Faculty of Computer Science and Information Technology

Machine Learning-Based Identification of Stock Car Models via Car Sound Analysis

Isaiah Ho Chi Ann

Bachelor of Computer Science with Honours (Information System) 2025

UNIVERSITI MALAYSIA SARAWAK

THESIS STATUS ENDORSEMENT FORM

TITLE MACHINE LEARNING-BASED IDENTIFICATION OF STOCK CAR MODELS VIA CAR SOUND ANALYSIS

ACADEMIC SESSION: 2024/2025-2

ISAIAH HO CHI ANN

(CAPITAL LETTERS)

hereby agree that this Thesis* shall be kept at the Centre for Academic Information Services, Universiti Malaysia Sarawak, subject to the following terms and conditions:

1. The Thesis is solely owned by Universiti Malaysia Sarawak
2. The Centre for Academic Information Services is given full rights to produce copies for educational purposes only
3. The Centre for Academic Information Services is given full rights to do digitization in order to develop local content database
4. The Centre for Academic Information Services is given full rights to produce copies of this Thesis as part of its exchange item program between Higher Learning Institutions [or for the purpose of interlibrary loan between HLI]
5. ** Please tick (√)

☐	CONFIDENTIAL	(Contains classified information bounded by the OFFICIAL SECRETS ACT 1972)
☐	RESTRICTED	(Contains restricted information as dictated by the body or organization where the research was conducted)
☑	UNRESTRICTED	

(AUTHOR'S SIGNATURE)

Validated by

(SUPERVISOR'S SIGNATURE)

Permanent Address

Block 6 Unit 2, Taman Uni
Academia, Jalan Uni Academia
94300 Kota Samarahan, Sarawak

Date: 19 June 2025

Date: _____

Note * Thesis refers to PhD, Master, and Bachelor Degree
** For Confidential or Restricted materials, please attach relevant documents from relevant organizations / authorities

MACHINE LEARNING-BASED IDENTIFICATION OF STOCK CAR MODELS VIA CAR SOUND ANALYSIS

ISAIAH HO CHI ANN

This project is submitted in partial fulfilment of the requirements for the degree of Bachelor of Computer Science with Honours (Information System)

Faculty of Computer Science and Information Technology

UNIVERSITI MALAYSIA SARAWAK 2025

IDENTIFIKASI MODEL STOK KERETA BERDASARKAN PEMBELAJARAN MESIN MELALUI ANALISIS BUNYI KERETA

ISAIAH HO CHI ANN

Projek ini merupakan salah satu keperluan untuk Ijazah Sarjana Muda Sains Komputer dengan Kepujian (Sistem Maklumat)

Fakulti Sains Komputer dan Teknologi Maklumat

UNIVERSITI MALAYSIA SARAWAK 2025

DECLARATION

I hereby declare that this project is my original work. I have not copied from any other student's work or from any other sources except where due reference or acknowledgement is not made explicitly in the text, nor has any part had been written for me by another person.

…….

(ISAIAH HO CHI ANN) 17 JANUARY 2025

Matric No: 78061

ACKNOWLEDGEMENT

I would like to begin by expressing my sincere gratitude to my supervisor, Dr. Mohammad bin Hossin, for his invaluable assistance, guidance, advice, and tutoring throughout the course of this project. His expertise and support have played a crucial role in enabling me to complete my work. I would also like to extend my thanks to my examiner, Dr. Liew Siaw Hong, for her insightful feedback and suggestions that significantly enhanced the quality of my project.

I would also like to acknowledge the help provided by Prof. Dr Wang Yin Chai, the final year project coordinator, who has given me the necessary information and guidance to complete my final year project. In addition, I would like to thank all my friends and especially my course mates who are also under Dr. Mohammad bin Hossin, they have supported me and provided me with so much invaluable information needed to finish this project.

