereby declare that this project has been done by us under the supervision of **Ms. Nazmun Nessa Moon, Associate Professor**, Department of Computer Science and Engineering, Daffodil International University. We also declare that neither this project nor any part of this project has been submitted elsewhere for the award of any degree or diploma.

Supervised by:

Nazmun Nessa Moon
Associate Professor
Department of Computer Science and Engineering Daffodil International University

Co-Supervised by:

MD. Shahriar Shakil
Lecturer
Department of Computer Science and Engineering Daffodil International University

Submitted by:

Arafat Sahin Afridi
Student ID: 211-15-3971
Department of Computer Science and Engineering Daffodil International University

Md. Arafath Kafy
Student ID: 211-15-3946
Department of Computer Science and Engineering Daffodil University

©Daffodil International University ii

ACKNOWLEDGEMENTS

This work would not have been possible without the support and contributions of many individuals over the past two semesters. We are deeply grateful to everyone who has assisted us in one way or another.

First, we express our heartfelt thanks and gratefulness to the almighty for His divine blessing making it possible for us to complete the **Final Year Design Project (FYDP)** successfully.

We are grateful and wish our profound indebtedness to **Supervisor Ms. Nazmun Nessa Moon**, **Associate Professor**, Department of Computer Science and Engineering, Daffodil International University, Dhaka, Bangladesh. Deep knowledge and keen interest of our supervisor in the field of **Artificial Intelligence and Machine Learning** carry out this project. Her endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts, and correcting them at all stages have made it possible to complete this project.

We would like to express our heartfelt gratitude to the Head of the Department of Computer Science and Engineering, for his kind help in finishing our project and also to other faculty members and the staff of the Department of Computer Science and Engineering, Daffodil International University.

We would like to thank our entire course-mates at Daffodil International University, who took part in this discussion while completing the coursework.

Finally, we must acknowledge with due respect the constant support and patience of our parents.

ABSTRACT

Distracted driving is a significant contributor to traffic accidents worldwide, necessitating effective driver behavior monitoring systems. This study investigates the efficacy of state-of-the-art Convolutional Neural Network (CNN) architectures—VGG16, ResNet50, InceptionV3, Xception, and Xception_Customize_Model—for real-time driver behavior detection. The research employs a systematic methodology to evaluate these architectures on parameters such as accuracy, computational efficiency, and generalization ability. Through data preprocessing techniques like image augmentation and normalization, models were trained to classify driver behaviors such as safe driving, texting, talking on the phone, turning, and others. The experimental results highlight that Xception_Customize_Model achieved the highest validation accuracy of 97.07% with a consistently low validation loss, demonstrating superior performance and stability. The Xception model followed closely with competitive accuracy but exhibited occasional fluctuations in validation performance. ResNet50 also performed well, with a validation accuracy of 88.25%, reflecting robust classification capabilities despite requiring more epochs for stabilization. In contrast, InceptionV3, VGG16, and CNN demonstrated lower performance, with validation accuracies not exceeding 70%, largely due to higher training losses and limited generalization ability. The findings of this study contribute to the development of intelligent transportation systems by enhancing real-time detection of driver distractions, thereby promoting road safety. This research provides valuable insights into optimizing CNN architectures for real-world applications, offering a pathway for practitioners to implement scalable, efficient, and accurate driver monitoring solutions.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Road accidents are a major cause of injury and death around the world, and Bangladesh is no exception. Due to the different driving conditions and increasing number of vehicles on the road, there is an increase in road accidents in country. Many of these accidents are caused by driver-related factors such as fatigue, distraction and reckless driving. So, driver behavior plays an important role in road safety. Distracted driving has become one of the leading cause of accidents globally. Drivers are often engaged in activities such as texting, talking on the phone, and falling asleep while driving. These behaviors divert a driver's attention from the road, leading to potentially catastrophic outcomes. To reduce risks and enhance road safety, these behaviors must be recognized and addressed. Automated driver behavior detection with in car cameras has become possible thanks to recent developments in machine learning and image processing.

This study focuses on leveraging machine learning techniques to classify driver behaviors intro categories such as Safe Driving, Turning, Texting Phone, Talking Phone and Others. The main focus was to contribute to the development of real-time monitoring systems that can identify potentially dangerous actions and alert the driver relevant authorities. Using image-based analysis to detect driver behavior usually entails a number of steps in a workflow sequence, such as image acquisition, preprocessing, feature extraction and classification. To guarantee precise behavior classification. To guarantee precise behavior classification from in car footage. The first step is collecting and annotating image data that captures various activities. After that, these photos which are frequently in JPG or PNG formats are preprocessed to improve quality and remove noise. Machine learning models such as CNN architectures (e.g., VGG16, ResNet50, InceptionV3, Xception) have been explore to identify and quantify driver behaviors. A dataset of driver behavior is used to validate these classification models, ensuring that the results can be applied to actual situations. In order to

improve road safety, this research attempts to increase the accuracy and efficiency of driver behavior detection systems by combining advanced algorithms for machine learning with reliable image processing techniques.

1.2 Motivation

The increasing number of road accidents, driven by the rapid rise in vehicle usage and the prevalence of driver distractions, highlights the urgent need for effective driver monitoring systems. Distracted behaviors like texting, talking on the phone or doing other things while driving are the causes of a lot of accidents. Existing solutions often rely on limited datasets or lack the robustness to handle real-time scenarios effectively. By utilizing cutting-edge CNN architectures like VGG16, ResNet50, InceptionV3, and Xception, this project, "Analyzing the Efficacy of Convolutional Neural Network Architectures for Real-Time Driver Behavior Detection," seeks to address these issues. This study aims to create an effective, real-time driver behavior detection system that can recognize a variety of risky behaviors by evaluating and contrasting these models' performance on various datasets. The ultimate objective is to improve road safety by lowering driver errors, delivering timely warnings, and lessening the negative social and financial effects of traffic accidents.

1.3 Objectives

This research's main goal is to use state-of-the-art deep learning architectures to develop a scalable and efficient driver behavior detection system. By using a real-time dataset of real drivers, this study seeks to enhance the real-time detection of risky driving behaviors such as distraction, drowsiness and reckless driving. The research also seeks to balance high detection accuracy and computational efficiency to enhance road safety and prepare the system for real-world applications. The following is a summary of the study's objectives:

- To improve road safety by offering a trustworthy system for classifying driver behavior.
- To develop a strong framework that can use image processing and machine learning methods to analyze and classify driver behaviors.
- To gain knowledge about real-time applications of AI and ML in driver monitoring systems.
- To investigate cutting-edge CNN architectures for better performance and classification accuracy.
- To increase the overall effectiveness and precision of identifying risky driving practices.

To advance driver behavior analysis as a means of advancing the field of intelligent transportation systems.

1.4 Methodology

The goal of this research is to use cutting-edge Convolutional Neural Network (CNN) architectures, such as VGG16, ResNet50, InceptionV3, and Xception, to create a reliable system for identifying driver behaviors in real-time applications. Three main elements are highlighted by the methodology: the creation of a web application interface for real-time feedback, model training and evaluation, and data collection and preprocessing.

Five different driver behaviors—safe driving, talking on the phone, texting, turning, and other distractions—are captured in the dataset, which was gathered under actual driving circumstances. For CNN model training, thorough preprocessing methods like image resizing, normalization, and augmentation guarantee high-quality input. Using methods like transfer learning and hyperparameter tuning, these models are adjusted and optimized to strike a balance between computational efficiency and accuracy. To verify model performance and guarantee robustness, evaluation metrics like accuracy, precision, and recall are used.

Additionally, the system has an easy-to-use web application that lets users upload data or images, examine driver behavior, and get real-time feedback with alerts and useful insights. Thorough project management guarantees the system's scalability and preparedness for real-world application. This includes financial planning, testing, and hardware and software specifications. In the end, this study provides a thorough and effective method for identifying risky driving practices and enhancing traffic safety.

1.5 Project Outcome

Based on the results of this investigation, the real-world implications of driver behavior detection using different classification techniques are explored. The goal of this research is to use deep learning models to develop a real-time, accurate and efficient system for detecting driver behavior. Following are the expected results of this study:

- Driver Behavior Detection: Use CNN-based architectures to precisely identify whether a driver is acting safely or distractedly.

- Progress in the Knowledge of ML/DL for Driver Monitoring: Learn more about the use of deep learning (DL) and machine learning (ML) models for monitoring and detecting driver behavior in real time.

- Behavior Categorization: Make it easier to classify distracted driving practices like drinking, texting, and reaching behind you. Comparing This Study to Others: To assess the performance gains of the chosen architectures, compare the results of this study with those of previous

research using older datasets.

- Model Optimization: Using the provided dataset, determine which deep learning architecture—of VGG16, ResNet50, InceptionV3, and Xception—is best suited for identifying driver behaviors. Also trying to develop a more efficient custom model.

1.6 Organization of the Report

The report is structured into seven chapters, each focusing on different aspects of the study "Analyzing the Efficacy of Convolutional Neural Network Architectures for real-time Driver Behavior Detection: An In-depth Study of VGG16, ResNet50, InceptionV3, and Xception".

Chapter 1, the introduction provides a comprehensive overview of the study, setting the stage for the subsequent chapters. It includes sections on the background and present state of driver behavior detection, the problem statement, the objectives of the study, the scope and limitations, the organization of the report, and a summary.

Chapter 2, literature review is based on existing literature related to driver behavior detection and classification. It begins with an overview, followed by a detailed examination of related works. It also highlights remaining problems in the field by comparing previous works. Key findings from the literature are summarized at the end of the chapter.

The conceptualization of this research effort is explained in the Chapter 3. To continue to address the theoretical aspect of the study, this chapter provides further detail on the statistical approaches that were employed. This chapter also demonstrates how to apply procedural ways to ml approaches.

Chapter 4 contains the test results, performance evaluation, and result explanation. A few test pictures are included in this chapter to aid in the project's realization. The use of ml algorithms is followed by an inquiry of the results after this chapter.

Chapters 5 includes challenges were faced during the project work.

Chapter 2

Background

2.1 Introduction

This chapter reviews the literature on Driver behavior detection, highlighting the significance of knowledge and technological advancements in these areas. The chapter then delves into various research efforts aimed at improving behavior detection techniques. Significant contributions include the integration of Deep Learning (DL) and Computer vision in addressing driver related factors contributed to road accidents, hybrid models combining Convolutional Neural Networks (CNNs) with other machine learning methods, are also discussed as innovative solutions to improve detection accuracy and efficiency. Each study is evaluated base on the dataset size, architecture of the proposed model, classification accuracy, computational efficiency, limitations, and contributions. The literature review highlights research gaps in existing methodologies and the societal need for more robust and scalable driver behavior detection system, focusing on the evaluation and comparison of VGG16, ResNet50, InceptionV3 and Xception for real time driver behavior detection to achieve better performance.

2.2 Literature Review

In a study a CNN-based fusion model was proposed for driving behavior primitive classification. Initially, using statistical methods primitive features were constructed to solve the issue of inconsistent durations among primitives. As the classifier input, these features were rebuilt as matrices. Second, a novel fusion technique was used to fuse the 1D-CNN and 2D-CNN simultaneously. The input data's global and local features, which comprehensively describe the different relationships between multiple types of variables, could be analyzed by this model concurrently. Additionally, two fusion models were produced by setting the fusion stages both before and after FC. Labels of driving behavior primitives were effectively identified based on the two fusion models, and the outcomes were compared with the outcomes of basic models [1]. In order to

improve the predictability of the results, the Synthetic Minority Oversampling Technique (SMOTE) was utilized to highlight the significance of data balancing in machine learning. To further guarantee the model's accuracy and consistency across the original features and the suggested RKnD probabilistic features of the data sets, k-fold cross-validation is employed. Showed how smartphone-based systems have the potential to greatly increase road safety by achieving such high accuracy. In order to detect driver behaviors, a novel method that uses smartphone motion sensor data was presented to achieve an impressive accuracy rate of 99.63% which was notable for applying machine learning methods in a useful and approachable way [2]. A survey was conducted to improve car safety and stop reckless driving-related accidents. A comprehensive overview of neural network-based research approaches was provided for analyzing these driver bio-metrics, along with a detailed analysis of their benefits and limitations. Two pertinent datasets that each categorize ten distinct in-cabinet behaviors independently are included in the evaluation, offering a methodical classification for the detection of driver behaviors. The ultimate goal was to provide guidance for the creation of systems that monitor driver behavior. The structure of the paper includes sections on driver behavior analysis techniques, neural networks, autonomous vehicles, dataset utilization, final findings, and recommendations for future research, making it understandable to a wide range of audiences [3]. In order to improve road safety overall, an efficient technique for the real-time detection of abnormal driving behavior was developed. Which also supports driving risk assessment and driving behavior improvement [4]. A CMO algorithm was introduced for ADAS anomaly detection based on AdaBoost MSVM. Here, power data from electric vehicles is used to validate the efficiency of the suggested method. The suggested CMO algorithm, which is based on AdaBoost MSVM, predicts safe and unusual driving behaviors with accuracy. By employing these methods, safe driving is ensured and the training speed of driving vehicles is increased [5]. In identifying unusual driving behavior, the 1D-DenseNet model's performance was compared to the original DenseNet model using the Kaggle distracted driver behavior dataset. The 1D-DenseNet model performs better than the original DenseNet model in terms of overhead, loss, and classification and validation accuracies, according to the results. Following 100 training epochs with Keras on top of TensorFlow, the 1D-DenseNet attained 99.80% classification accuracy and 99.96% validation accuracy, with a categorical cross-entropy loss of 0.19 on the validation set. These results show how well the 1DDenseNet model performs when it comes to enhancing the identification of unusual driving behavior [6]. In order to categorize driver behavior and its performance, various study types, datasets, data sources, features, preprocessing methods, and artificial intelligence algorithms were highlighted and examined. The major objective was to determine the main contributions and difficulties of researching driver behavior classification and to suggest possible paths for future research and practice for practitioners and researchers, based on the findings from the analysis of the chosen works [7]. LR-

RFC approach was proposed which achieves the highest performance score. It was shown through experiments that the random forest, employing the suggested LR-RFC method, achieved the highest performance score of 99%. Hyperparameter optimization and k-fold cross-validation were used to validate the performance [8]. The precision and usefulness of the detection model for public sensors was enhanced with the introduction of a dependable DBD system built on Graph Convolutional Long Short-Term Memory networks. Public sensors were also used to evaluate the efficacy of the model. The suggested model demonstrated its ability to generate reliable and accurate results for both scenarios by achieving an accuracy of 97.5% for public sensors and an average accuracy of 98.1% for non-public sensors [9]. Proposed a hybrid CNN framework (HCF) to identify behaviors of distracted drivers. Three cooperative pretrained CNN models were used to extract all features at various scales. The features were then concatenated to create feature maps. The fully connected layer was then trained to categorize every instance of distracted driving. To stop the training model from overfitting to the training data, dropout technology was used during the training process. CAM was also used to draw attention to the detection area findings. The suggested HCF performed exceptionally well, achieving 96.74% accuracy in identifying distracted driving behaviors [10]. To improve the classification accuracy, a temporal voting system based on historical inference instances was introduced. The suggested InterCNN with MobileNet convolutional blocks can classify nine distinct behaviors with 73.97% accuracy and five "aggregated" behaviors with 81.66% accuracy, according to an experiment on their own collected dataset in a mock-up car environment [11]. Proposed a system that uses architecture to identify a driver's level of drowsiness. Four deep learning models—AlexNet, VGG-FaceNet, FlowImageNet, and ResNet—that use RGB videos of drivers as input and aid in drowsiness detection make up the suggested architecture. Gained an accuracy of 85% [12]. Proposed an ADNet for driver behavior recognition that incorporates the attention mechanism into the DNN model. The ADNet approach outperforms the state-of-the-art techniques with a Top-1 accuracy of 98.48% [13]. CNN-based models were created to identify and notify drivers of mistakes. On a dataset of 22,424 images in 10 error classes, image features were extracted and classified using k-NN, SVM, and RF algorithms using transfer learning with the SqueezeNet architecture. With k-NN producing the best results, classification accuracies were 98.1% for k-NN, 95.8% for SVM, and 88.7% for RF. To determine the best model for real-time driver error detection, additional metrics and training/testing durations were examined [14]. suggested a way to use vehicle sensor data from steering and pedal pressure to identify drowsy driving. A specific ensemble network model based on convolutional neural networks (CNNs) was created to identify the long- and short-duration types of drowsy driving. In order to address data imbalance, the model modifies the ratio of normal to sleepy driving data and uses time series analysis for feature fusion. A detection accuracy of up to 94.2% was attained using a dataset of 198.3 hours

from simulated driving across various road types [15]. The precision and usefulness of the detection model for public sensors was enhanced with the introduction of a dependable DBD system built on Graph Convolutional Long Short-Term Memory networks. Public sensors were also used to evaluate the efficacy of the model. The suggested model demonstrated its ability to generate reliable and accurate results for both scenarios by achieving an accuracy of 97.5% for public sensors and an average accuracy of 98.1% for non-public sensors [16]. Proposed a hybrid CNN framework (HCF) to identify behaviors of distracted drivers. Three cooperative pretrained CNN models were used to extract all features at various scales. The features were then concatenated to create feature maps. The fully connected layer was then trained to categorize every instance of distracted driving. To stop the training model from overfitting to the training data, dropout technology was used during the training process. CAM was also used to draw attention to the detection area findings. The suggested HCF performed exceptionally well, achieving 96.74% accuracy in identifying distracted driving behaviors [17].

Table 2.1: Summary of Literature Reviewed.

Author (s)	**Year**	**Title**	**Methodology**	**Key Findings**
Xiaotong Cui	2024	Driving behavior primitive classification using CNN-based fusion models	Used Fusion of 1D-CNN and 2D-CNN got 93.47% accuracy.	Presented a novel CNN-based fusion model that efficiently analyzes both local and global features of input data by combining 1D-CNN and 2D-CNN in parallel.
Mohammad Shariful Islam	2024	Elevating Driver Behavior Understanding with RKnD: A Novel Probabilistic Feature Engineering Approach	Combination of Random Forest, K-nearest classifier, Decision tree was applied where the result of accuracy was 99.63%.	Highlighted the value of engineered data in improving model performance and demonstrated how RKnD feature engineering is superior to original features.
Fangming Qu	2024	Comprehensive study of driver behavior monitoring systems using computer vision and machine learning techniques.	AI system with Deep Neural Networks (DNNs).	Proposed a DNN-based method for computer vision-based driver behavior classification.