Lastly, I am deeply thankful to my parents and siblings for their unwavering love, support, and encouragement. Their constant belief in me has given me the strength and motivation to see this project through to the end. This achievement would not have been possible without their enduring support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The identification of vehicle models traditionally relies on visual recognition methods, which can be hindered by poor lighting, physical obstructions, or adverse weather conditions. This project introduces an innovative approach to vehicle model identification by leveraging machine learning techniques and car sound analysis. Each stock car model emits a unique acoustic signature influenced by its engine configuration and exhaust design. By analysing these sound patterns, the proposed system aims to classify stock car models with high accuracy. Through the utilisation of modern machine learning models, they have the power and intelligence to easily and most importantly quickly identify stock car models with verifiable accuracy. A comprehensive dataset of 233 stock car models was created using high-quality audio recordings from the game Forza Horizon 5. A robust pipeline was implemented for audio preprocessing and feature extraction, utilizing Mel-Frequency Cepstral Coefficients (MFCCs), spectral centroid, and other tonal features to represent the acoustic signatures numerically. Three machine learning models were developed and evaluated: Support Vector Machine (SVM), Random Forest (RF), and a Convolutional Neural Network (CNN). Evaluation using 5-fold cross-validation demonstrated high performance, with the SVM (95.72%) achieving the highest mean accuracy, followed by CNN (95.28%) and RF (93.02%). While the system works well on the controlled dataset, its accuracy drops when tested on real-world audio samples, highlighting the need for better generalization. This project successfully demonstrates the viability of acoustic-based classification in controlled settings and contributes to the advancement of non-visual vehicle identification techniques. The findings highlight key limitations and pave the way for future research.

ABSTRAK

Pengenalpastian model kenderaan secara tradisional bergantung kepada kaedah pengecaman visual, yang boleh terganggu oleh pencahayaan yang lemah, halangan fizikal, atau keadaan cuaca yang buruk. Projek ini memperkenalkan pendekatan baharu untuk mengenal pasti model kenderaan dengan memanfaatkan teknik pembelajaran mesin dan analisis bunyi kenderaan. Setiap model kereta stok mengeluarkan tanda akustik unik yang dipengaruhi oleh konfigurasi enjinnya dan reka bentuk ekzos. Dengan menganalisis corak bunyi ini, sistem yang dicadangkan bertujuan untuk mengklasifikasikan model kereta stok dengan ketepatan tinggi. Melalui penggunaan model pembelajaran mesin moden, teknologi ini mempunyai kuasa dan kepintaran untuk mengenal pasti model kereta stok dengan mudah dan, yang paling penting, dengan pantas serta ketepatan yang boleh disahkan. Satu set data komprehensif yang mengandungi 233 model kereta stok telah dicipta menggunakan rakaman audio berkualiti tinggi daripada permainan video Forza Horizon 5. Satu saluran paip yang mantap telah dilaksanakan untuk pra-pemprosesan audio dan pengekstrakan ciri, menggunakan Mel-Frequency Cepstral Coefficients (MFCC), sentroid spektrum, dan ciri-ciri nada lain untuk mewakili tandatangan akustik secara numerik. Tiga model pembelajaran mesin telah dibangunkan dan dinilai: Support Vector Machine (SVM), Random Forest (RF), dan Convolutional Neural Network (CNN). Penilaian menggunakan pengesahan silang 5-lipatan menunjukkan prestasi tinggi, dengan SVM (95.72%) mencapai purata ketepatan tertinggi, diikuti dengan CNN (95.28%) dan RF (93.02%). Walaupun sistem berfungsi dengan baik pada set data terkawal, ketepatannya menurun apabila diuji pada sampel audio dunia sebenar, menekankan keperluan untuk generalisasi yang lebih baik. Projek ini berjaya menunjukkan kebolehlaksanaan pengelasan berasaskan akustik dalam persekitaran terkawal

dan menyumbang kepada kemajuan teknik pengecaman kenderaan bukan visual. Penemuan ini menonjolkan batasan utama dan membuka jalan bagi penyelidikan masa depan.

Chapter 1: Introduction

1.1 Preliminaries

In the automotive industry, the identification of vehicle models has large been traditionally depended on two of our built-in senses, our sight and maybe audio when identified by pros. However, advancements in machine learning and audio processing allow for new, non-visual approaches to vehicle recognition. Recent advances in machine learning and signal processing have enabled sophisticated audio classification systems, as demonstrated in various domains such as speech recognition, music classification, and even bird species identification means it can technically be applied to in the automotive field too (Sprengel et al., 2016). Each stock car model possesses a unique acoustic signature primarily determined by its engine configuration, exhaust system design, and mechanical characteristics. By analysing these sounds, machine learning models can classify car models effectively, providing a reliable method for vehicle identification that transcends the limitations of visual recognition (Wieczorkowska et al., 2018). Car enthusiasts and professionals alive have for many decades traditionally used this distinct combination of exhaust and engine sounds to identify vehicles, demonstrating the feasibility of acoustic-based vehicle identification. Acoustic-based vehicle identification offers a promising alternative, as the unique sound signatures produced by different car models can provide valuable information. This project aims to leverage machine learning algorithms, specifically focusing on audio classification techniques from previously known techniques, to develop a system that can identify stock car models based solely on car sound analysis.

1.2 Problem Statement

Current vehicle identification methods rely heavily on visual recognition such as license plate recognition, car model detection or physical inspection of vehicle features like make, model, and trim badges. While these methods have proven effective in controlled environments, they face significant challenges under less-than-ideal conditions. Poor lighting, common in underground parking garages or during nighttime, can hinder the performance of visual systems. Similarly, physical obstacles like parked vehicles, pedestrians, or debris can obscure the view, rendering these systems ineffective. Weather conditions such as fog, heavy rain, or snow further limit visibility, and even factors like camera angle limitations may prevent a clear and consistent view of critical vehicle identifiers.

In contrast, car sounds offer a unique and underexplored avenue for vehicle identification. Exhaust systems and engine configurations are highly differentiated for different car models, even among those from the same manufacturer. For instance, variations in engine displacement, turbocharging, cylinder count, and exhaust design contribute to unique acoustic signatures (Kozhisseri & Bikdash, 2009). Seasoned car enthusiasts and mechanics can often identify car models by their exhaust notes alone, recognizing patterns in engine tone, pitch, and frequency. However, this detection for both human and machine will only work provided that the vehicles are stock and unmodified.

There is a huge untapped potential for kickstarting non-visual identification method in the automotive field. Creating an automated system will open new avenues for vehicle identification. This research aims to address these limitations by developing a machine

learning-based system capable of accurately identifying stock car models through their car sounds (Wu, 2019).

1.3 Scope

This project aims to develop a machine learning-based system capable of identifying stock car models from their car sounds. The project will leverage audio signal processing and machine learning techniques to design a robust classification system. The scope of the project encompasses:

Included:

1. **Data Collection**: Gather a diverse dataset of car sounds from various stock car models through Forza Horizon 5. One of the key advantages of Forza Horizon 5 is its extensive selection of stock vehicles, paired with authentic in game sounds that have been recorded directly from real-world scenario while also being affordable and accessible to students. The dataset will focus on capturing audio samples under controlled conditions to ensure consistency and reliability.
    - Car sounds from different car manufacturers will be recorded
    - Standardized protocols will be used to ensure uniform data quality
2. **Feature Extraction**: Use signal processing techniques to extract relevant features from the audio data.
    - Extract frequency-domain features, such as Mel-Frequency Cepstral Coefficients (MFCCs), which are widely used in audio analysis

- Spectrograms will also be used, allowing us an easier detection of subtle differences between car models

3. **Model Training**: Train machine learning models to classify stock car models based on the extracted features.
    - Implement Support Vector Machines (SVMs), Random Forests
    - Implement Convolutional Neural Networks (CNNs) as a comparison between traditional machine learning models
4. **Model Evaluation**: Evaluate the performance of the model using validation techniques and refine it to improve accuracy.
    - Metric such as accuracy will be used to ascertain performance
    - Models will be continuously refined to achieve optimal accuracy and reliability

Excluded:

1. **Modifications**: Focus will be on stock car models to establish a baseline for accurate identification since modified or aftermarket engine and exhaust systems significantly alter the acoustic signature of a vehicle.
2. **Vehicle types**: No motorcycles. The focus will remain exclusively on cars to narrow the scope and ensure model specificity due to manpower and limited time.
3. **Noise Reduction**: Assume that the system will operate in environments without active noise reduction.

4. **Third Party Integration**: Integration with existing vehicle security systems. This project focuses on developing a functioning system with 3rd party integration being outside the scope.

1.4 Aims and Objectives

1. To collect a comprehensive dataset of car sounds from various stock car models

2. To develop effective audio processing and feature extraction methods

3. To design and implement machine learning models for classification

1.5 Significance of the Project

There are three main significances of the project: Knowledge Value, Economy Applications and Community Contributions.

1.5.1 Knowledge Value

This project aims to offer knowledge value by advancing audio-based vehicle identification by exploring the potential of machine learning techniques to accurately classify vehicles based on their unique acoustic signatures, introduce the novel application of machine learning in automotive technology by applying machine learning to the domain of automotive technology, this research contributes to the development of innovative

solutions for vehicle identification and security and lastly push for development of new technologies in audio identifications, thereby opening up many possibilities beyond vehicle identification.