Yongfeng Ma	2023	Real-time detection of abnormal driving behavior based on long short-term memory network and regression residuals	Applied LSTM-Residual (LSTM-R) and gained 86.6% accuracy	Proposed LSTM-R for real-time abnormal driving detection, outperforming five other algorithms in tests.
Ravikumar Sethuraman	2023	An optimized AdaBoost Multi-class support vector machine for driver behavior monitoring in the advanced driver assistance systems	AdaBoost Multi-class SVM with Cat Mouse Optimizer (CMO) brought 91.45% accuracy.	Proposed an optimized AdaBoost MSVM with CMO algorithm for accurate ADAS anomaly detection, achieving superior metrics.
Aisha Ayad	2023	Detecting Abnormal Driving Behavior Using Modified DenseNet	Compared 1D-Densenet model with DenseNet, VGG, ResNet, Inception to get the best accuracy 99.80%.	Developed 1D-DenseNet with adapted dense blocks and preprocessing techniques, improving accuracy and speed.
Soukaina Bouhsissin	2023	Driver Behavior Classification: A Systematic Literature Review	LSTM, LLDA, RF, SVM, GRU were applied and the most performing model showed 98.5% accuracy.	Demonstrated SVM's ability to handle naturalistic driving data for driver behavior classification.
Ali Raza	2023	Preventing Road Accidents Through Early Detection of Driver Behavior Using Smartphone Motion Sensor Data: An Ensemble Feature Engineering Approach	RF with LR-RFC (Feature Engineering) gave 99% accuracy.	Introduced LR-RFC feature engineering, which greatly increased the accuracy of the RF model and was verified by hyperparameter tuning and k-fold cross-validation.
Khadija Kanwal	(2023)	Smartphone Inertial Measurement Unit Data Features for	RF and XGBoost was used in this study.	Showed how crucial the "timestamp" feature is to reaching 100% accuracy; examined the impact

		Analyzing Driver Driving Behavior		of feature selection and classification strategies; and emphasized machine learning over deep learning.
Chung-Hong Lee	2023	A Privacy-Preserving Learning Method for Analyzing HEV Driver's Driving Behaviors	By implementing BiGRU 500 & 1000 epochs 97.5% accuracy was gained.	Introduced the BiGRU model, which demonstrated the benefits of deep learning for detecting aggressive or safe driving behavior. Its kappa score was 0.965 and its accuracy was 97.5 percent.
Chaoyn Zhang	2020	Driver Behavior Recognition via Interwoven Deep Convolutional Neural Nets with Multi-Stream Inputs	Interwoven CNN (InterCNN) provided 73.97% for 9 classes and 81.66% for 5 classes.	Real-time inferences with a 15 ms latency per instance were accomplished by developing the InterCNN architecture, which successfully fuses multi-stream video data (front and side views with optical flow); temporal voting was added for robust classification.
Shafeeq Kanaan Shakir Al-Doori	2021	Distracted Driving Detection with Machine Learning Methods by CNN Based Feature Extraction	KNN, SVM, Random Forest was applied in this study and the best accuracy gained was 98.1%.	Models showed promise for integration into real-time driver monitoring systems after demonstrating excellent performance in identifying driver errors from image data.
Yongsu Jeon	2021	Ensemble CNN to Detect Drowsy Driving with In-Vehicle Sensor Data	Proposed an ensemble CNN model with specialized subnetworks for short-duration (SD) and long-duration (LD) drowsy driving detection. Compared with	Achieved 94.20% accuracy and 94.18% F1-score using L=10,S=10L=10, S=10L=10,S=10. Performance increased with subnetworks but plateaued after L=6,S=6L=6,

			DNN, 1D-CNN, LSTM, and random forest models.	S=6L=6,S=6. Outperformed traditional methods and prior CNN-LSTM approaches.

2.2.1 Related Research

By reviewing other studies, it was found that different researchers proposed different approaches with different type of outcomes. Some of those research dataset was collected from online platform and some were collected by them. They brought high performance by detecting those behaviors more accurately using their preferred methods or algorithms. There was variation in dataset contains. Some contained more variety of classes in their dataset some contain less variety but all of them made a great contribution in driver behavior detection.

2.3 Gap Analysis

Impressive accuracy and validation scores have been demonstrated by previous research on driver behavior detection using machine learning (ML) and deep learning (DL) models. However, because of a number of significant limitations, these results frequently lack generalizability and robustness. The lack of diversity and variation in the datasets used is a major problem. The complexity of real-world driving situations, such as shifting lighting, different camera placements, and the varied behaviors of drivers across various demographics, is not sufficiently captured by the controlled datasets used in many studies. These limitations limit how well the models can generalize in practical settings. Many studies ignore a wider range of behaviors necessary for thorough driver monitoring in favor of concentrating on a small number of behavior classes, such as drinking or texting. Concerns regarding practical reliability are also raised by the use of artificially generated data, which is helpful for testing but falls short in capturing the complexity of real-world driving situations. A further significant drawback is the disregard for real-time applicability. Few models address the requirement for effective real-time operation, which is essential for prompt warnings or interventions in driver monitoring systems, even though many models perform well in offline analyses.

2.4 Summary

The literature review highlights advancements in deep learning models for driver behavior detection but notes challenges such as limited datasets, real-time performance issues, and model overfitting. Future research should focus on developing diverse, robust datasets, exploring novel CNN architectures, and improving computational efficiency for seamless real-time applications.

Chapter 3

Research Methodology

3.1 Methodology

The goal of this study is to create a system that can accurately and efficiently detect driver behavior in real-time applications. Analyzing how well advanced Convolutional Neural Network (CNN) architectures—VGG16, ResNet50, InceptionV3, and Xception—perform on a variety of driver behavior datasets is the main goal. In order to improve road safety and prevent accidents, this study intends to develop a strong model that can accurately identify a variety of driver distractions by utilizing state-of-the-art computer vision techniques.

3.1.1 Overview

This chapter offers a thorough overview of the methodology, requirement analysis, and design specifications for the project. It describes the proposed approach, covering system design, hardware and software requirements, project management strategies, and financial analysis. The section also presents a step-by-step methodology to be followed during project execution, ensuring a structured framework for successful delivery. Furthermore, it emphasizes the specific tools, technologies, and resources required for the project, along with a clear plan for managing time, budget, and resources.

3.1.2 Proposed Methodology

The proposed system combines Convolutional Neural Network (CNN) techniques with web application development to provide an effective and scalable solution for detecting driver behavior in real-time. It is organized into three main components:

1. **Data Acquisition and Preprocessing:** This phase emphasizes gathering high-quality data that captures a range of driver behaviors, such as safe driving, distracted driving (like texting and phone usage), turning, and

other driving actions. The collected data undergoes careful preprocessing to improve its quality, utilizing techniques like noise reduction, normalization, and augmentation to ensure the input is well-suited for training and achieves optimal model performance.

2. **Behavior Detection Model Using CNNs:** This component utilizes advanced CNN architectures like VGG16, ResNet50, InceptionV3, and Xception to train strong models that can effectively classify and detect various driver behaviors. The models are fine-tuned for real-time detection, focusing on both accuracy and computational efficiency, making them scalable and practical for use in real-world situations.

3. **Web Application Interface:** The system includes a user-friendly web application that enables users—such as drivers, fleet managers, and traffic authorities—to upload and analyze data in real-time. The platform provides immediate feedback on driver behavior, offering alerts and recommendations for interventions. Additionally, the web interface supports data storage, visualization, and reporting, facilitating continuous monitoring and actionable insights for improving road safety.

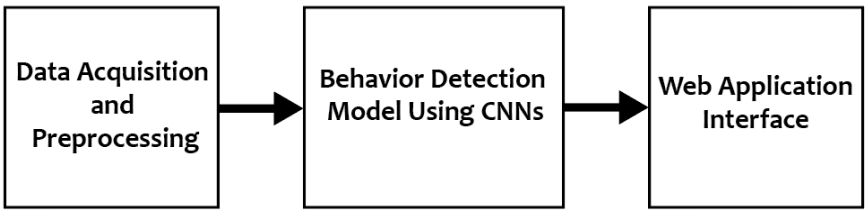

Figure 3.1: Work procedure diagram.

Together, these components form a powerful, comprehensive solution that not only enables the detection of unsafe driver behaviors but also empowers real-time interventions, contributing to enhanced road safety and more effective traffic management.

3.1.3 Data Flow Diagram Level 1

The data flow within the system follows a structured sequence to ensure efficient processing, model prediction, and user feedback:

- Image Input: The process begins when users upload images (or data files, depending on the system's input format) related to driver behavior, such as captured video frames or photos, through the web

application interface.

- Preprocessing: Once the data is uploaded, it undergoes a preprocessing phase. This step includes resizing images to a consistent dimension, normalizing pixel values, and performing data augmentation (e.g., rotation, flipping, scaling) to improve model robustness. This ensures that the input data aligns with the specific requirements of the Convolutional Neural Network (CNN) model being used, such as VGG-19, ResNet50, or InceptionV3.

- Model Prediction: After preprocessing, the images are fed into the trained CNN model. The model processes the data and classifies the images based on predefined categories (e.g., distracted driving, safe driving, texting, talking on the phone). The model's output is typically a probability distribution across the potential classes, from which the most likely class is selected as the predicted driver behavior.

- Result Display: The system then presents the prediction results on the user interface. The predictions, such as the specific driver behavior or detected anomaly, are displayed along with actionable feedback, which may include recommended actions, alerts, or additional insights. This feedback enables users (drivers, fleet managers, or traffic authorities) to take timely corrective actions.

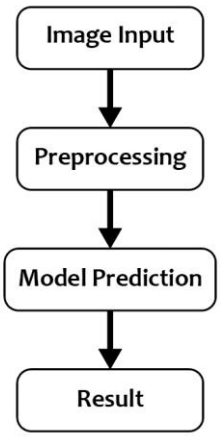

Figure 3.2: Data flow diagram

3.1.4 Implementation Steps

The system implementation is divided into five key phases, each ensuring that the end product is functional, efficient, and ready for real-world deployment.

1. **Data Collection and Preprocessing:**
   - **Data Collection:** A diverse and representative dataset is collected to

cover various driver behaviors under different conditions (e.g., lighting, angles, vehicle types). The dataset includes images, videos, or sensor data relevant to driver behavior (e.g., texting, talking, drowsiness, etc.).

- **Data Preprocessing:** The raw dataset undergoes several preprocessing steps, including image resizing, normalization (to standardize the range of input data), and data augmentation (to artificially expand the dataset). This ensures that the input data is suitable for training the machine learning models and helps mitigate overfitting.

2. **Model Training:**
   - **Training CNN Models:** In this phase, CNN models such as VGG-19, ResNet50, InceptionV3, or Xception are trained using the preprocessed dataset. The models learn to identify patterns and features specific to different driver behaviors. During training, hyperparameters such as learning rates, batch sizes, and optimization algorithms are carefully tuned to maximize performance.
   - **Model Optimization:** The models undergo fine-tuning, where they are adjusted to achieve the best trade-off between training accuracy and generalization to unseen data. This may involve using transfer learning (starting with pre-trained models) to accelerate the training process.

3. **Model Evaluation:**
   - **Performance Metrics:** Once the models are trained, they are evaluated using validation and test datasets that were not seen during training. Key performance metrics such as accuracy, precision, recall, F1 score, and confusion matrices are used to assess the model's performance.
   - **Cross-validation:** Cross-validation techniques are employed to ensure that the models generalize well across various subsets of the data. This step helps identify any overfitting and ensures the models' robustness in real-world conditions.

4. **Web Application Development:**
   - **User Interface Design:** The web application is developed to provide a seamless and intuitive interface for users (drivers, fleet managers, authorities). This interface enables users to upload images or sensor data and view the results of the model's predictions.
   - **Backend Development:** The backend system is developed using Flask (or other web frameworks), which connects the trained model with the user interface. The backend processes incoming requests, performs inference using the model, and sends the results back to the frontend.
   - **Real-time Feedback:** The application is designed to provide real-time feedback, enabling immediate alerts and suggestions for corrective actions based on the detected driver behavior.

5. **Integration and Testing:**

- **System Integration:** The trained machine learning model is integrated into the web application, ensuring that the system functions as a unified solution. This involves linking the backend logic (model inference) with the frontend (user interface) for smooth interaction.
- **Comprehensive Testing:** Extensive testing is performed across all components of the system, including functional testing (ensuring that all features work as expected), performance testing (evaluating how well the system handles different data sizes), and usability testing (ensuring that users can navigate the system effectively).
- **Validation and Optimization:** After integration, the system undergoes validation to ensure its accuracy in real-world scenarios. Optimization is also carried out to enhance the system's efficiency, particularly in terms of computation speed and real-time responsiveness.

This version provides more in-depth information about each phase, highlighting the technical details of the data processing, model training, and web application development, ensuring the system is fully optimized and ready for practical use.

3.2 Detailed Methodology and Design

3.2.1 Data Description

This dataset, published on Mendeley [18], is a primary and original collection of high-resolution images gathered and made publicly available by our team. The data was captured in Ashulia, Dhaka, Bangladesh, within both private vehicles and public buses, under real-world driving conditions. Using mobile phone cameras, we obtained a diverse and realistic set of visual data reflecting typical driving scenarios.

The dataset, categorized into five behavior classes, provides authentic, high-quality data, supporting research in computer vision, driver safety, and behavior analysis.

Table 3.1 Classes of the Data set

Class Name	Description	Visualization
Safe Driving	Driving with complete focus, no distractions are only labeled as safe driving. The driver maintains good posture and a firm grip on the steering wheel while paying close attention to the road. There are no outside distractions like using electronics, conversing with other passengers, or eating. Maximum safety is ensured by the driver's perfect, distraction-free driving style.	

Talking phone	The driver is using the phone in their hand or holding it to their ear while making a phone call. This conduct is a frequent contributor to accidents since it takes the driver's focus away from the road. Due to the cognitive and physical distractions it causes, talking on the phone while operating a motor vehicle is considered dangerous.	
Texting phone	While turning the car, the driver must give it their whole attention to make sure the turn is made safely. To perform the maneuver, the driver may be seen moving or modifying the steering wheel. In order to prevent collisions or swerving off the road, proper turning requires paying attention.	
Turning	While turning the car, the driver must give it their whole attention to make sure the turn is made safely. To perform the maneuver, the driver may be seen moving or modifying the steering wheel. In order to prevent collisions or swerving off the road, proper turning requires paying attention.	
Others	This category covers different kind of distracting behaviors that a driver may partake in, including talking to a passenger, eating snacks or drinking beverages, using earbuds to listen to music, sleeping while driving (a serious hazard), and turning away from the road. Each of these actions raises the chance of an accident by drastically lowering the driver's awareness and reaction time.	

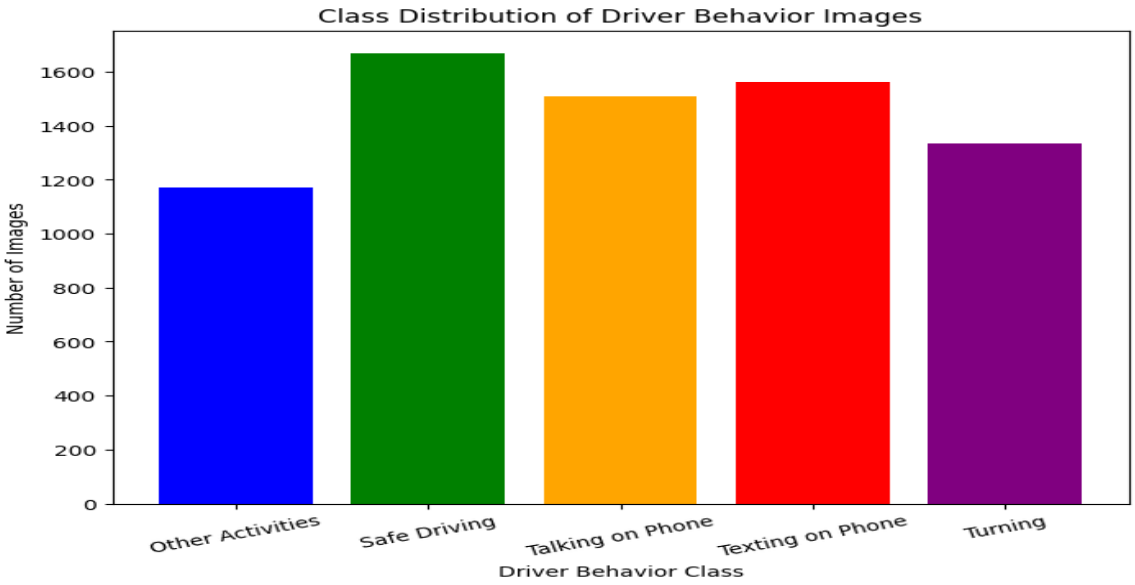

Figure 3.3: Class Distribution of Driver Behavior Images

The bar chart Figure 3.3 illustrates the distribution of images across five distinct driver behavior classes: Other Activities, Safe Driving, talking on the phone, texting on the phone, and turning. The dataset comprises a total of 7,286 images. With 1,679 photos, the Safe Driving class represents the largest portion of the dataset. The Turning class has 1,343 images, the Talking on Phone class has 1,513 images, and the Other Activities class contains 1,190 images. The smallest category, Texting on Phone, includes 1,161 images.

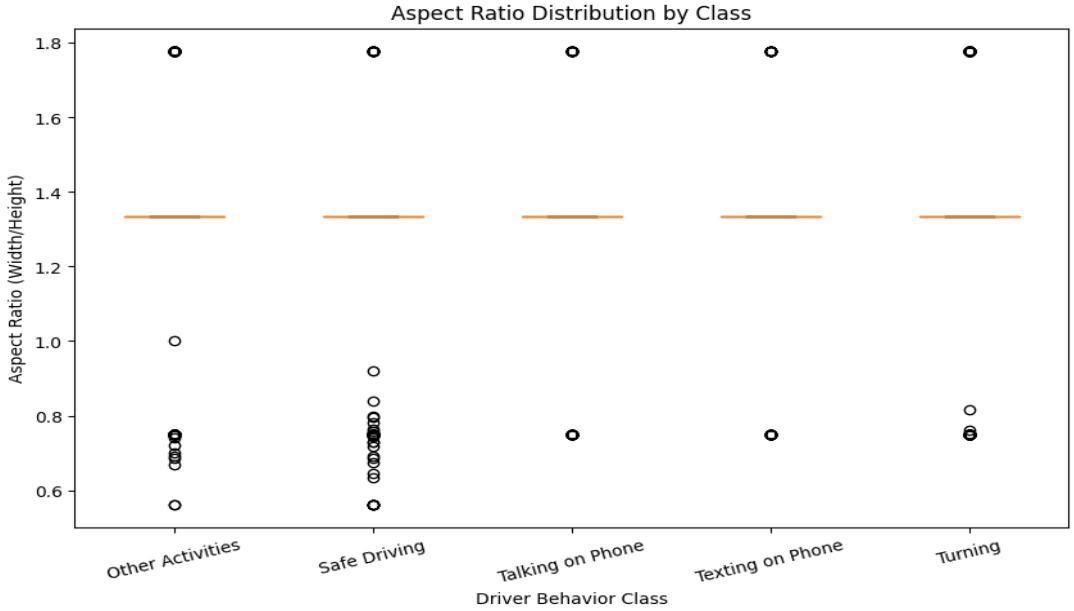

Figure 1.4: Aspect Ratio Distribution by Class

The chart Figure 1.4 presents the aspect ratio (width/height) distribution for images across five driver behavior classes: Other Activities, Safe Driving, talking on the phone, Texting on the phone, and Turning. The orange lines indicate the median aspect ratio for each class, which consistently centers around 1.4. Scatter

points below the median show variations, with most aspects of ratios clustering between 0.6 and 0.8. Outliers above 1.4 are sparsely distributed across classes. This suggests that while most images follow a standard aspect ratio, there are notable variations in certain cases.

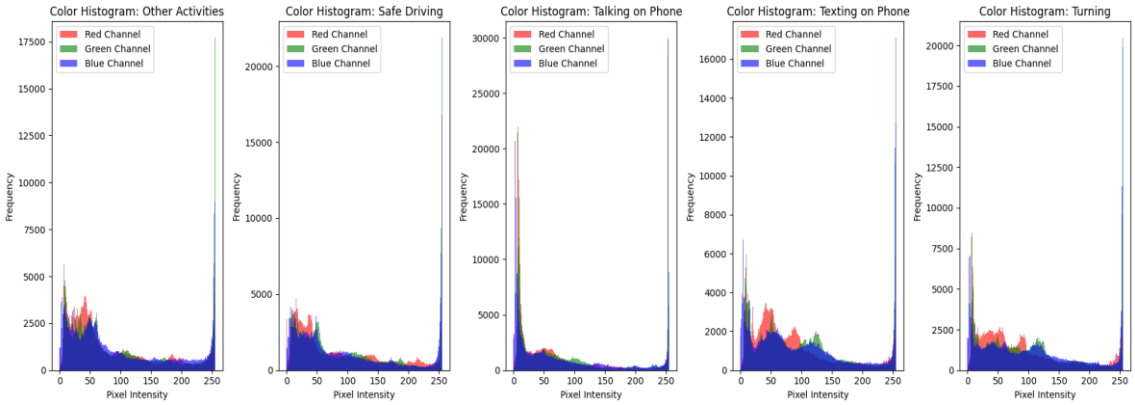

Figure 3.5: Color Histogram Analysis by Class

Figure 3.5 depicts the color histograms for the red, green, and blue channels across five driver behavior classes: Other Activities, Safe Driving, Talking on the Phone, Texting on the Phone, and Turning. The x-axis represents pixel intensity values (0–255), while the y-axis shows the frequency of pixel intensities. Each class exhibits similar patterns, with a prominent spike at higher intensity values, particularly in the green and blue channels. Lower intensities display wider distributions, indicating diverse color usage in the images. This consistency suggests similar color characteristics across the behavior classes, with minor variations.

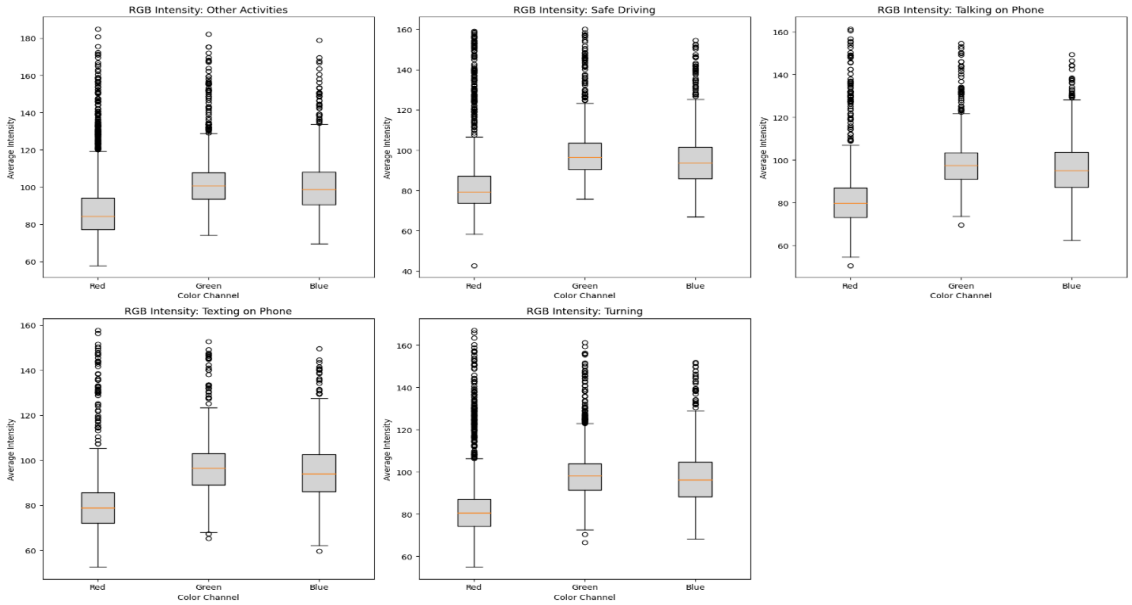

Figure 3.6: RGB Intensity Across Driver Activities

This Figure 3.6 illustrates the distribution of average RGB intensity values (Red,

Green, and blue channels) for five distinct driving behaviors: Other Activities, Safe Driving, Talking on the Phone, Texting on the Phone, and Turning. Each boxplot demonstrates variations in RGB intensities across the respective color channels for the different activities. Notable differences in intensity are observed among activities, with slight deviations in the green channel across behaviors. Outliers are marked, highlighting occasional high and low-intensity instances. This analysis helps assess visual patterns in driver behavior classification.

Figure 3.7: Categorization of Driver Behaviors Based on Interaction

This Figure 3.7 showcases a visual categorization of driver behaviors into four distinct groups: "Random images of people who talk with their phone," "Random images of people who turn around," "Random images of people who text with

their phone," and "Random images of people who drive safely."," "Random images of people who are in other positions." Each category comprises images depicting individuals engaged in the corresponding activities, illustrating their actions in real-world driving environments. The labels highlight unsafe practices such as phone use and inattentiveness versus safe driving behavior. This visual distinction provides a framework for analyzing the impact of driver distractions on safety outcomes.

3.2.2 Data Preparation

The dataset used for training the model consisted of images categorized into Five classes, each representing a different type of behavior or condition (or the relevant focus area). The images were preprocessed using data augmentation techniques, including rescaling, shearing, zooming, and horizontal flipping. These techniques were applied to enhance the model's robustness, improving its ability to generalize and perform effectively across different conditions and scenarios.

3.2.3 Image Preprocessing

We developed a deep learning-based framework to reliably achieve classification performance and effectively train the driver behavior recognition dataset. The structured validation process consists of five main steps: data preprocessing, dataset partitioning, model training, performance evaluation with a validation set, and testing with a separate test set. This process ensures that the model works well with current data and generates accurate, reliable results.

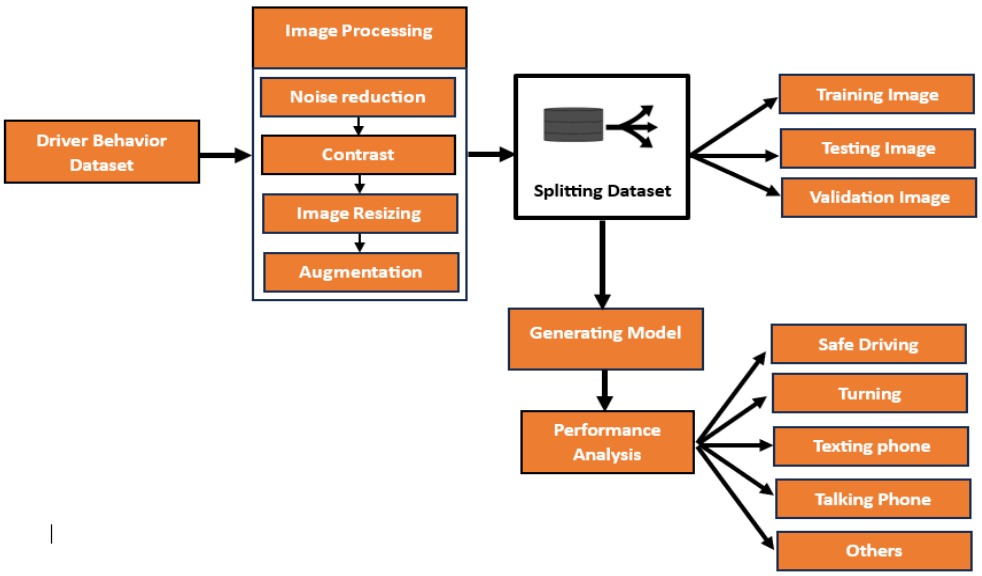

Figure 3.2 Image Preprocessing Flowchart

Data preprocessing is a crucial step to improve the quality of the dataset and make it more suitable for training deep learning models. The changes listed below have been made.

Noise Reduction: Impulse noise has been removed from the images using non-linear median filtering to ensure clarity and increase the signal-to-noise ratio.

Contrast Enhancement: Histogram equalization was utilized to adjust for irregular lighting and enhance contrast, producing continuously excellent images.

Image Resizing: To maintain uniformity and compatibility with the input requirements of deep learning models, all images were resized to a fixed dimension.

Data Augmentation: To overcome the small size of the dataset and simulate real-world scenarios, augmentation techniques such as rotation, mirroring, zooming, and brightness adjustments were used. Consequently, a number of variations were added to the data set, strengthening the robustness of the models.

3.2.4 Data augmentation

The dataset comprises five classes: Safe Driving, Turning, Texting, Talking Phone, and Other, with each class stored in separate folders. Data augmentation techniques, including rotation, flipping, zooming, brightness adjustment, and cropping, were applied to simulate real-world scenarios while preserving original captions. Augmented images were stored with their respective classes to maintain accurate labels automatically.

The expansion process significantly expanded the data set by providing each class with a set of unique image variations. These enhanced photos maintained their semantic meaning while capturing the natural variations in driver behavior as well as external elements such as lighting and angle changes. This expanded dataset was used to train deep learning models, improving their performance in real-world scenarios and generalization.

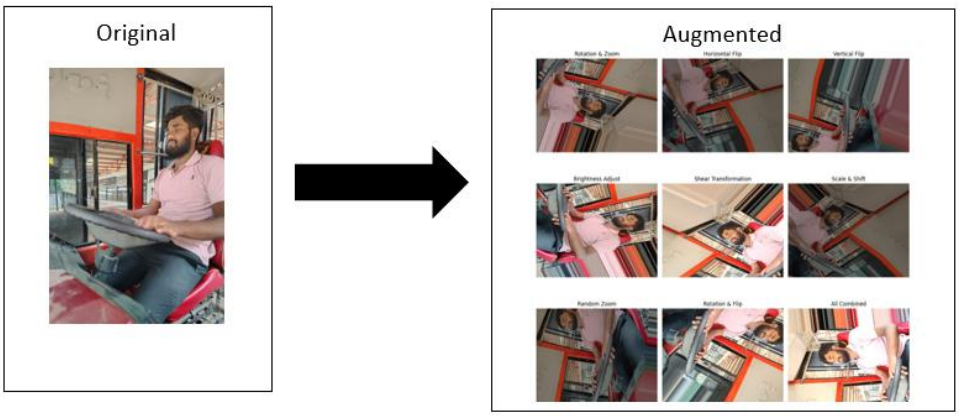

Figure 3.3 Original Image and Augmented Variations

This 3.9 displays an example of data augmentation techniques applied to an original image. The left side shows the original image of a driver, while the right

side illustrates augmented variations, including rotation, zoom, vertical flip, brightness adjustment, shear transformation, scaling, and combinations of these techniques. These augmentations increase dataset diversity, enhancing model generalization and performance. The transformations are labeled for clarity.

3.2.5 Train, Test, Validation

In a ratio of 85:10:5, the collected data set was divided into three different subsets: the test, validation, and training sets. More specifically, the training dataset included 85% of the images, the validation dataset included 10% for evaluating and refining the model during training, and the testing dataset included the remaining 5%. The test set was specifically reserved for evaluating the final performance of the model on unknown data to ensure an unbiased assessment of the generalization capabilities of the trained model.

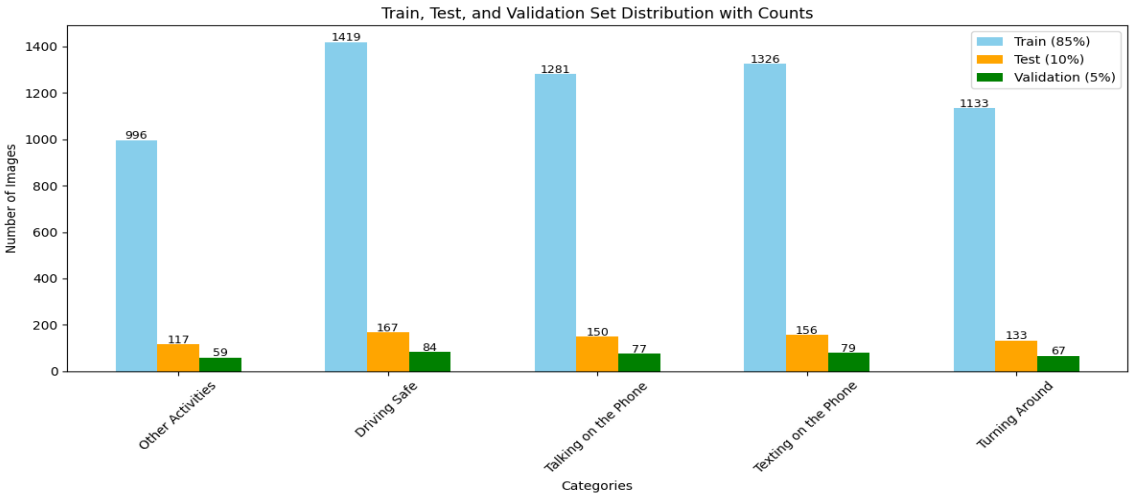

Figure 3.4 Dataset Distribution Across Categories

In this Figure 3.5 illustrates the distribution of images across five categories in the training, testing, and validation datasets, highlighting their respective proportions: 85% (training), 10% (testing), and 5% (validation). The categories include "Other Activities," "Driving Safe," "Talking on the Phone," "Texting on the Phone," and "Turning Around." Each category demonstrates a significantly higher count in the training set compared to testing and validation. The exact counts for each set are labeled on the bars for clarity. This distribution ensures a balanced dataset for robust model training and evaluation.

3.2.6 Train Model

The machine learning model chosen for this project is a Convolutional Neural Network (CNN), known for its deep architecture and strong performance in image classification tasks. The model was customized to suit the specific requirements of the project, focusing on the identification of driver behaviors (or the relevant focus area). Adjustments were made to optimize the model for the project's image data processing needs, ensuring high accuracy in real-time

predictions.

VGG16: VGG16, developed by A. Zisserman and K. Simonyan from the University of Oxford, is a convolutional neural network with 16 layers. This deep architecture, which employs smaller convolutional filters in a stack of layers, is well known. This design is appropriate for tasks requiring detailed pattern recognition because it allows the extraction of complex feature hierarchies.

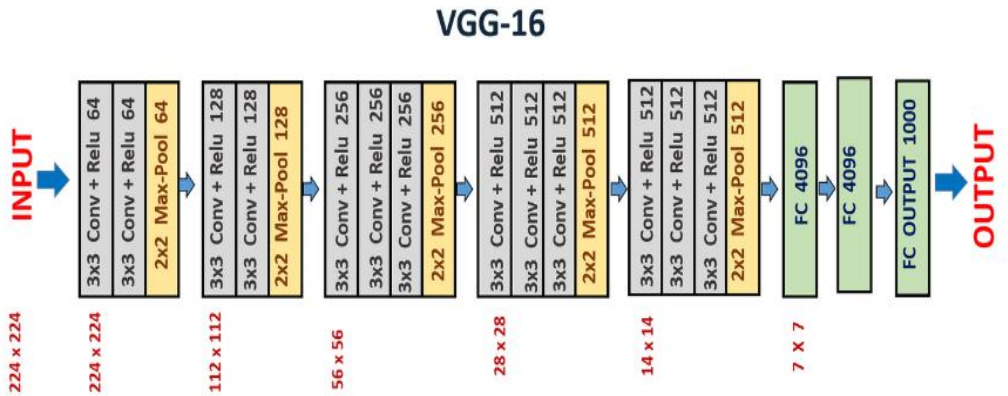

Figure 3.6 VGG-16 Architecture Overview

This Figure 3.6illustrates the architecture of the VGG-16 convolutional neural network. It processes 224x224 input images through 16 layers, including 3x3 convolutional layers with ReLU activation, followed by 2x2 max-pooling layers. The architecture increases the depth of feature extraction through 64, 128, 256, and 512 filters before transitioning to fully connected (FC) layers. The final layers output predictions across 1,000 object categories. This design emphasizes simplicity with smaller kernels while achieving high classification accuracy.

ResNet50: ResNet50 addresses the vanishing gradient issue in deep networks by introducing residual connections. Because of these connections, training very deep architecture with high accuracy and effective feature learning is possible.

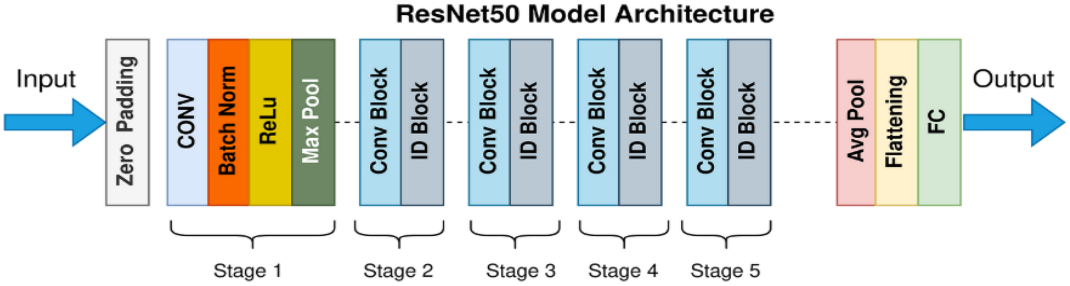

Figure 3.7 ResNet-50 Model Architecture

This Figure 3.7 illustrates the architecture of ResNet-50, a deep convolutional neural network designed for image classification tasks. It begins with an input layer followed by zero-padding, convolution, batch normalization, ReLU

activation, and max-pooling layers in Stage 1. The subsequent stages (2–5) feature identity (ID) and convolutional blocks, introducing skip connections to enable efficient training of deeper networks. The final layers include average pooling, flattening, and fully connected (FC) layers, producing the output predictions. This architecture leverages residual learning to mitigate the vanishing gradient problem.

InceptionV3: InceptionV3 uses an advanced module design to capture multi-scale features in images. It is a good option for large-scale image classification because of its architecture, which balances model complexity and computational efficiency.

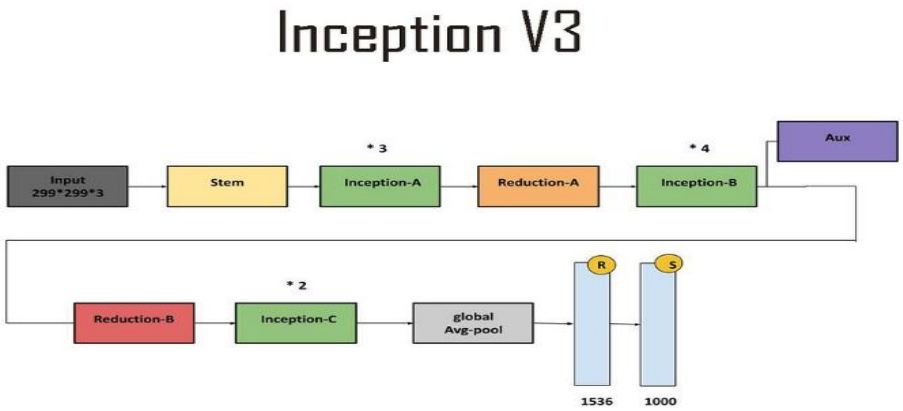

Figure 3.8 Inception V3 Model Architecture

This Figure 3.83 depicts the architecture of Inception V3, a convolutional neural network designed for image classification. It starts with an input layer (299x299x3), followed by the Stem module for feature extraction. The model includes stacked Inception-A, Reduction-A, Inception-B, and Reduction-B modules, employing parallel convolutional operations to capture diverse features. Inception-C modules follow, with global average pooling used for dimensionality reduction. The network ends with fully connected layers to produce predictions across 1,000 categories, with an auxiliary classifier (Aux) added to assist during training.

Xception: This extension of the Inception architecture, which is based on depth wise separable convolutions, improves computational efficiency while maintaining a high feature extraction capacity. For real-time applications, its lightweight design and high accuracy make it especially desirable.

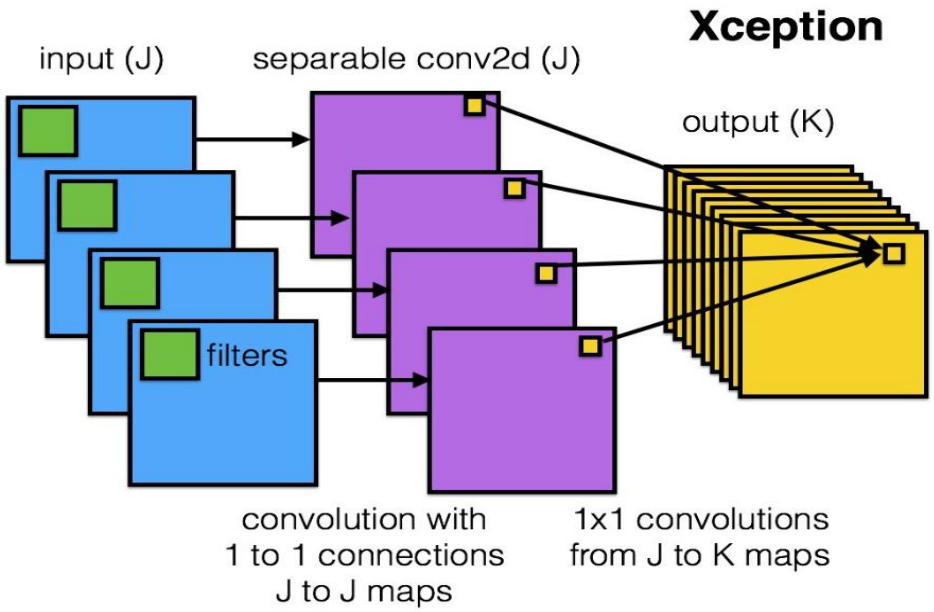

Figure 3.9 Xception Model Architecture

This Figure 3.9.14 outlines the architecture of the Xception model, which utilizes depth wise separable convolutions. The input feature maps (J) are processed through separable Conv2D layers, where each spatial convolution is followed by a 1x1 pointwise convolution. This approach efficiently captures spatial and cross-channel correlations. The output layer (K) aggregates features into the desired dimensions. The Xception architecture emphasizes efficiency and performance by replacing traditional convolutions with separable ones.