1.5.2 Economy Applications

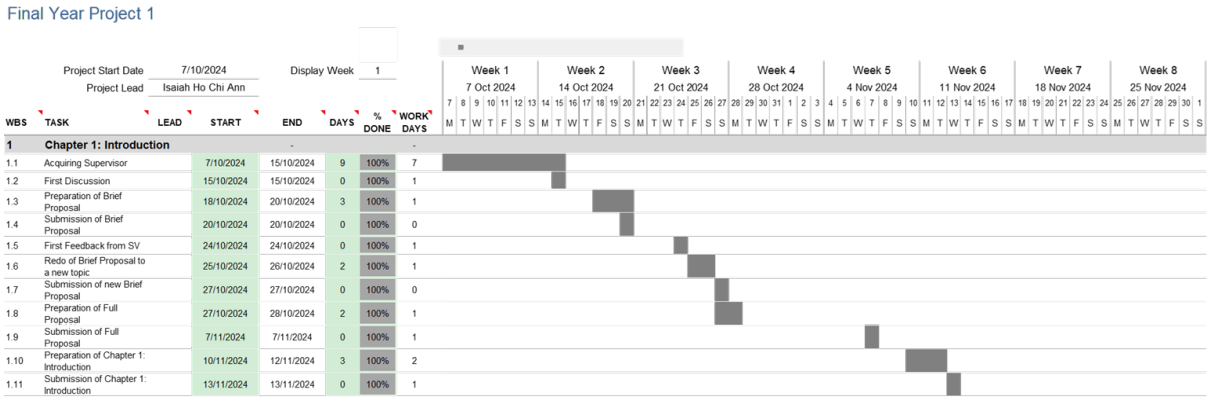
While for economy applications, this project could prove to be useful for usage in enhanced vehicle security systems which can be integrated into 3rd party systems although it's not currently part of the scope, be utilised in the automotive enthusiasts community which may find it useful, operated by traffic monitoring and management authorities who certainly will find the system as an added benefit and also employed in the vehicle management diagnostics to detect anomalies in engine performance or identify specific mechanical issues.

1.5.3 Community Contributions:

Last but not least, the findings may contribute to broader research in audio classification, advancing and thereby contributing more to speech recognition, audio information retrieval plus environmental sound analysis. The successful application of machine learning in this project might be shining torch to lead and inspire others to further research into various fields (Rong, 2016).

1.6 Project Schedule

The Gantt Chart was selected as the main way to track the progress of this Final year Project and was performed in Microsoft Excel.