CNN: A convolutional neural network (CNN) is a category of machine learning model. Specifically, it is a type of deep learning algorithm that is well suited to analyzing visual data. CNNs are commonly used to process image and video tasks. And, because CNNs are so effective at identifying objects, they are frequently used for computer vision tasks, such as image recognition and object recognition, with common use cases including self-driving cars, facial recognition and medical image analysis.

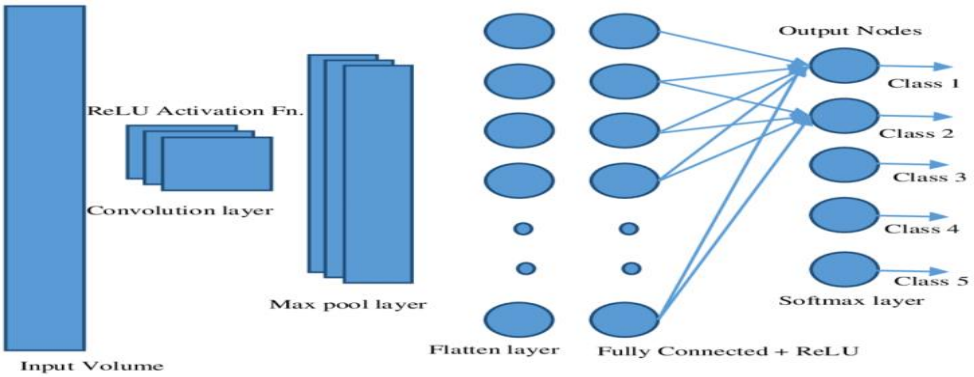

Figure 3.15 CNN Model Architecture.

Figure 3.15 illustrates the architecture of a Convolutional Neural Network

(CNN) used for classification tasks. The network begins with an input volume, such as an image, which passes through a convolutional layer where local features are extracted using filters, followed by ReLU activation to introduce non-linearity. A max pooling layer then reduces the spatial dimensions of the feature maps, retaining key features while improving computational efficiency. The resulting feature maps are flattened into a single vector, which is fed into fully connected layers to learn high-level representations. Finally, a softmax layer outputs class probabilities, with the highest probability determining the predicted class. This hierarchical structure demonstrates CNN's ability to transform raw data into meaningful classifications through successive layers.

Xception.Custom_Model: Driver behavior classification improves road safety and enables intelligent transportation systems. The Modified Xception Model, shown in the snapshot code, is designed to address this challenge efficiently. It uses depthwise separable convolutions for feature extraction, residual connections for stable learning, and dropout layers to reduce overfitting. Batch normalization enhances training stability, making the model robust for multi-class classification tasks. This approach balances efficiency and accuracy, ensuring its applicability to real-world scenarios.

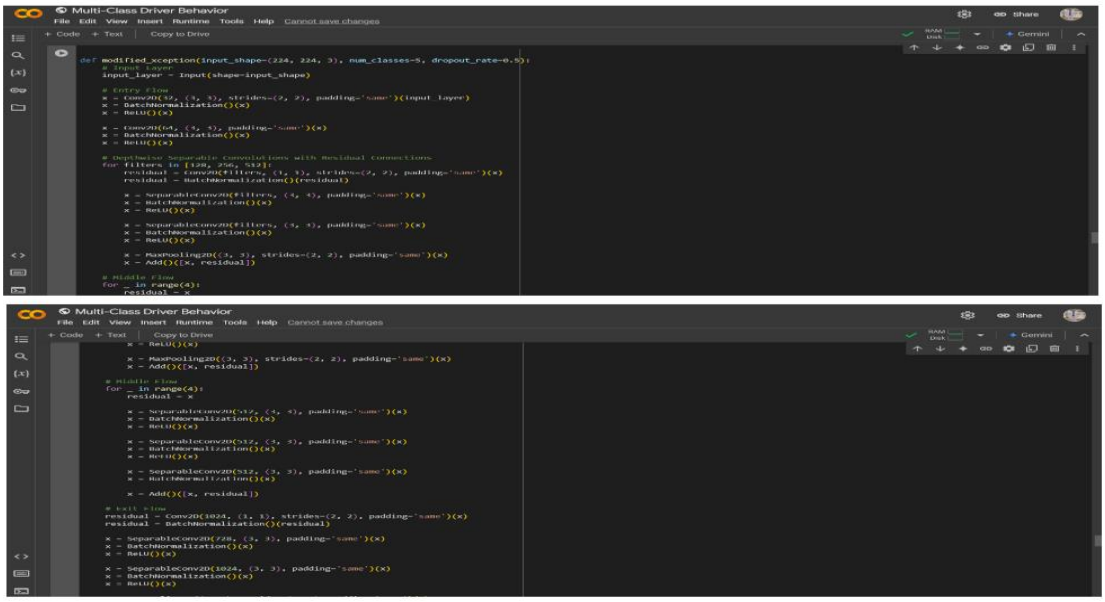

Figure 3.16 Modified_Xception_Model Code Snapshot.

Figure 3.16 showcases the code implementation of a Modified Xception Model designed for multi-class driver behavior classification. The model begins with an input layer and in depthwise separable convolution layers paired with batch normalization and ReLU activation to enhance computational efficiency. Residual connections are incorporated to maintain gradient flow and improve learning stability. The architecture is refined through repeated separable

convolutional blocks, enabling hierarchical feature extraction. Dropout layers are included to mitigate overfitting, and the final dense layer performs multi-class classification. This implementation reflects an optimized approach to balancing model complexity and efficiency.

Table 3.2: Comparison of Base and Modified Xception Models

Aspect	Base Xception Model	Modified Xception Model
Input Size	Fixed at (299, 299, 3) 299×299 = 89,401	Customizable (e.g., (224, 224, 3)) 224 × 224 = 50,176
Output Classes	Fixed at 1000 (ImageNet)	Customizable for specific tasks, such as 5 classes for domain-specific problems.
Entry Flow Filter Sizes	Uses 32, and 64 filters in the initial Conv2D layers.	Similar filter sizes (32, 64 filters) but adapted for new input dimensions.
Middle Flow Blocks	It contains 8 middle flow blocks.	Reduced to 4 blocks with 512 filters per block for faster processing and improved efficiency.
Dropout	Not included	Integrated dropout layers (e.g., 50% rate) to enhance regularization and prevent overfitting.
Residual Connections	Predefined	Enhanced with explicit Conv2D layers (e.g., 128, 256, 512 filters) for better feature learning.
Exit Flow	Fixed filters	Tuned filters: 728, 1024, 1536, 2048 filters, optimized for domain-specific tasks.
Flexibility	Rigid structure	Highly adaptable, allowing customization for unique use cases and datasets.

Table 3.2 highlights how the modified Xception model enhances flexibility by supporting customizable input sizes and output classes, addressing the limitations of the fixed structure in the base model. It improves efficiency by reducing middle flow blocks from 8 to 4 and incorporates dropout for improved regularization. The model further strengthens adaptability with enhanced

residual connections and optimized filters for specific tasks. It offers a more efficient and task-oriented solution compared to the rigid base Xception model.

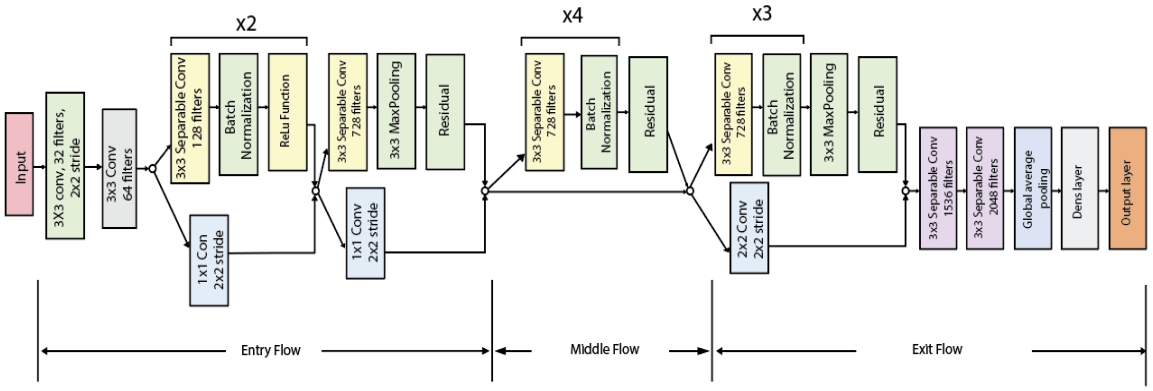

Figure 3.17 Xception Custom_Model Architecture.

Figure 3.17 The Xception architecture is a deep convolutional neural network for image classification, comprising three components: Entry Flow, Middle Flow, and Exit Flow. The Entry Flow uses convolutional layers, separable convolutions, and max-pooling to extract features efficiently, supported by residual connections for stable gradient flow. The Middle Flow repeats separable convolutional layers four times to capture complex spatial patterns, while residual connections prevent overfitting. The Exit Flow refines features using deeper separable convolutions, global average pooling, and a final fully connected layer for predictions. Depthwise separable convolutions throughout ensure high performance and computational efficiency, making Xception ideal for complex datasets.

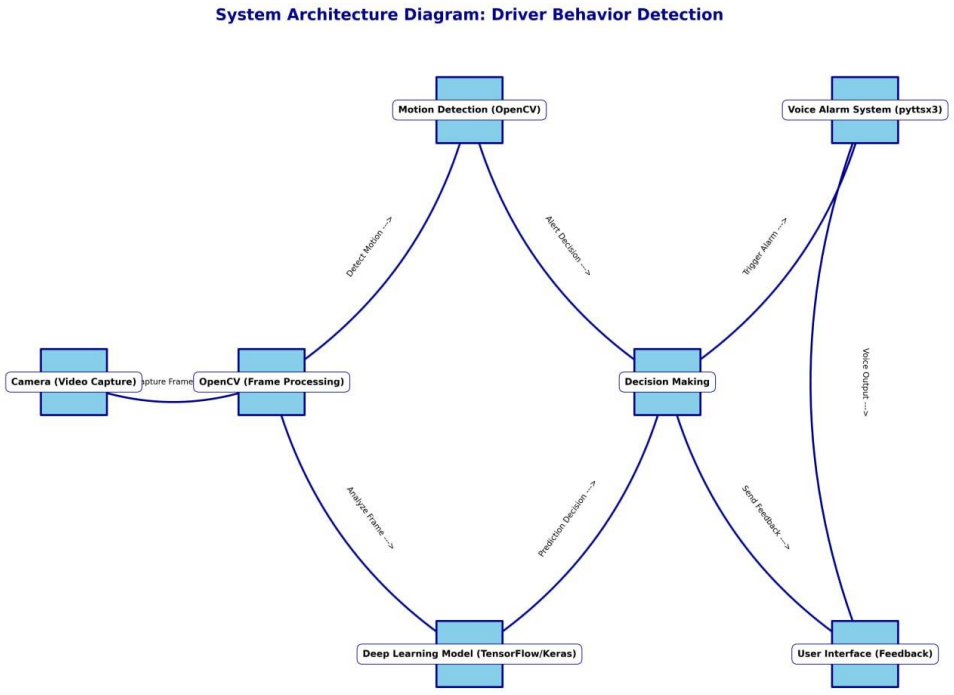

Figure 3.18 System Architecture diagram.

Figure 3.18 represents the concept of detecting driver behavior in the real world. This workflow demonstrates how the system processes data from a camera to a voice alarm and user interface feedback step-by-step: Camera (Video Capture) The system begins by capturing real-time video frames using the connected camera. A front-facing camera mounted in a vehicle captures a driver's activity. OpenCV (Frame Processing) The video frames are sent to the OpenCV module, which processes the images for clarity and prepares them for further analysis. The frames are converted to grayscale to simplify computations, or resized to match the input requirements of subsequent modules. Motion Detection (OpenCV) Processed frames are analyzed for motion using OpenCV algorithms like frame differencing. Sudden movements such as a driver's hand reaching out are detected by calculating pixel intensity changes between consecutive frames. Deep Learning Model (TensorFlow/Keras) Simultaneously, frames are sent to a deep learning model for activity classification. The model classifies the driver's activity into categories like "Driving Safely," "Using Phone," or "Turning." Decision Making Outputs from the motion detection and deep learning model are combined to make decisions. Voice Alarm System (pyttsx3) Based on the decision; a voice alarm is triggered to alert the driver. User Interface (Feedback) Simultaneously, the system logs the activity and outputs feedback to a user interface for monitoring. This workflow ensures efficient detection and real-time feedback to improve safety.

3.3 Project Plan

Table 3.3 is not the exact timeline of work, but it was the initial planning for our project.

Table 3.3: Project Plan timeline.

Tasks	Weeks																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Data Collection																				
Data Pre-processing																				
Model implementation and result analysis																				
Final Model Selection																				
Report Writing																				
Simulation development																				

3.4 Task Allocation

Table 3.4 represents the contribution of each team member for every task.

Table 3.4: Task allocation of team.

Task	Team Mate	
	Team mate 1	Team mate 2
Literature review	No	Yes
Dataset Collection	Both	Both
Data labeling	No	Yes
Data preprocessing	Yes	No
Model Run	Yes	No
Model Selection	Both	Both
Report Writing	Both	Both

3.5 Summary

The objective of the project is to use innovative CNN architectures like VGG16, ResNet50, InceptionV3, and Xception to precisely identify driver behaviors in real-time applications. Detailed data collection and preprocessing, CNN model training and optimization, and the creation of an intuitive web application for real-time analysis and feedback are all part of the method's structured approach. To guarantee reliable classification, the dataset—which was gathered under actual driving circumstances—is divided into five different behaviors. The efficiency, scalability, and efficacy of the system in enhancing traffic safety and lowering accident rates are guaranteed by comprehensive design specifications, model validation methodologies, and the incorporation of cutting-edge tools and technologies.

Chapter 4

Implementation and Results

4.1 Environment Setup

To start working on this project, environment setup was the most crucial part. As computational performance has a huge contribution on the result. For this, appropriate environment setup analysis was done before working on this project. Following are the device requirements for our project.

4.1.1 Hardware Requirements

1. Development Machine
   - Processor: Intel Core i5 or higher
   - RAM: Minimum 8 GB
   - Storage: 512 GB SSD or higher
   - GPU: Optional, but recommended for accelerated model training

2. Deployment Server
   - Processor: Intel Xeon or equivalent
   - RAM: Minimum 16 GB
   - Storage: 500 GB SSD or higher
   - GPU: NVIDIA Tesla or equivalent.

4.1.2 Software Requirements

1. Operating System
   - Development Environment: Windows 11 or Ubuntu 18.04
   - Deployment Environment: Ubuntu 18.04 or higher, or Windows Server
2. Programming Language
   - Python 3.8 or higher
3. Libraries and Frameworks
   - TensorFlow/Keras: For machine learning model development
   - Flask: For web application development
   - NumPy: For numerical computations

- Pandas: For data manipulation
- Scikit-learn: For auxiliary machine learning tools
- Matplotlib: For data visualization
- OpenCV: For computer vision-related tasks
- Pillow (PIL): For image handling and preprocessing
- Warnings Library: For managing warning filters

4. Development Tools
   - Jupiter Notebook: For prototyping and testing
   - Anaconda: For managing Python environments and dependencies
   - Visual Studio Code (VS Code): For code development and debugging
   - Kaggle: For dataset acquisition, model benchmarking, and experimentation
5. Web Server
   - Apache or Nginx for deployment and handling web traffic

4.2 Comparative Analysis

This study successfully blends the benefits of pre-trained features with dataset-specific adaptations by optimizing the pre-trained Convolutional Neural Network (CNN) architectures while freezing their initial layers, improving feature extraction and overall performance. By optimizing weight updates and reducing classification errors, the RMSprop optimizer and categorical cross-entropy loss ensure effective training. Together, these improvements improve the models' capacity to accurately categorize and generalize driver behavior.

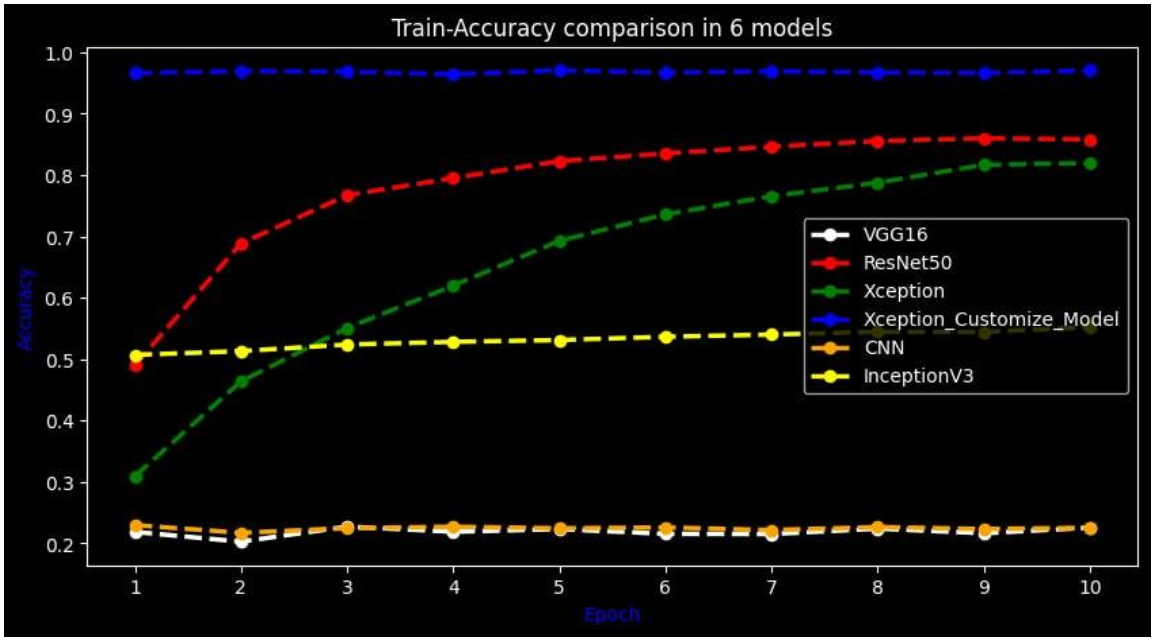

Figure 4.1 Training Accuracy Comparison Across Six Models

Figure 4.1 illustrates the training accuracy trends of six deep learning models (VGG16, ResNet50, InceptionV3, Xception, Xception_Customize_Model, and CNN) over 10 epochs. Among these, the Xception_Customize_Model (blue dashed line) achieved the highest training accuracy, consistently nearing 1.0, followed closely by Xception (red dashed line), with accuracy stabilizing slightly below 0.950. ResNet50 (green dashed line) showed competitive performance, achieving accuracies between 0.8200 and 0.8800 over the epochs. InceptionV3 (yellow solid line) displayed moderate accuracy, converging around 0.7000, while VGG16 (white dotted line) and CNN (orange dashed line) showed significantly lower training accuracy. Both VGG16 and CNN started at approximately 0.2000 and stabilized around 0.2300 and 0.3000, respectively. This comparison highlights the superior training performance of advanced architectures like Xception and its customized variant, demonstrating their capability to achieve higher accuracies compared to simpler models. The graph effectively uses distinct colors and line styles to differentiate the models, aiding in easy visual analysis of their performance trends.