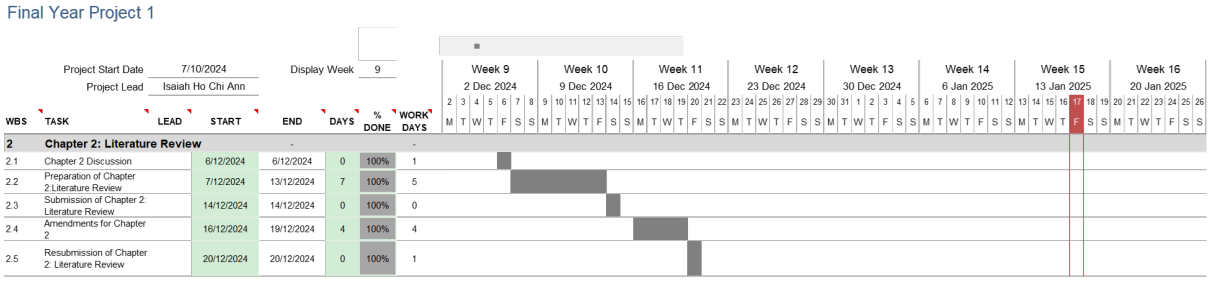


Final Year Project 1

Project Start Date 7/10/2024 | Display Week 1
Project Lead Isaiah Ho Chi Ann

WBS	TASK	LEAD	START	END	DAYS	% DONE	WORK DAYS
1	**Chapter 1: Introduction**			-			-
1.1	Acquiring Supervisor		7/10/2024	15/10/2024	9	100%	7
1.2	First Discussion		15/10/2024	15/10/2024	0	100%	1
1.3	Preparation of Brief Proposal		18/10/2024	20/10/2024	3	100%	1
1.4	Submission of Brief Proposal		20/10/2024	20/10/2024	0	100%	0
1.5	First Feedback from SV		24/10/2024	24/10/2024	0	100%	1
1.6	Redo of Brief Proposal to a new topic		25/10/2024	26/10/2024	2	100%	1
1.7	Submission of new Brief Proposal		27/10/2024	27/10/2024	0	100%	0
1.8	Preparation of Full Proposal		27/10/2024	28/10/2024	2	100%	1
1.9	Submission of Full Proposal		7/11/2024	7/11/2024	0	100%	1
1.10	Preparation of Chapter 1: Introduction		10/11/2024	12/11/2024	3	100%	2
1.11	Submission of Chapter 1: Introduction		13/11/2024	13/11/2024	0	100%	1

Figure 1.1: Gantt Chart Chapter 1



Final Year Project 1

Project Start Date 7/10/2024 | Display Week 9
Project Lead Isaiah Ho Chi Ann

WBS	TASK	LEAD	START	END	DAYS	% DONE	WORK DAYS
2	**Chapter 2: Literature Review**			-			-
2.1	Chapter 2 Discussion		6/12/2024	6/12/2024	0	100%	1
2.2	Preparation of Chapter 2 Literature Review		7/12/2024	13/12/2024	7	100%	5
2.3	Submission of Chapter 2: Literature Review		14/12/2024	14/12/2024	0	100%	0
2.4	Amendments for Chapter 2		16/12/2024	19/12/2024	4	100%	4
2.5	Resubmission of Chapter 2: Literature Review		20/12/2024	20/12/2024	0	100%	1

Figure 1.2: Gantt Chart Chapter 2

Final Year Project 1

Project Start Date 7/10/2024 Display Week 9
Project Lead Isaiah Ho Chi Ann

WBS	TASK	LEAD	START	END	DAYS	% DONE	WORK DAYS
3	**Chapter 3: Methodology**			-			-
3.1	Preparation of Chapter 3:Methodology		28/12/2024	4/1/2025	8	100%	5
3.2	Submission of Chapter 3		5/1/2025	5/1/2025	0	100%	0
3.3	Chapter 3 Discussion		9/1/2025	9/1/2025	0	100%	1
3.4	Amendments for Chapter 3		9/1/2025	12/1/2025	4	100%	2
3.5	Further Discussions		14/1/2025	14/1/2025	0	100%	1
3.6	Resubmission of Chapter 3: Methodology		17/1/2025	17/1/2025	0	100%	1

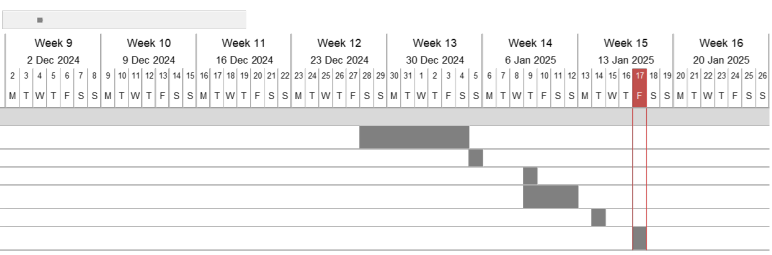


Figure 1.3: Gantt Chart Chapter 3

Final Year Project 1

Project Start Date 7/10/2024 Display Week 24
Project Lead Isaiah Ho Chi Ann

WBS	TASK	LEAD	START	END	DAYS	% DONE	WORK DAYS
4	**Chapter 4: Implementation**			-			-
4.1	Chapter 4 Discussion		20/3/2025	20/3/2025	1	100%	1
4.2	Preparation of Chapter 4:Implementation		24/3/2025	18/4/2025	26	100%	20
4.3	Submission of Chapter 4:Implementation		18/4/2025	18/4/2025	1	100%	1
4.4	Amendments for Chapter 4		19/4/2025	23/4/2025	5	100%	3
4.5	Resubmision of Chapter 4:Implementation		23/4/2025	23/4/2025	1	100%	1