Table 4.1 Model Accuracy Comparison

Epoch	CNN Accuracy (%)	VGG16 Accuracy (%)	ResNet50 Accuracy (%)	Inception V3 Accuracy (%)	Xception Accuracy (%)	Xception. Custom_Model Accuracy (%)
1	21.06	20.76	82.97	23.61	24.82	96.71
2	19.89	22.05	83.47	22.88	44.16	96.77
3	22.13	22.04	83.71	23.32	52.8	96.58
4	22.57	21.88	84.98	22.78	60.4	96.85
5	22.05	22.62	86.38	22.68	67.33	96.35
6	21.85	22.05	86.23	22.20	72.39	96.88
7	22.55	22.30	87.62	22.30	75.73	96.9
8	21.85	23.16	88.48	21.17	77.68	96.71
9	23.02	21.03	87.79	21.88	81.21	96.3
10	23.93	22.3	88.22	23.25	83.24	97.07

Table 4.1 presents the training accuracy trends across six deep learning models (CNN, VGG16, ResNet50, InceptionV3, Xception, and Xception_Customize_Model) over 10 epochs, with all values expressed as percentages. The Xception_Customize_Model consistently outperformed all other models, achieving the highest accuracy of 97.07% at epoch 10, demonstrating its superior learning capacity and stability throughout training. The Xception model followed as a strong contender, peaking at 96.98% in epoch

7 and ending with an accuracy of 96.3% at epoch 10. ResNet50 showed robust performance with steady improvement, concluding at 88.22% in the final epoch. In contrast, InceptionV3, VGG16, and CNN exhibited significantly lower accuracies, with CNN achieving the highest peak among them at 23.93%. This table highlights the superior training performance of advanced architectures like Xception_Customize_Model and Xception, emphasizing their efficiency and effectiveness in classification tasks, while other models demonstrated significant room for improvement.

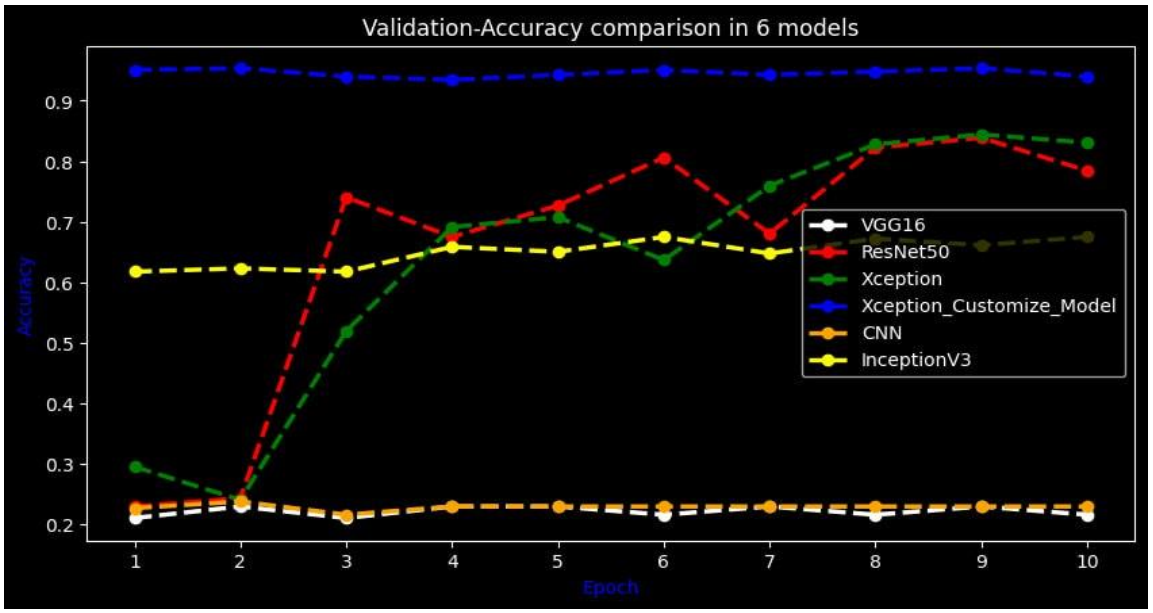

Figure 4.2 Validation Accuracy Comparison Across Six Models

Figure 4.2 depicts the validation accuracy trends for six deep learning models (VGG16, ResNet50, InceptionV3, Xception, Xception_Customize_Model, and CNN) over 10 epochs. Among these, the Xception_Customize_Model (blue dashed line) exhibited the highest and most consistent validation accuracy, stabilizing above 0.900. The Xception model (red dashed line) achieved competitive accuracy but displayed significant fluctuations, peaking around 0.800. ResNet50 (green dashed line) showed a steady increase in validation accuracy, converging near 0.750 after the initial epochs. InceptionV3 (yellow solid line) demonstrated moderate performance, stabilizing around 0.700. On the other hand, VGG16 (white dotted line) and CNN (orange dashed line) presented lower validation accuracy trends, both starting at approximately 0.200 and barely exceeding 0.300 by the final epoch. This graph highlights the robustness of the Xception_Customize_Model in achieving high validation accuracy, while other models, such as ResNet50 and Xception, also performed well with varying stability. The clear use of distinct colors and line styles enhances the interpretability of the performance comparison among the models.

Table 2.2 Model Validation Accuracy Comparison

Epoch	CNN Validation Accuracy (%)	VGG16 Validation Accuracy (%)	ResNet50 Validation Accuracy (%)	Inception V3 Validation Accuracy (%)	Xception Validation Accuracy (%)	Xception .Custom_ Model Validation Accuracy
1	21.31	21.58	80.05	22.68	29.51	95.08
2	21.31	21.04	81.97	22.95	24.04	95.36
3	22.68	21.58	85.52	22.95	51.91	93.99
4	23.5	21.04	84.43	22.95	69.13	93.44
5	22.4	22.95	84.43	22.95	70.77	94.26
6	18.85	22.95	83.61	21.58	63.66	95.08
7	22.95	22.95	86.34	22.95	75.96	94.26
8	22.68	21.04	84.15	22.4	82.79	94.81
9	22.95	21.04	77.87	22.21	84.43	95.36
10	22.95	22.95	88.25	22.13	83.06	95.99

Table 4.2 shows the validation accuracy of six deep learning models (CNN, VGG16, ResNet50, InceptionV3, Xception, and Xception_Customize_Model) across 10 epochs, with all values expressed as percentages. The Xception_Customize_Model achieved the highest validation accuracy throughout, peaking at 95.99% in epoch 10, highlighting its exceptional generalization ability. The Xception model also performed well, maintaining competitive accuracy with a peak of 83.06% in the final epoch. ResNet50 demonstrated consistent performance, steadily improving to achieve 88.25% by epoch 10. However, models like InceptionV3, VGG16, and CNN displayed significantly lower validation accuracy, with all three struggling to surpass 23%, highlighting their limitations in generalization. This table underscores the superior validation performance of the Xception_Customize_Model, followed by Xception and ResNet50, while other models require further optimization to enhance their validation accuracy.

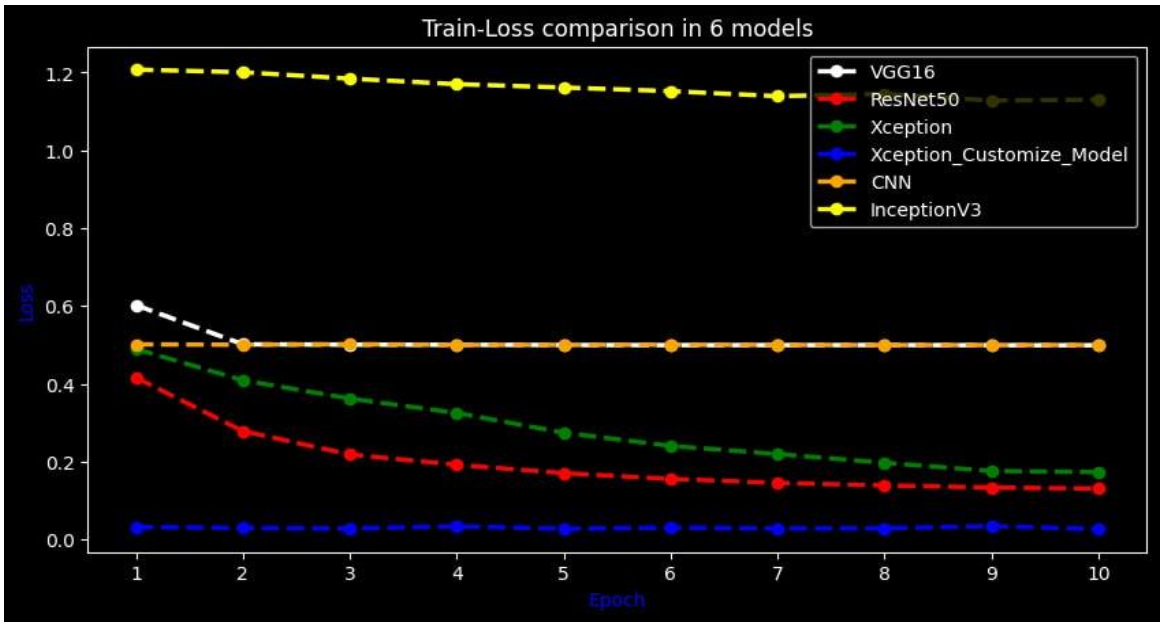

Figure 4.3 Training Loss Comparison Across Six Models

Figure 4.3 illustrates the training loss trends for six deep learning models (VGG16, ResNet50, InceptionV3, Xception, Xception_Customize_Model, and CNN) over 10 epochs. The Xception_Customize_Model (blue dashed line) achieved the lowest training loss, stabilizing close to 0.0 by the final epochs, indicating superior convergence. The Xception model (red dashed line) also demonstrated a significant decrease in loss, stabilizing below 0.200. ResNet50 (green dashed line) exhibited a steady decline in loss, converging around 0.300. InceptionV3 (yellow solid line) and CNN (orange dashed line) displayed relatively higher loss values throughout the epochs, with minimal improvement and stabilization around 0.400 and 0.500, respectively. VGG16 (white dotted line) showed moderate performance, with loss stabilizing near 0.600. The graph effectively highlights the superior training efficiency of advanced architectures like Xception_Customize_Model and Xception, which achieve lower training loss compared to simpler models. The use of distinct colors and line styles ensures clarity in distinguishing model performance.

Table 4.3 Model Training Loss Comparison

Epoch	CNN Loss (%)	VGG16 Loss (%)	ResNet50 Loss (%)	Inception V3 Loss (%)	Xception Loss (%)	Xception. Custom_Model Loss
1	102.06	73.13	15.99	15.99	53.64	3.15
2	57.6	50.04	15.71	15.71	42.16	3.14
3	51.99	50.04	15.67	15.67	37.36	3.29
4	50.81	49.94	13.9	13.9	33.24	3.15
5	50.86	50.03	13.16	13.16	28.49	3.25

6	50.47	50.07	12.63	12.63	24.59	2.93
7	50.41	50.02	11.85	11.85	22.67	3.03
8	50.31	49.97	11.2	11.2	20.78	2.97
9	50.29	49.98	11.41	11.41	17.7	4.26
10	50.35	49.94	11.36	11.36	16.64	4.01

Table 4.3 compares the training loss trends across six deep learning models (CNN, VGG16, ResNet50, InceptionV3, Xception, and Xception_Customize_Model) over 10 epochs, with all values expressed as percentages. The Xception_Customize_Model consistently demonstrated the lowest training loss, starting at 3.15% in the first epoch and stabilizing around 4.01% by the 10th epoch, highlighting its superior convergence and efficient learning. The Xception model followed with slightly higher loss values, starting at 53.64% and reducing to 16.64% by the final epoch, showcasing steady improvement. ResNet50 exhibited robust performance with loss values starting at 15.99% and declining to 11.36% by epoch 10. Meanwhile, InceptionV3 mirrored ResNet50's trend with similar loss values. In contrast, VGG16 and CNN displayed significantly higher loss percentages throughout the training process. CNN started at an exceptionally high loss of 102.06%, and although it reduced over time, it remained around 50.35% in the final epoch. Similarly, VGG16 ended with a loss of 49.94%, highlighting its limited training efficiency. This table underscores the superior training performance of Xception_Customize_Model, followed by Xception and ResNet50, while models like CNN and VGG16 demonstrated substantial room for optimization.

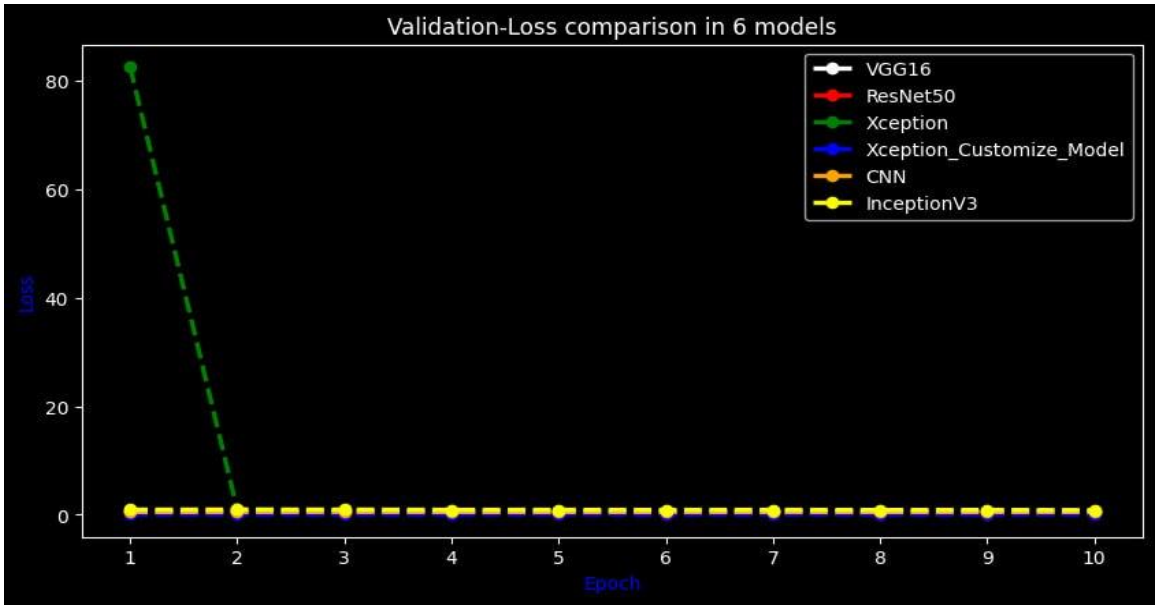

Figure 4.4 Validation Loss Comparison Across Six Models

Figure 4.4 depicts the validation loss trends for six deep learning models

(VGG16, ResNet50, InceptionV3, Xception, Xception_Customize_Model, and CNN) over 10 epochs. The ResNet50 model (green dashed line) started with a very high validation loss (above 80) but drastically reduced it to near-zero by the second epoch, maintaining this minimal loss throughout subsequent epochs. The InceptionV3 model (yellow solid line) consistently displayed the lowest validation loss from the beginning, stabilizing near zero across all epochs. Other models, including VGG16 (white dotted line), Xception (red dashed line), Xception_Customize_Model (blue dashed line), and CNN (orange dashed line), maintained near-zero validation loss throughout the epochs, indicating strong generalization and efficient training. The chart highlights the rapid reduction of validation loss for ResNet50 and the consistent minimal validation loss achieved by InceptionV3 and other models. Distinct colors and line styles are effectively used to differentiate between the models.

Table 4.4 Model Validation Loss Comparison

Epoch	CNN Validation Loss (%)	VGG16 Validation Loss (%)	ResNet50 Validation Loss (%)	Inception V3 Validation Loss (%)	Xception Validation Loss (%)	Xception .Custom_ Model Validation Loss
1	52.41	50.45	25.26	68.69	82.56	04.53
2	50.45	49.93	22.44	49.87	64.30	4.78
3	50.57	49.97	13.49	49.9	58.05	0.433
4	50.84	50.00	13.44	54.24	27.88	5.98
5	50.54	49.87	14.17	68.25	29.08	8.34
6	51.47	49.92	17.87	50.45	33.16	4.34
7	50.05	49.88	12.22	55.75	20.46	6.54
8	49.89	49.94	15.59	53.88	15.85	6.36
9	49.83	49.9	22.14	74.4	15.61	5.11
10	49.85	49.93	10.41	74.4	13.04	6.9

Table 4.4 displays the validation loss trends across six deep learning models (CNN, VGG16, ResNet50, InceptionV3, Xception, and Xception_Customize_Model) over 10 epochs, with all values expressed as percentages. The Xception_Customize_Model consistently achieved the lowest validation loss, beginning at 4.53% in the first epoch and maintaining stability with a loss of 6.9% by the 10th epoch, showcasing its superior generalization ability. The Xception model demonstrated competitive performance, starting at 82.56% and improving significantly to 13.04% by the final epoch, indicating its capability to reduce validation loss over time. ResNet50 exhibited steady progress, reducing its validation loss from 25.26% in the first epoch to 10.41% in

the 10th epoch. InceptionV3 showed higher and more variable validation loss, remaining around 74.4% by epoch 10. In contrast, CNN and VGG16 exhibited consistently higher validation loss values, with CNN starting at 52.41% and ending at 49.85%, and VGG16 starting at 50.45% and ending at 49.93%, reflecting limited capacity to generalize effectively. This table highlights the superior validation performance of the Xception_Customize_Model, followed by Xception and ResNet50, while other models exhibited significant limitations in minimizing validation loss.

4.3 Results and Discussion

4.3.1 Experimental Result

In this part, we provide a detailed examination of the results and analysis of the various models tested. This includes the performance metrics of each model and a summary of the findings. We specifically highlight the performance of our custom Xception model in comparison to other state-of-the-art models.

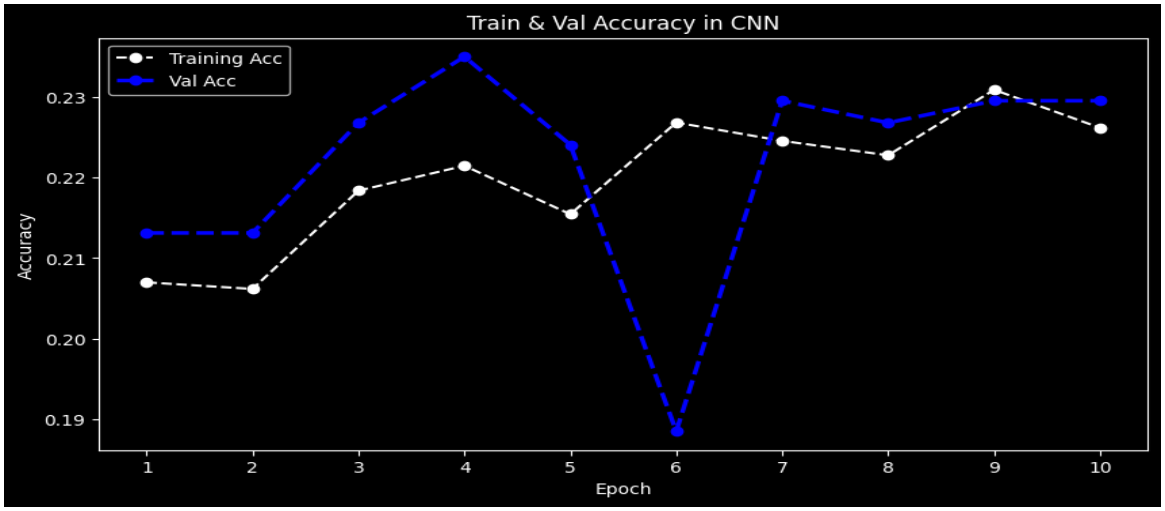

Figure 10 Training and Validation Accuracy for CNN

Figure 4.5 illustrates the training and validation accuracy trends across 10 epochs for the CNN model. Training accuracy started at 0.2100, gradually increased to 0.2300 by epoch 5, and maintained a steady improvement, peaking at 0.2320 in epoch 9. Validation accuracy demonstrated higher variability, starting at 0.2120, peaking at 0.2360 in epoch 4, but sharply declining in epoch 6 before recovering to 0.2310 in later epochs. The fluctuations in validation accuracy suggest possible instability or sensitivity to data variance during training. The dotted white line represents training accuracy, while the dashed blue line indicates validation accuracy, with distinct markers for clear visualization of trends.

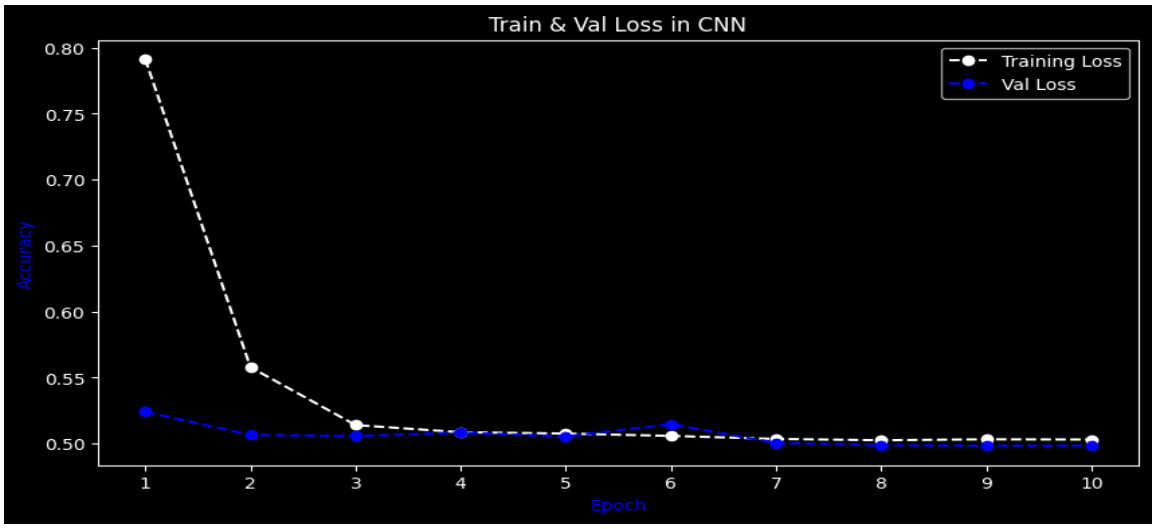

Figure 11 Training and Validation Loss for CNN

Figure 4.6 presents the training and validation loss trends across 10 epochs for the CNN model. The training loss began at 0.8000 in epoch 1, rapidly decreased to approximately 0.5000 by epoch 3, and stabilized around this value for the remaining epochs. Validation loss started at 0.5100, remained relatively stable throughout, and closely aligned with the training loss after epoch 3. The rapid initial decrease in training loss reflects effective model optimization, while the stability in validation loss suggests good generalization performance. The dotted white line represents training loss, and the dashed blue line indicates validation loss, with distinct markers enhancing clarity.

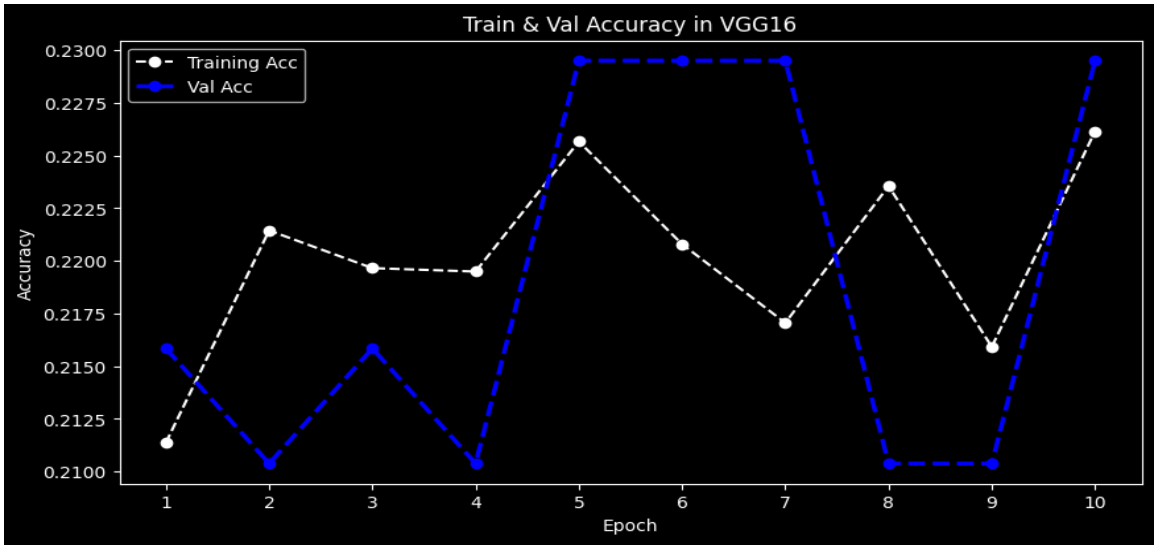

Figure 12 Training and Validation Accuracy for VGG16

Figure 4.7 illustrates the progression of training and validation accuracy over 10 epochs for the VGG16 model. Training accuracy improved from 0.2076 in epoch 1 to a peak of 0.2316 in epoch 8, with minor fluctuations thereafter. Validation accuracy started at 0.2158 and achieved its highest value of 0.2295 during epochs 5, 6, and 7. The consistent overlap and higher validation accuracy in later epochs

suggest adequate model generalization. The chart employs distinct line styles and markers (dotted white for training and dashed blue for validation) for clear visualization of trends.

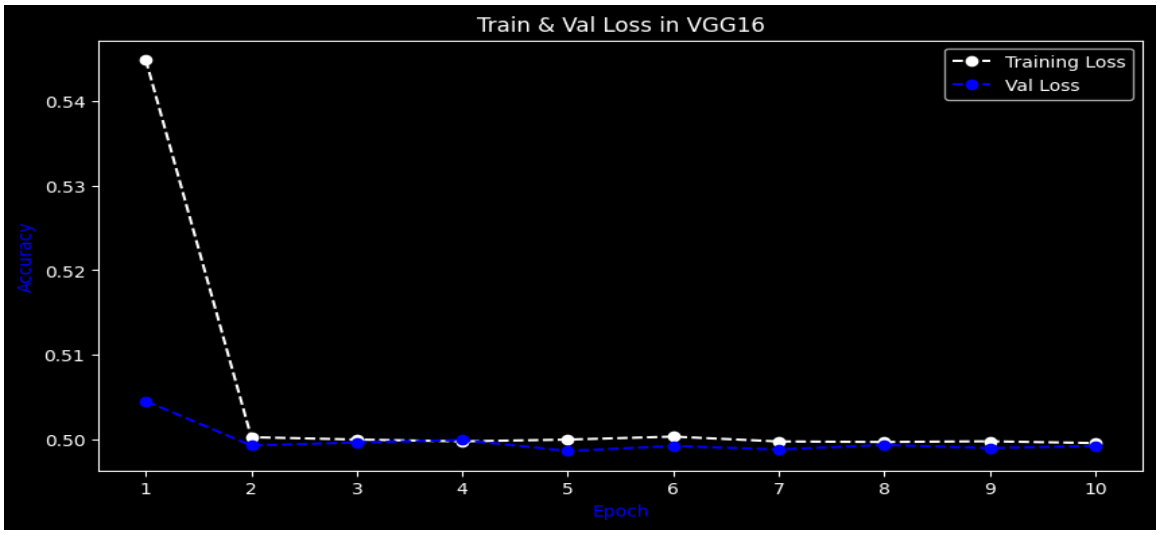

Figure 13 Training and Validation Loss for VGG16

Figure 4.8 presents the training and validation loss trends across 10 epochs for the VGG16 model. Training loss began at 0.7313 in epoch 1 and rapidly declined to stabilize around 0.5004 from epoch 2 onwards. Validation loss, starting at 0.5045, showed minor fluctuations but remained consistent near 0.4993 after the initial epoch. The chart demonstrates effective loss minimization during training, with the validation loss closely mirroring the training loss throughout, indicating minimal overfitting. Distinct line styles (dotted white for training loss and dashed blue for validation loss) facilitate easy interpretation of trends.

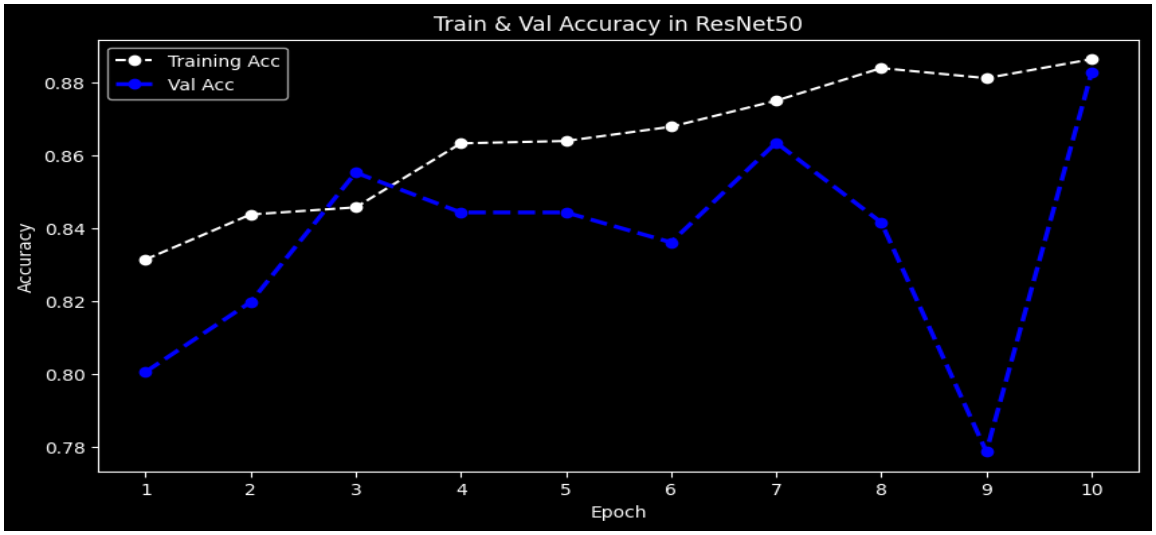

Figure 14 Training and Validation Accuracy for ResNet50

Figure 4.9 shows ResNet50's training and validation accuracy over 10 epochs. Training accuracy steadily rose from 0.8210 to 0.8800, while validation accuracy

fluctuated, dropping to 0.7800 in epoch 9 before recovering to 0.8800 in epoch 10. The graph highlights potential overfitting or data variability, with clear markers and colors distinguishing the trends.

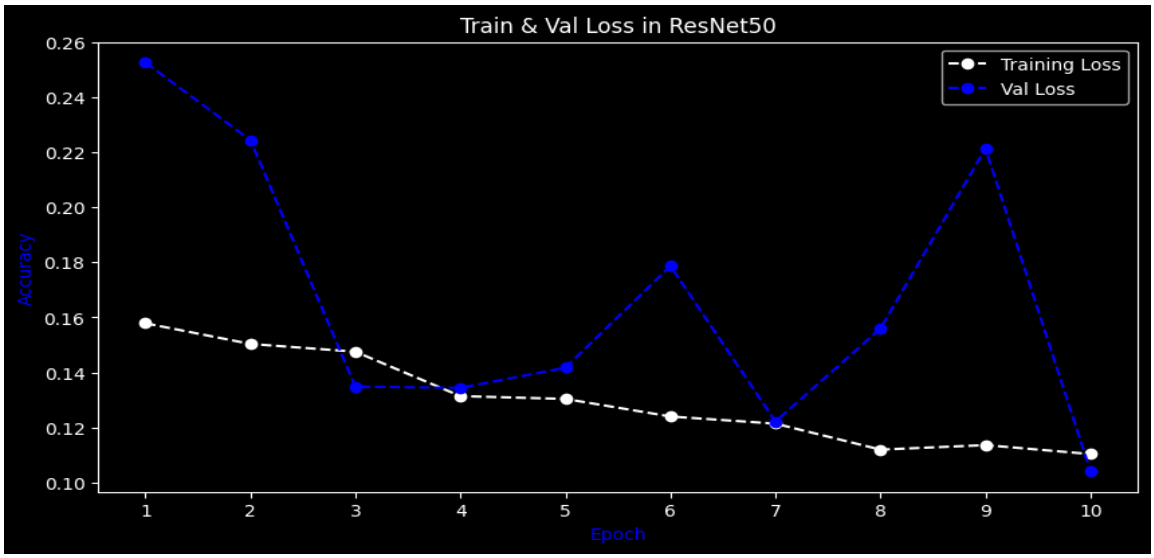

Figure 4.10 Training and Validation Loss for ResNet50

Figure 4.10 illustrates the training and validation loss trends over 10 epochs for the ResNet50 model. The training loss steadily decreased from 0.1600 in epoch 1 to approximately 0.1100 by epoch 10, indicating effective learning. Validation loss started at 0.2500 in epoch 1, showing a sharp decline by epoch 3, followed by fluctuations across subsequent epochs. The loss peaked at 0.2200 in Epoch 8 before dropping significantly to align closely with the training loss in Epoch 10. The chart demonstrates good convergence, with minimal divergence between training and validation loss in later epochs. Distinct line styles (dotted white for training loss and dashed blue for validation loss) ensure clear differentiation of trends.

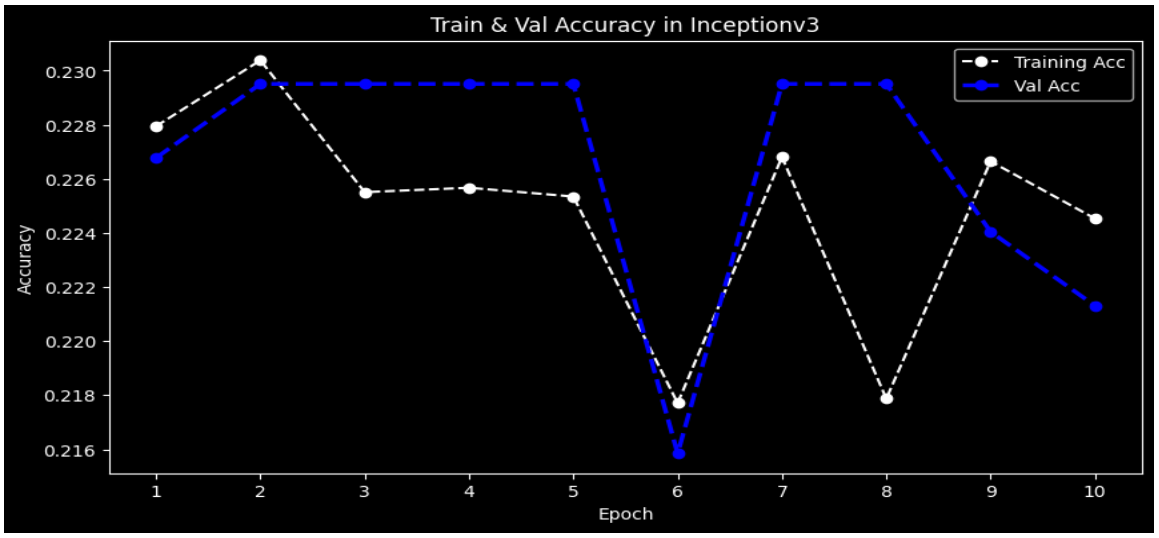

Figure 15 Training and Validation Accuracy for InceptionV3

This figure 4.11 shows the trends of training and validation accuracy over 10 epochs for the InceptionV3 model. Training accuracy starts at 0.2230 in epoch 1, peaks at 0.2300 in epoch 2, and fluctuates across subsequent epochs, reaching a low of 0.2160 in epoch 6 before recovering to 0.2270 in epoch 10. Validation accuracy follows a similar trend, starting at 0.2260, peaking at 0.2300 in epoch 2, and experiencing a notable drop to 0.2160 in epoch 6 before stabilizing around 0.2280 in later epochs. The chart highlights variability in accuracy, indicating potential challenges in convergence or generalization. The dotted white line represents training accuracy, while the dashed blue line indicates validation accuracy, ensuring visual clarity.

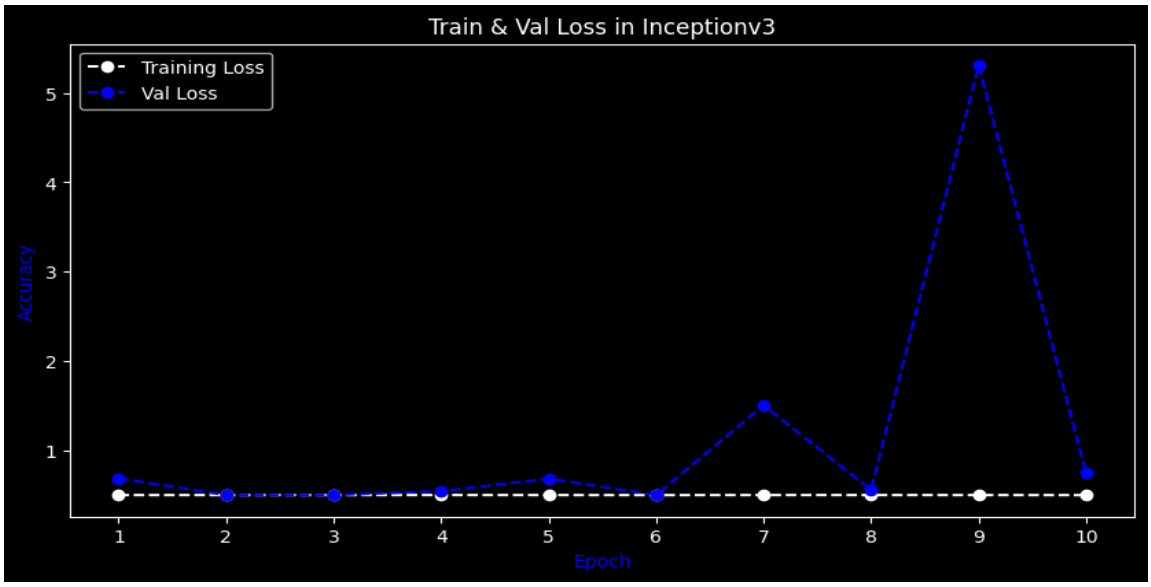

Figure 16 Training and Validation Loss for InceptionV3

Figure 4.12 presents the trends in training and validation loss over 10 epochs for the InceptionV3 model. Training loss remained consistently low and stable around 1.0 across all epochs, indicating effective learning. Validation loss, however, showed variability, starting at 1.0, spiking significantly to over 5.0 in epoch 9, and sharply dropping to align with the training loss in epoch 10. The pronounced spike in validation loss suggests a possible overfitting or anomaly during that epoch. The dotted white line represents training loss, and the dashed blue line indicates validation loss, with clear markers ensuring ease of comparison.