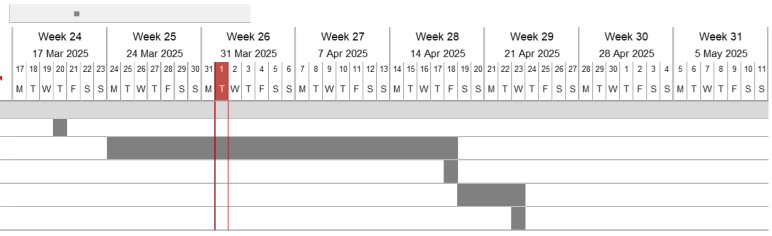


Figure 1.4: Gantt Chart Chapter 4

Final Year Project 1

Project Start Date 7/10/2024 Display Week 29
Project Lead Isaiah Ho Chi Ann

WBS	TASK	LEAD	START	END	DAYS	% DONE	WORK DAYS
5	**Chapter 5: Evaluation, testing and results**			-			-
5.1	Chapter 5 Discussion		24/4/2025	24/4/2025	1	100%	1
5.2	Preparation of Chapter 5: Evaluation, testing and results		26/4/2025	10/5/2025	15	100%	10
5.3	Submission of Chapter 5:Evaluation, testing and results		10/5/2025	10/5/2025	1	100%	0
5.4	Amendments for Chapter 5		11/5/2025	15/5/2025	5	100%	4
5.5	Resubmision of Chapter 5:Evaluation, testing and results		15/5/2025	15/5/2025	1	100%	1

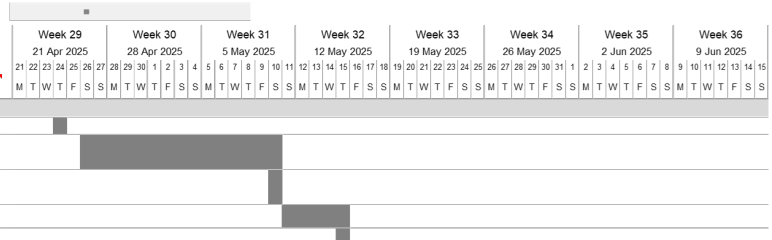


Figure 1.5: Gantt Chart Chapter 5

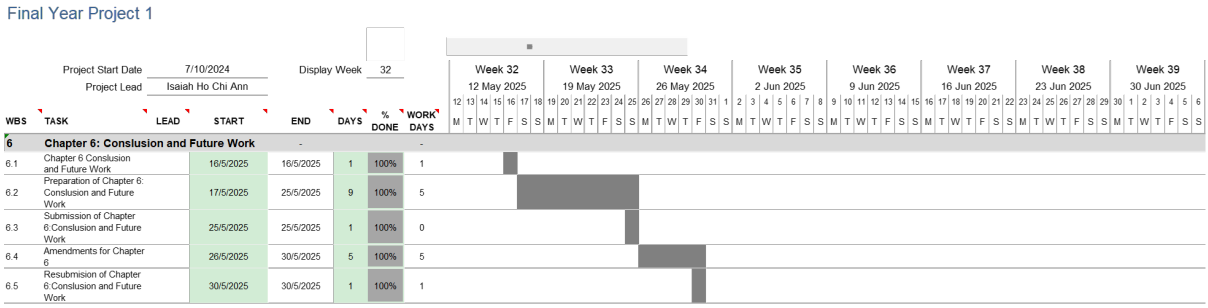


Figure 1.6: Gantt Chart Chapter 6

1.7 Chapter Outlines

This Final Year Project consists of six chapters that are organised as follows:

Chapter 1: Introduction

This chapter serves as the introduction of Machine Learning-Based Identification of Stock Car Models via Car Sound. This section includes – Introduction, Problem Statement, Scope, Aims and Objectives, Significance of the Project, Project Schedule, Expected Outcome and Chapter Outlines.

Chapter 2: Literature Review

This chapter will review similar pre-existing works of Machine Learning-Based Identification of Stock Car Models via Car Sound. The research of the recent works will be analysed and presented here.

Chapter 3: Methodology

This section delves into the methodologies and the proposed design plus development of the would-be web application of this Final Year Project.

Chapter 4: Implementation

This chapter focuses on the implementation of the proposed application. All the steps of implementation will be properly documented in this chapter.

Chapter 5: Evaluation, Testing and Result

This chapter will commence after Chapter 4 to make sure the developed application runs smoothly and fulfils all the requirements. Feedback will also be gathered to gauge the overall performance of the application.

Chapter 6: Conclusion and Future Works

This chapter establishes the conclusion of the Final Year Project and presents the result of the study. Furthermore, this chapter also lays out any future work that might be conducted.