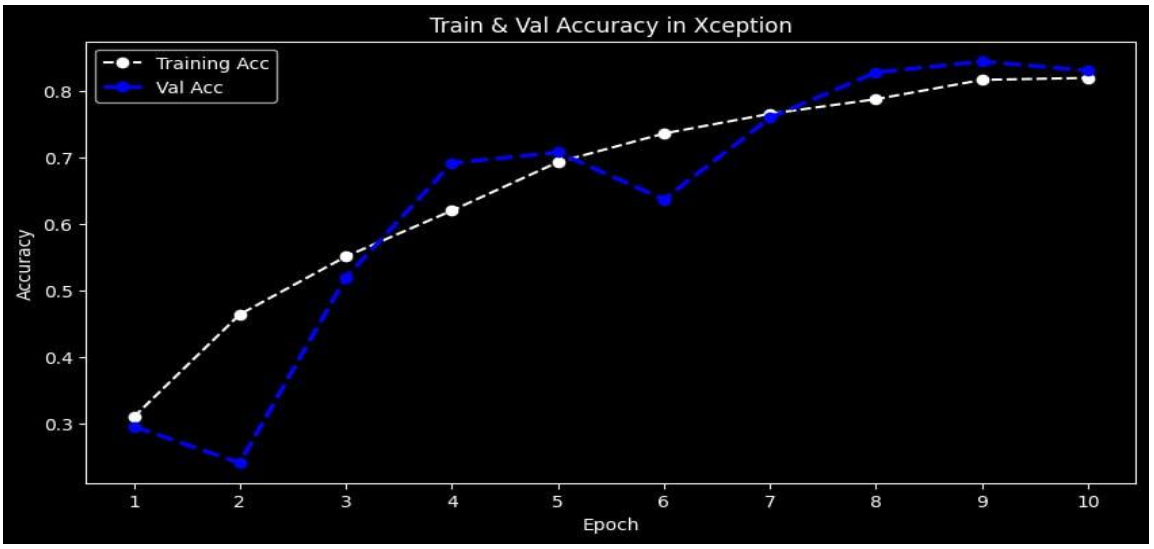

Figure 4.13 Training and Validation Accuracy for Xception

Figure 4.13 shows the training and validation accuracy trends for the Xception model over 10 epochs. Training accuracy steadily increases from 0.30 to 0.85, while validation accuracy follows a similar trend, slightly below training accuracy. The graph indicates minimal overfitting and efficient generalization, with distinct colors and line styles aiding performance assessment.

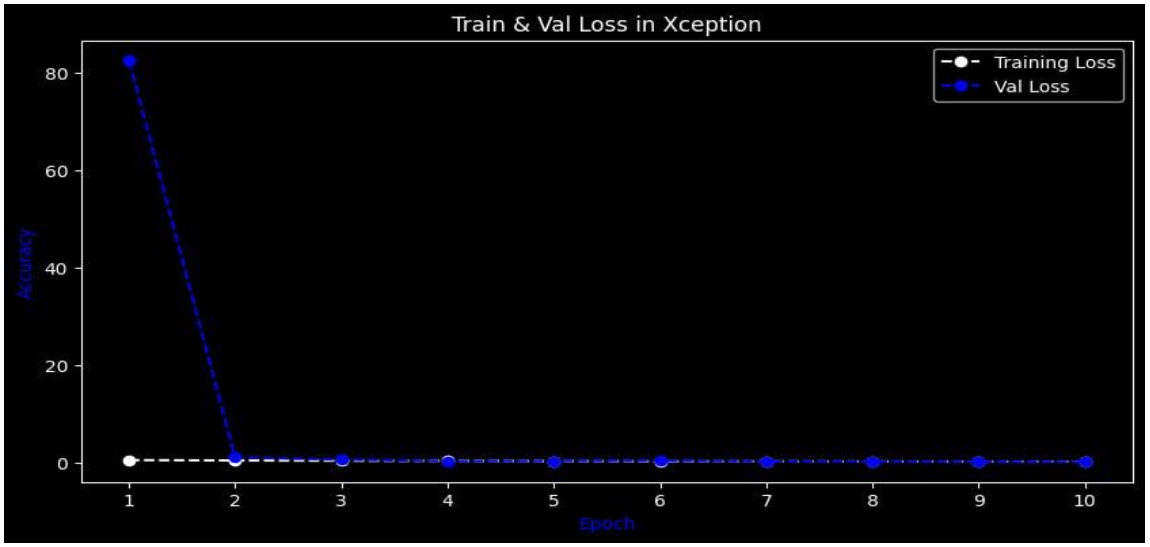

Figure 17 Training and Validation Loss for Xception

Figure 4.14 depicts the training and validation loss trends for the Xception model over 10 epochs. Validation loss dropped sharply from over 80 to near zero by the second epoch, maintaining minimal values thereafter. Similarly, training loss stabilized near zero early on. The graph highlights the model's rapid convergence and efficient training, with clear differentiation of metrics using distinct colors and line styles.

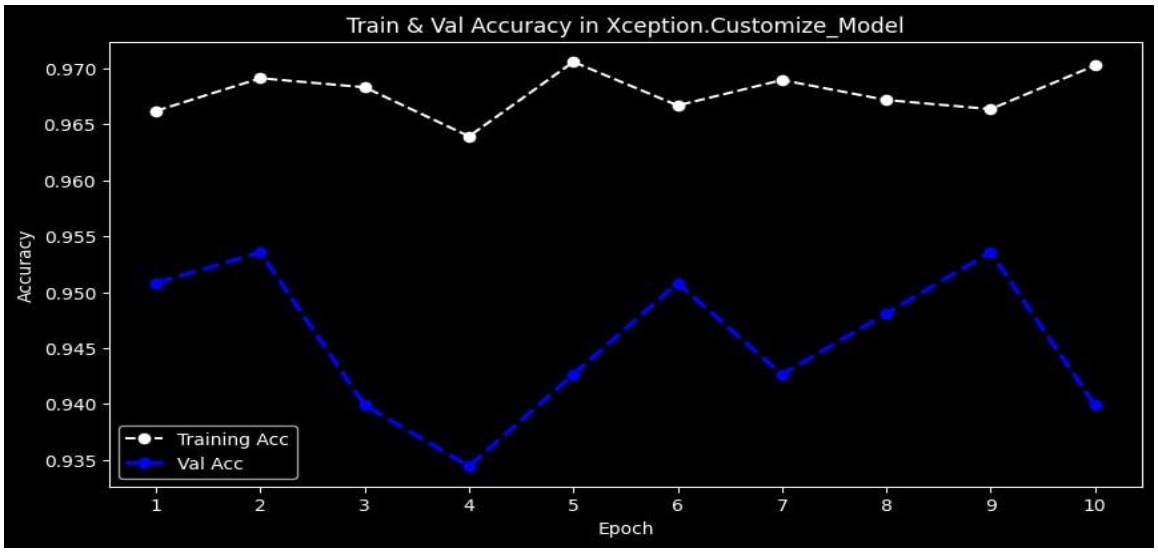

Figure 4.15 Training and Validation Accuracy for Xception.Customize_Model

Figure 4.15 shows the training and validation accuracy trends of the Xception_Customize_Model over 10 epochs. Training accuracy remained high, stabilizing around 0.970, while validation accuracy started near 0.950, dipped mid-way, and recovered to 0.955. The graph highlights minimal overfitting, good generalization, and clear performance differentiation through distinct line styles and colors.

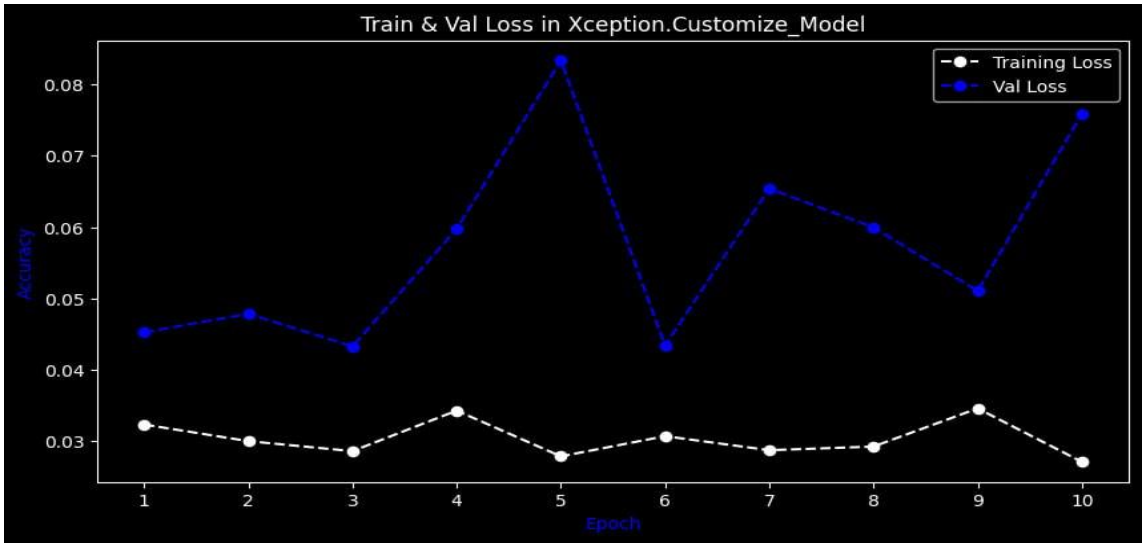

Figure 4.16 Training and Validation Loss for Xception.Customize_Model

Figure 4.16 shows the training and validation loss trends for the Xception_Customize_Model over 10 epochs. Training loss remained low, stabilizing around 0.03, while validation loss fluctuated, peaking around the 5th and 10th epochs above 0.08 but generally stabilizing below 0.07. The graph highlights efficient training with occasional variability in generalization, using distinct colors and styles for clarity.

Table 4.5 Model evaluation

Model Name	Training Accuracy (%)	Validation Accuracy (%)	Training Loss (%)	Validation Loss (%)
CNN	23.93	23.5	50.29	49.83
VGG16	23.16	22.95	49.94	49.87
ResNet50	88.48	88.25	11.2	10.41
InceptionV3	23.61	22.95	11.2	49.87
Xception	83.57	83.06	17.43	17.86
Xeception. Custom_Model	97.07	95.11	2.93	4.89

Table 4.5 provides a summary evaluation of six deep learning models (CNN, VGG16, ResNet50, InceptionV3, Xception, and Xception_Custom_Model) in terms of training accuracy, validation accuracy, training loss, and validation loss, with all values expressed as percentages. The Xception_Custom_Model outperformed all other models, achieving the highest training accuracy (97.07%), validation accuracy (95.11%), and the lowest training loss (2.93%) and validation loss (4.89%), demonstrating superior performance and generalization. The Xception model followed with competitive results, attaining a training accuracy of 83.57%, validation accuracy of 83.06%, training loss of 17.43%, and validation loss of 17.86%. ResNet50 showed strong performance with training and validation accuracies of 88.48% and 88.25%, respectively, and relatively low training (11.2%) and validation losses (10.41%). In contrast, models like InceptionV3, VGG16, and CNN exhibited significantly lower accuracies and higher losses. Both InceptionV3 and VGG16 had the same validation accuracy of 22.95%, while their training losses were 11.2% and 49.94%, respectively. CNN showed the poorest results, with a training accuracy of 23.93% and validation accuracy of 23.5%, alongside the highest training (50.29%) and validation losses (49.83%). This table underscores the exceptional performance of the Xception_Custom_Model compared to other models, while emphasizing the need for optimization in simpler architectures like CNN and VGG16.

Overall, Xception_Customize_Model emerged as the top-performing model with consistently high accuracy and low training loss, demonstrating its ability to generalize well with minimal overfitting. While Xception and ResNet50 showed competitive results, the former exhibited fluctuations in validation performance, and the latter required more epochs to stabilize. Models like InceptionV3, VGG16, and CNN performed comparatively lower, indicating the need for further tuning and optimization to match the advanced architectures. This analysis highlights the superior performance and reliability of customized and advanced models like Xception_Customize_Model in achieving state-of-the-art results.

Figure 4.17 Classification Results for Driver Activities

Figure 4.17 provides a visual summary of the classification results for various driver activities, highlighting both accurate and misclassified predictions. Each sub-image displays the actual class ("Real Class") and the predicted class ("Predicted Class") along with confidence percentages. Most images are correctly classified in the "Other activities" category with a confidence level of 100%. However, some misclassifications are evident, including predictions for "Turning" at 98.52% confidence and "Drive Safe" at 96.23%, and 89.19% confidence. These diverse examples demonstrate the model's effectiveness in accurately identifying activities while also exposing areas where it struggles to differentiate between similar classes. This visualization underscores the need for further model refinement to enhance classification performance.

4.3.2 Simulation Result

With the Driver Behavior Detection model, a web application can be built using Python libraries or existing industry-standard frameworks. The web application is implemented using Python, leveraging the Flask framework to create an interactive web-based interface.

- **Model Loading**: The application begins by loading the pre-trained model from a saved .h5 file. This model has been meticulously trained and optimized to achieve high accuracy in the classification of various driver behaviors. The pre-trained model allows for quick and reliable predictions by utilizing the extensive training it has undergone.
- **Image Preprocessing**: When a user uploads an image, it undergoes a series of preprocessing steps to ensure compatibility with the model's input requirements. The image is resized to 224x224 pixels to match the expected input size. Additionally, pixel values are normalized to the range [0, 1], aiding in achieving consistent and accurate predictions. The preprocessing is performed using Keras and Pillow libraries for Python.

- **Behavior Classification**: After preprocessing, the image is passed to the model for classification. The model processes the image and outputs confidence scores for various driver behaviors. These scores indicate the model's certainty regarding the presence of each behavior, providing a ranked list of potential matches.
- **Results Formatting**: The classification results are formatted into an interactive web structure. This structure clearly displays the detected driver behavior alongside its confidence level. The use of HTML and CSS ensures that the results are visually appealing and user-friendly.
- **User Interface**: A user-friendly interface is created using Flask, along with HTML, CSS, and JavaScript. The interface allows users to upload an image of a driver and view the predicted behavior with confidence levels. The intuitive design ensures users can interact with the application easily and understand the results provided by the model.

Web Interface:

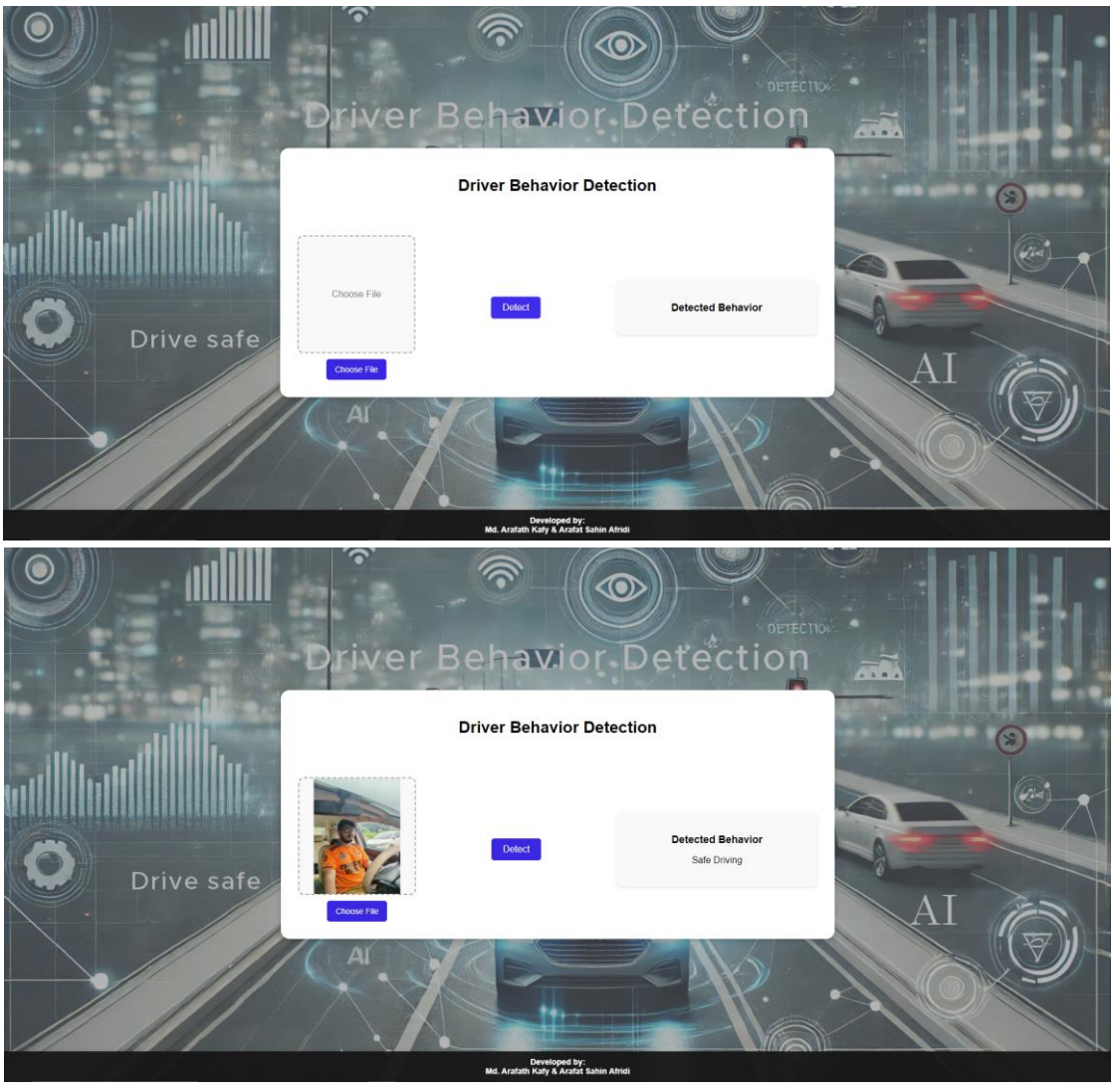

Figure 4.18 Web Interface and UI

4.4 Summary

This study evaluates the performance of five deep learning models—CNN, VGG16, ResNet50, InceptionV3, and Xception—on training and validation tasks. Xception demonstrated the best performance, achieving the highest training accuracy of 96.35% and validation accuracy of 95.36%, with the lowest training and validation losses of 3.43% and 4.48%, respectively. ResNet50 also performed strongly, achieving training and validation accuracies of 88.48% and 88.25%, with low losses. In contrast, CNN, VGG16, and InceptionV3 exhibited limited learning, with an accuracy of around 23% and validation losses near 50%, highlighting potential underfitting. While Xception and ResNet50 excelled in both optimization and generalization, misclassifications in some driver activities revealed opportunities for further improvement. This study underscores the importance of advanced architectures like Xception and ResNet50 for superior deep learning performance.

Chapter 5

Engineering Standards and Design Challenges

5.1 Compliance with the Standards

This section outlines the engineering standards adhered to during the development of this project. Each standard is discussed with its alternatives, pros, and cons, as well as the rationale for selection.

5.1.1 Software Standards

The software development process followed the IEEE 12207 standard for software lifecycle processes. This standard ensures a systematic and disciplined approach to software engineering. Key practices included:

- Requirements analysis.
- Software design and development.
- Testing and validation.

Alternatives Considered:

6. ISO/IEC 25010 (System and Software Quality Models): Focused on software quality.
7. Agile Manifesto: Iterative and flexible development.

Rationale for Selection: IEEE 12207 was chosen for its comprehensive lifecycle coverage, ensuring rigorous quality assurance practices essential for real-time applications like driver behavior detection.

5.1.2 Hardware Standards

The project adhered to ISO/IEC 14764 for hardware maintenance and upgrade processes. This ensured that the hardware used in data collection and system integration was reliable and scalable.

Alternatives Considered:

- MIL-STD-810: For ruggedized equipment.

- ISO 9241-210: Focused on user ergonomics.

Rationale for Selection: ISO/IEC 14764 was selected as it aligned with the project's requirements for flexible yet robust hardware support without overly focusing on military-grade or ergonomic constraints.

5.1.3 Communication Standards

TLS (Transport Layer Security)

- Alternatives: HTTPS, SSH.
- Rationale: Widely supported encryption standard, ensuring secure AI-driven data communication.

IEEE 802.11 (Wi-Fi Protocols)

- Alternatives: Ethernet, 5G.
- Rationale: Provides reliable wireless communication, aligning with existing educational infrastructure.

5.2 Impact on Society, Environment and Sustainability

5.2.1 Impact on Life

The research project "Analyzing the Efficacy of Convolutional Neural Network Architectures for Real-time Driver Behavior Detection" holds significant potential to enhance road safety and public health. By addressing critical challenges in driver monitoring, this study contributes to reducing accidents and saving lives.

- **Improved Road Safety:** By accurately detecting and classifying driver behaviors in real-time, the system enables timely warnings, helping prevent accidents caused by distractions or unsafe actions. This has a direct impact on protecting the lives of drivers, passengers, and pedestrians.
- **Enhanced Public Awareness:** The research fosters awareness of unsafe driving practices, encouraging individuals to adopt safer behaviors, thereby improving overall traffic discipline and safety.
- **Support for Policy Implementation:** The system provides reliable, data-driven insights into driver behavior, supporting the development and enforcement of road safety regulations, thus contributing to safer driving environments.

5.2.2 Impact on Society & Environment

The project's societal and environmental implications extend to enhancing public safety, minimizing accident-related damages, and promoting sustainable

transportation practices.

- **Social Impact:** By reducing accidents, this research alleviates the economic and emotional toll on individuals and families. Additionally, it promotes a culture of responsibility and accountability among drivers, positively influencing societal attitudes toward road safety.
- **Environmental Benefits:** Fewer accidents lead to reduced traffic congestion and fuel wastage, lowering vehicle emissions and contributing to a cleaner environment. Improved traffic flow also decreases the environmental footprint of urban transportation systems.
- **Technological Empowerment:** The integration of advanced machine learning tools with real-time applications empowers policymakers, automotive industries, and individuals to leverage innovative technologies for improved safety measures.

5.2.3 Ethical Aspects

Ethical considerations are central to our research on Analyzing the Efficacy of Convolutional Neural Network Architectures for real-time Driver Behavior Detection: An In-depth Study of VGG16, ResNet50, InceptionV3, and Xception. The study ensures responsible data use, respect for individual privacy, and transparency in methodologies. All driver behavior data is collected and utilized in accordance with ethical guidelines, safeguarding the privacy and rights of participants. The research strictly avoids any intrusive or harmful methods during data collection. Transparency and accountability are prioritized throughout the project, with clear documentation of methodologies, model development, and data usage. This commitment ensures that the outcomes of the study are both ethically sound and beneficial to society, contributing to safer driving practices and improved road safety.

5.2.4 Sustainability Plan

The sustainability plan for our research, Analyzing the Efficacy of Convolutional Neural Network Architectures for real-time Driver Behavior Detection: An In-depth Study of VGG16, ResNet50, InceptionV3, and Xception, incorporates several key strategies to ensure the long-term effectiveness and relevance of its outcomes. With frequent updates to account for changing driving behaviors, distractions, and behavioral trends, this project creates a reliable system for gathering, organizing, and preserving real-time driver behavior data. The project ensures that the classification models continue to be accurate and applicable in a variety of driving scenarios by consistently growing and improving the dataset. In order to encourage cooperation and encourage broad adoption, the study also highlights the importance of involving stakeholders, including automakers, traffic safety regulators, and technology developers. Additionally, a system of

ongoing monitoring and evaluation will determine the project's impact and efficacy, allowing for iterative improvements and guaranteeing its long-term contributions to driver behavior monitoring and road safety.

5.3 Project Management and Financial Analysis

This project adopts an Agile methodology, which emphasizes iterative development and continuous feedback. The project is organized into multiple sprints, each focused on achieving specific deliverables, ensuring flexibility and adaptability as the project progresses.

5.3.1 Project Phases

- **Sprint 1 (Data Collection and Preprocessing):** The first sprint focuses on gathering and preparing the dataset for model training. This involves image collection, data cleaning, augmentation, and preprocessing to ensure high-quality input for the machine learning models.
- **Sprint 2 (Model Training and Initial Evaluation):** During this sprint, the machine learning model (e.g., CNN) will be trained using the prepared dataset. Initial evaluations of the model's performance will be conducted, followed by necessary adjustments to improve accuracy.
- **Sprint 3 (Web Application Development):** This sprint is dedicated to developing the web application. The focus will be on creating the user interface, backend infrastructure, and integrating the model into the web platform for real-time predictions.
- **Sprint 4 (Model Integration and Testing):** The machine learning model will be integrated into the web application. Comprehensive testing will be conducted to ensure seamless interaction between the model and the application, as well as to validate the accuracy and functionality of the system.
- **Sprint 5 (Final Testing and Deployment):** The final sprint involves extensive testing of the integrated system. Once any issues or bugs are resolved, the system will be deployed for live use.

5.3.2 Project Timeline

The step-by-step project work with their time line is mentioned below. Here the weeks are representing after ending one work another starts from

- **Phase 1: Research and Data Acquisition (4-8 weeks)**
  This phase includes gathering the dataset, understanding the problem domain, and ensuring the data quality for subsequent model development.
- **Phase 2: Machine Learning Model Development (8-12 weeks)**

The development of the machine learning model will be the main focus of this phase, involving model selection, training, fine-tuning, and performance evaluation.

- **Phase 3: Web Application Design and Development (12-16 weeks)**
  This phase will focus on building the front-end and back-end components of the web application. This will include integrating the trained machine learning model into the application.
- **Phase 4: Testing and Refinement (16-17 weeks)**
  Extensive testing will be conducted to ensure that the system works as expected. Any necessary refinements and bug fixes will be implemented during this phase.
- **Phase 5: Deployment and Launch (16-17 weeks)**
  The final system will be deployed for use. This phase includes final testing, user training, and the launch of the system.
- **Phase 6: Monitoring and Maintenance (Ongoing)**
  After deployment, the system will require ongoing monitoring to ensure stability and address any emerging issues. Maintenance and updates will be carried out as needed.

5.3.3 Financial analysis

A comprehensive financial analysis ensures that the project stays within budget and achieves its objectives efficiently. This section outlines the projected costs, funding sources, and financial management strategies for the project.

Table 5.1 Estimated Cost for Driver Behavior Detection Project

SN	Components	Estimated Cost (BDT)
01.	Software and Development Tools	1500-2000
02.	Data Collection	500-1000
03.	Data Processing	2000- 2500
04.	Documentation and Report Writing	500-1000
05.	Miscellaneous (e.g., online resources, tools)	600-1000
06.	Web Application Development	1000-2000
07.	Contingency (10% of total)	1000-1500
	Total Estimated Cost	7,100-11,000

5.4 Complex Engineering Problem

5.4.1 Complex Problem Solving and Knowledge Profile Mapping

Complex Problem Solving (EP1, EP2, EP3, EP4, EP7)

EP1: Depth of Knowledge

This project demonstrates advanced knowledge of convolutional neural networks (CNNs) such as VGG16, ResNet50, InceptionV3, and Xception. This includes expertise in:

- **Data Preprocessing:** Techniques like resizing, normalization, and augmentation.
- **Model Training:** Transfer learning and hyperparameter tuning for enhanced performance.
- **Performance Evaluation:** Metrics such as accuracy, precision, recall, and F1 score.

These skills align with K3 (Engineering Fundamentals) and K4 (Specialist Knowledge).

EP2: Range of Conflicting Requirements

The project effectively balances conflicting requirements, such as:

- **Accuracy:** Achieving high classification performance for driver behaviors.
- **Efficiency:** Ensuring real-time processing with minimal computational delay.

This trade-off was addressed through careful model selection and optimization, showcasing its strong design foundation (K5).

EP3: Depth of Analysis

Extensive analysis of the CNN architectures was conducted, including:

- Comparative evaluation of VGG16, ResNet50, InceptionV3, and Xception.
- Fine-tuning hyperparameters to achieve optimal performance.
- Cross-validation to ensure model robustness.

This depth of analysis highlights the project team's expertise in addressing complex engineering challenges.

EP7: Interdependence

The project achieves interdependence by:

- **Data Preprocessing:** Preparing images for input into CNN models.
- **Model Inference:** Real-time detection of driver behaviors.
- **Web Interface Integration:** (Future work) Facilitating user interaction with the system.

This ensures a seamless flow between components, contributing to a cohesive real-time system.

Table 5.2: Mapping with complex problem solving.

EP1 Dept of Knowledge	EP2 Range Of Conflicting Requirements	EP3 Depth of Analysis	EP4 Familiarity of Issues	EP5 Extent of ApplicableCodes	EP6 Extent Of Stake-holder Involvement	EP7 Interdependence
✓	✓	✓				✓

Mapping with Knowledge Profile for EP1

Table 5.3: Mapping with knowledge Profile.

K3 Engineering Fundamentals	K4 Specialist Knowledge	K5 Engineering Design	K6 Engineering Practice	K8 Research Literature
✓	✓	✓	✓	✓

Justifying Knowledge Profile Mapping (K3, K4, K5, K6, K8)

The knowledge profiles required for this project span engineering fundamentals, specialist knowledge, design principles, practical engineering practices, and research insights. Below is a detailed justification for each:

K3: Engineering Fundamentals
The project relies on engineering fundamentals to understand machine learning principles, convolutional neural networks, and mathematical concepts essential for optimizing model performance.

K4: Specialist Knowledge
Specialist knowledge is demonstrated in the selection and fine-tuning of CNN architectures such as VGG16, ResNet50, InceptionV3, and Xception. Expertise in advanced optimization techniques is a key factor in the success of the project.

K5: Engineering Design
The design focuses on creating scalable and efficient models for real-time driver behavior detection. This includes balancing computational efficiency and classification accuracy, ensuring a robust design foundation.

K6: Engineering Practice
Practical engineering skills were applied in implementing and validating CNN architectures. Real-time testing and performance evaluations ensured readiness for deployment.

K8: Research Literature
Insights from existing research in driver behavior detection and CNN architectures informed model selection and implementation strategies. This included evaluating best practices and addressing identified gaps in prior works.

5.4.2 Engineering Activities

Complex Engineering Activities Mapping (EA1, EA4, EA5)

EA1: Range of Resources
The development of this project utilized a variety of resources, including advanced tools for machine learning and image preprocessing. Key resources include:

- TensorFlow and Keras: For building and training CNN architectures (VGG16, ResNet50, InceptionV3, Xception).
- OpenCV: For image preprocessing (e.g., resizing, augmentation).
- Python Programming Language: For scripting and implementing algorithms.
- Hardware Resources: NVIDIA GPUs to facilitate high-performance computations.

These resources showcase the ability to leverage diverse engineering tools effectively, highlighting the technical foundation of the project.

EA4: Consequences for Society and Environment
The project aims to enhance road safety by detecting driver distractions and potentially reducing traffic accidents. Socially, it promotes responsible driving habits through early intervention and monitoring systems. Environmentally, the system minimizes the need for physical hardware installations by integrating software solutions, reducing resource consumption and waste.

EA5: Familiarity
The project builds on lessons learned from existing works in driver behavior detection but tailors them to a unique dataset. Familiarity with established CNN models and preprocessing techniques ensured efficient adaptation to project-specific needs. Previous studies provided insights into performance benchmarks, guiding the selection and optimization of CNN architectures.

Rationale for Engineering Activities
EA1: Range of Resources
Leveraging TensorFlow, Keras, and OpenCV streamlined the development process by providing robust libraries for machine learning and image processing. The use of advanced GPUs ensured faster computation and efficient model training.

EA4: Consequences for Society and Environment
By detecting driver distractions in real time, the system can help reduce

accidents, making roads safer. Additionally, the use of software-based solutions limits environmental impacts compared to physical infrastructure-heavy systems.

EA5: Familiarity

Reviewing existing research in driver behavior detection and CNN model benchmarks informed the project's implementation strategies, ensuring alignment with best practices.

Table 5.4: Mapping with complex engineering activities.

EA1 Range of re-sources	EA2 Level of Interaction	EA3 Innovation	EA4 Consequences for society and environment	EA5 Familiarity
✓	✓	✓	✓	✓

5.5 Summary

The project, "Analyzing the Efficacy of Convolutional Neural Network Architectures for Real-Time Driver Behavior Detection," integrates complex engineering principles and design challenges to ensure a robust and effective system. By leveraging advanced CNN models, including VGG16, ResNet50, InceptionV3, and Xception, the project addresses critical challenges such as balancing accuracy and computational efficiency, achieving real-time performance, and adhering to ethical and sustainability standards.

The implementation complies with industry standards, ensuring the system's scalability, accessibility, and reliability. The project evaluates the societal and environmental impacts of its design, focusing on reducing road accidents and improving driver accountability. Ethical considerations, such as user data protection and fairness in AI, have been integral to the project's design. The sustainability plan prioritizes ongoing model refinement, dataset updates, and energy-efficient practices to ensure the system remains relevant and impactful over time.

In conclusion, this chapter highlights the project's ability to solve complex engineering problems through a collaborative, standards-compliant, and ethical approach. The system not only addresses technical challenges but also contributes to societal betterment by fostering safer roads and sustainable AI-driven solutions.

Chapter 6

Conclusion

This chapter provides an overview of the study's results, points out the shortcomings of the suggested system, and describes upcoming initiatives to expand its functionality and applicability.

6.1 Summary

This study explored the effectiveness of advanced Convolutional Neural Network (CNN) architectures—VGG16, ResNet50, InceptionV3, and Xception—in detecting driver behaviors in real-time applications. The research project entailed gathering a wide range of data that captured five different driving behaviors: turning, texting, talking on the phone, and safe driving, among others. Data preprocessing, model training, performance assessment, and the creation of a conceptual web application interface for real-time analysis were all part of the methodology.

The results of the study showed that CNN-based models could accurately classify driver behaviors, providing a viable way to increase traffic safety. The study shed light on these architectures' advantages and disadvantages in terms of classification accuracy and computational efficiency by contrasting how well they performed.

6.2 Limitation

- **Computational Resources:** High-performance GPUs and other computational infrastructure are needed to train sophisticated CNN models, including VGG16, ResNet50, InceptionV3, and Xception, on extensive driver behavior datasets. The overall project timeline may be impacted by restricted access to these resources, which can also drastically slow down the model training process and restrict the ability to experiment with hyperparameter tuning and model optimization.

- **Diversity and Quality of the Dataset:** The quality and diversity of the dataset have a significant impact on how well the driver behavior detection system works. Even though the dataset used in this study includes a variety of behaviors, it still falls short in capturing a variety of real-world situations, including different lighting conditions, camera angles, and driver demographics. These restrictions might have an impact on how well the model generalizes in real-world scenarios.

- **Limited Integration:** The current system lacks full integration into a real-time monitoring setup or deployment-ready framework, such as an application that can be directly used in vehicles.

6.3 Future Work

1. **Data Expansion and Augmentation:** The model's generalization will be greatly enhanced by gathering a larger and more varied dataset under a range of real-world circumstances. To better manage edge cases and uncommon behaviors, more data preprocessing methods can be investigated.
2. **Incorporation of Advanced Features:** To improve the system's functionality, features like predictive analytics for behavioral trends and multi-modal data integration—such as merging sensor or GPS data with image data—will be investigated. This may make it possible to implement more thorough monitoring and accident-prevention techniques.
3. **Industry Collaboration:** By collaborating with the fleet management and automotive sectors, it will be possible to test the system in practical settings, get input, and improve the solution before launching it on the market.

References

[1] X. Cui, X. Li, X. Zheng, and Y. Ren, "Driving Behavior Primitive Classification Using CNN-Based Fusion Models," IEEE Access, vol. 12, pp. 56344-56355, 2024, doi: 10.1109/ACCESS.2024.3391170.

[2] M. S. Islam, M. A. T. Rony, M. Safran, S. Alfarhood, and D. Che, "Elevating Driver Behavior Understanding With RKnD: A Novel Probabilistic Feature Engineering Approach," IEEE Access, vol. 12, pp. 65780-65798, 2024, doi: 10.1109/ACCESS.2024.3397725.

[3] F. Qu, N. Dang, B. Furht, et al., "Comprehensive study of driver behavior monitoring systems using computer vision and machine learning techniques," Journal of Big Data, vol. 11, no. 32, 2024, doi: 10.1186/s40537-024-00890-0.

[4] Y. Ma, Z. Xie, S. Chen, F. Qiao, and Z. Li, "Real-time detection of abnormal driving behavior based on long short-term memory network and regression residuals," Transportation Research Part C: Emerging Technologies, vol. 146, 2023, Art. no. 103983.

[5] R. Sethuraman, S. Sellappan, J. Shunmugiah, N. Subbiah, V. Govindarajan, and S. Neelagandan, "An optimized AdaBoost multi-class support vector machine for driver behavior monitoring in the advanced driver assistance systems," Expert Systems with Applications, vol. 212, 2023, Art. no. 118618.

[6] Ayad and M. E. Abdulmunim, "Detecting abnormal driving behavior using modified DenseNet," Iraqi Journal for Computer Science and Mathematics, vol. 4, no. 3, pp. 48-65, 2023.

[7] S. Bouhsissin, N. Sael, and F. Benabbou, "Driver Behavior Classification: A Systematic Literature Review," IEEE Access, vol. 11, pp. 14128-14153, 2023, doi: 10.1109/ACCESS.2023.3243865.

[8] Raza et al., "Preventing Road Accidents Through Early Detection of Driver Behavior Using Smartphone Motion Sensor Data: An Ensemble Feature Engineering Approach," IEEE Access, vol. 11, pp. 138457-138471, 2023, doi: 10.1109/ACCESS.2023.3340304.

[9] P. Khosravinia, T. Perumal, and J. Zarrin, "Enhancing Road Safety Through Accurate Detection of Hazardous Driving Behaviors With Graph Convolutional Recurrent Networks," IEEE Access, vol. 11, pp. 52983-52995, 2023, doi: 10.1109/ACCESS.2023.3280473.

[10] Y. Jeon, B. Kim, and Y. Baek, "Ensemble CNN to detect drowsy driving with in-vehicle sensor data," Sensors, vol. 21, no. 7, 2021, Art. no. 2372, doi: 10.3390/s21072372.

[11] Zhang, R. Li, W. Kim, D. Yoon, and P. Patras, "Driver behavior recognition via interwoven deep convolutional neural nets with multi-stream inputs," IEEE Access, vol. 8, pp. 191138-191151, 2020, doi: 10.1109/ACCESS.2020.3032882.

[12] M. Dua, Shakshi, R. Singla, et al., "Deep CNN models-based ensemble approach to driver drowsiness detection," Neural Computing & Applications, vol. 33, pp. 3155–3168, 2021, doi: 10.1007/s00521-020-05209-7.

[13] W. Xiao, H. Liu, Z. Ma, and W. Chen, "Attention-based deep neural network for driver behavior recognition," Future Generation Computer Systems, vol. 132, pp. 152-161, 2022.

[14] S. K. S. Al-doorı, Y. S. Taspınar, and M. Koklu, "Distracted driving detection with machine learning methods by CNN based feature extraction," International Journal of Applied Mathematics Electronics and Computers, vol. 9, no. 4, pp. 116-121, 2021.

[15] Y. Jeon, B. Kim, and Y. Baek, "Ensemble CNN to detect drowsy driving with in-vehicle sensor data," Sensors, vol. 21, no. 7, 2021, Art. no. 2372, doi: 10.3390/s21072372.

[16] P. Khosravinia, T. Perumal, and J. Zarrin, "Enhancing Road Safety Through Accurate Detection of Hazardous Driving Behaviors With Graph Convolutional Recurrent Networks," IEEE Access, vol. 11, pp. 52983-52995, 2023, doi: 10.1109/ACCESS.2023.3280473.

[17] Y. Jeon, B. Kim, and Y. Baek, "Ensemble CNN to detect drowsy driving with in-vehicle sensor data," Sensors, vol. 21, no. 7, 2021, Art. no. 2372, doi: 10.3390/s21072372.

[18] Sahin Afridi, Arafat; Kafy, Arafath ; Nessa Moon, Ms. Nazmun; Shakil, Md. Shahriar (2024), “Multi-Class Driver Behavior Image Dataset ”, Mendeley Data, V1, doi: 10.17632/mzb4b6dff3.1.

Analyzing the Efficacy of Convolutional Neural Network Architectures for real- time Driver Behavior Detection: An In-depth Study of VGG16, ResNet50, Inception V3, and Xception

ORIGINALITY REPORT

13%	7%	8%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Daffodil International University Student Paper	1%
2	V. Sharmila, S. Kannadhasan, A. Rajiv Kannan, P. Sivakumar, V. Vennila. "Challenges in Information, Communication and Computing Technology", CRC Press, 2024 Publication	1%
3	doaj.org Internet Source	1%
4	www.techtarget.com Internet Source	1%
5	Submitted to United International University Student Paper	<1%
6	Sajjan Singh, Sarabpreet Kaur. "Latest Trends in Engineering and Technology - AICTE Sponsored 2nd International Conference on 13th - 14th July, 2023", CRC Press, 2024 Publication	<1%