Chapter 2: Literature Review

2.1 Stock Car Sounds Identification

Attempts at using recording stock car sounds and audio identification to determine specific models is a relatively new venture as of the latest developments in December 2024 (Yassin et al., 2023). A detailed search of the internet was performed and any related or similar attempts to this research was done on the 4th of December 2024. The results were a few similar articles and conference paper that have been selected to be used as references in this literature review section. The oldest article is dated from 1999 (Huadong Wu et al., 1999) while the rest are relatively modern and are from post 2000s and 2020s (Kozhisseri & Bikdash, 2009) (Yassin et al., 2023). Since no topics of research and methods are exactly similar to this Final Year Project, similar research such as classifying car type (light, medium, heavy) (George, Cyril, et al., 2013), car detection using only engine sound and others (Kubera et al., 2015) were selected. These articles/conference papers were deemed suitable to be used since they still focus on identifying cars using acoustic signatures. This research aims to investigate the feasibility/potential of using machine learning techniques to identify stock car models based on their unique acoustic signatures.

2.2 Machine Learning Algorithms

Machine Learning is now the forefront of transformative technology across many fields, allowing a created system to learn and make predictions or decisions without being explicitly programmed. It works by analysing patterns in data, ML algorithms excel at

handling complex and high-dimensional problems, including classification, regression, and clustering tasks. The following section provides the foundational concepts of Machine Learning and its application to audio analysis, particularly in the context of vehicle identification.

Machine Learning is a subset of artificial intelligence (AI) that focuses on designing algorithms capable of improving their performance through experience. This means algorithms can be developed that can learn from and make predictions based on data and most importantly do all this in an automated manner. The primary types of machine learning include:

- **Supervised Learning:** This involves training a model on labelled data, where the algorithm learns to map inputs to outputs based on examples. Some examples of this would be classification and regression tasks.
- **Unsupervised Learning:** Involves training a model with unlabelled data, whereby the ML model will attempt to identify patterns or groupings in the data themselves. Common examples of this are hierarchical clustering and K-Means techniques.

For this project, supervised learning is the primary focus, as it involves classifying stock car models based on labelled car sound data.

Machine learning has been instrumental in analysing audio data due to its ability to extract and leverage complex patterns. Audio classification tasks, such as speech recognition, music genre identification, and environmental sound detection, often rely on ML models to distinguish between subtle acoustic variations. Key to these applications is the preprocessing of raw audio into meaningful representations, such as spectrograms or Mel-Frequency

Cepstral Coefficients (MFCCs), which capture the essential features of the sound. Recent studies related pertaining to identifying vehicles with the help of acoustic signatures from (Bulatovic & Djukanovic, 2022; George, Cyril, et al., 2013, 2013; George, Mary, et al., 2013; Kozhisseri & Bikdash, 2009; Kubera et al., 2015; Wieczorkowska et al., 2018; Yassin et al., 2023), have successfully tested multiple supervised machine learning methods such as Artificial Neural Network, K-nearest Neighbours, Random Forest, Deep Learning and Support Vector Machines. All the methods list here have varying level of efficiency and accuracy based on the research done as mentioned before. The continued use of machine learning highlights its effectiveness and versatility in sound-based applications, which have already been successfully demonstrated in related fields such as music recognition, speech recognition, and bird species identification.

2.3 Machine Learning Model for Car Identification

This section will delve into three such machine learning algorithms: Support Vector Machine (SVM), Random Forest (RF) and Convolutional Neural Network (CNN). Each approach has its own differences when applied to car sound classification, and their specific uses will be examined in detail in the subsequent sections.

2.3.1 Support Vector Machines (SVM)

Support Vector Machine (SVM) is a powerful and versatile supervised machine learning algorithm used for classification, regression, and outlier detection tasks. It is particularly well-regarded for its performance in high-dimensional spaces, making it a

strong candidate for complex classification problems like audio analysis. The fundamental principle of SVM is to find an optimal decision boundary, known as a hyperplane, that separates data points of different classes with the maximum possible margin (Cortes & Vapnik, 1995).

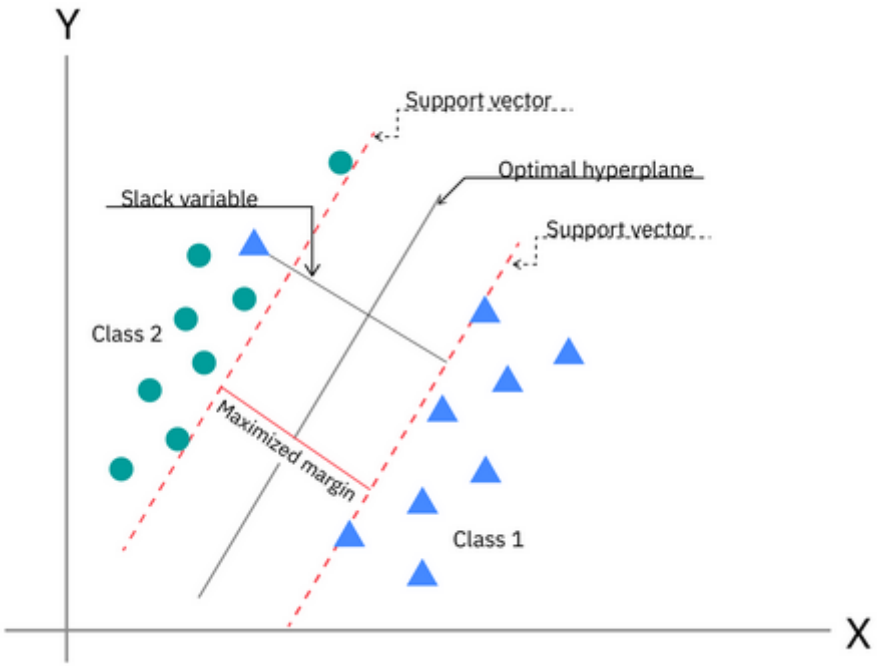


Figure 2.1: Representation of SVM

For a given set of training data, each marked as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other. The optimal hyperplane is the one that is farthest from the closest data points of any class. These closest points are called **support vectors**, and they are the critical elements that define the position and orientation of the hyperplane. By maximizing the

margin, the SVM enhances the classifier's ability to generalize to new, unseen data, thus reducing the probability of misclassification.

In cases where the data is not linearly separable, SVMs employ a technique known as the **kernel trick**. This involves using kernel functions (e.g., Polynomial, Sigmoid, or Radial Basis Function (RBF)) to map the input data into a higher-dimensional feature space where a linear separation becomes possible (Lu et al., 2003). The RBF kernel is a popular choice and was empirically observed by (Lu et al., 2003) to perform better than other kernels for audio classification. This ability to handle non-linear data makes SVMs highly effective for complex, real-world datasets where the relationships between features are not straightforward.

The effectiveness of SVMs in acoustic classification has been consistently demonstrated across various studies. (Lu et al., 2003) used SVMs for content-based audio classification and segmentation, comparing them against K-Nearest Neighbors (KNN) and Gaussian Mixture Models (GMM). Their findings showed that the SVM-based classifier was significantly more accurate, especially when dealing with complex audio signals. (Rai et al., 2016)also successfully utilized SVMs in conjunction with Mel-Frequency Cepstral Coefficients (MFCCs) to automatically classify bird species, achieving high accuracy on their dataset.

For the specific domain of vehicle sound analysis, SVMs have been a cornerstone for classification. Both (Wieczorkowska et al., 2018) and (Kubera et al., 2015) applied SVMs to classify different vehicle and engine types based on their acoustic signatures. Their work highlights the model's capability to discern between nuanced audio features.

2.3.2 Random Forest (RF)

Random Forest (RF) is a powerful and popular ensemble learning method used for both classification and regression tasks. Developed by Leo Breiman, the algorithm works by constructing a multitude of decision trees during training and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees (Breiman, 2001). This ensemble approach makes Random Forest robust against overfitting, which is a common problem with single decision trees, and it often yields higher accuracy.

The core idea behind Random Forest is to combine the predictions of several decision trees, each trained on a different subset of the data. The process involves two key techniques. The first one is Bootstrap Aggregating (Bagging) where from the original training dataset, multiple new training sets are created by random sampling with replacement. This means that some data points may be used several times in a single tree's training set, while others may not be used at all. Each tree in the forest is trained on one of these bootstrap samples. The second is Random Feature Selection referring to when splitting a node in a decision tree, Random Forest considers only a random subset of the available features. This prevents any single feature from dominating the decision-making process across all trees and ensures that the individual trees are decorrelated. The final classification is determined by a majority vote from all the trees in the forest.

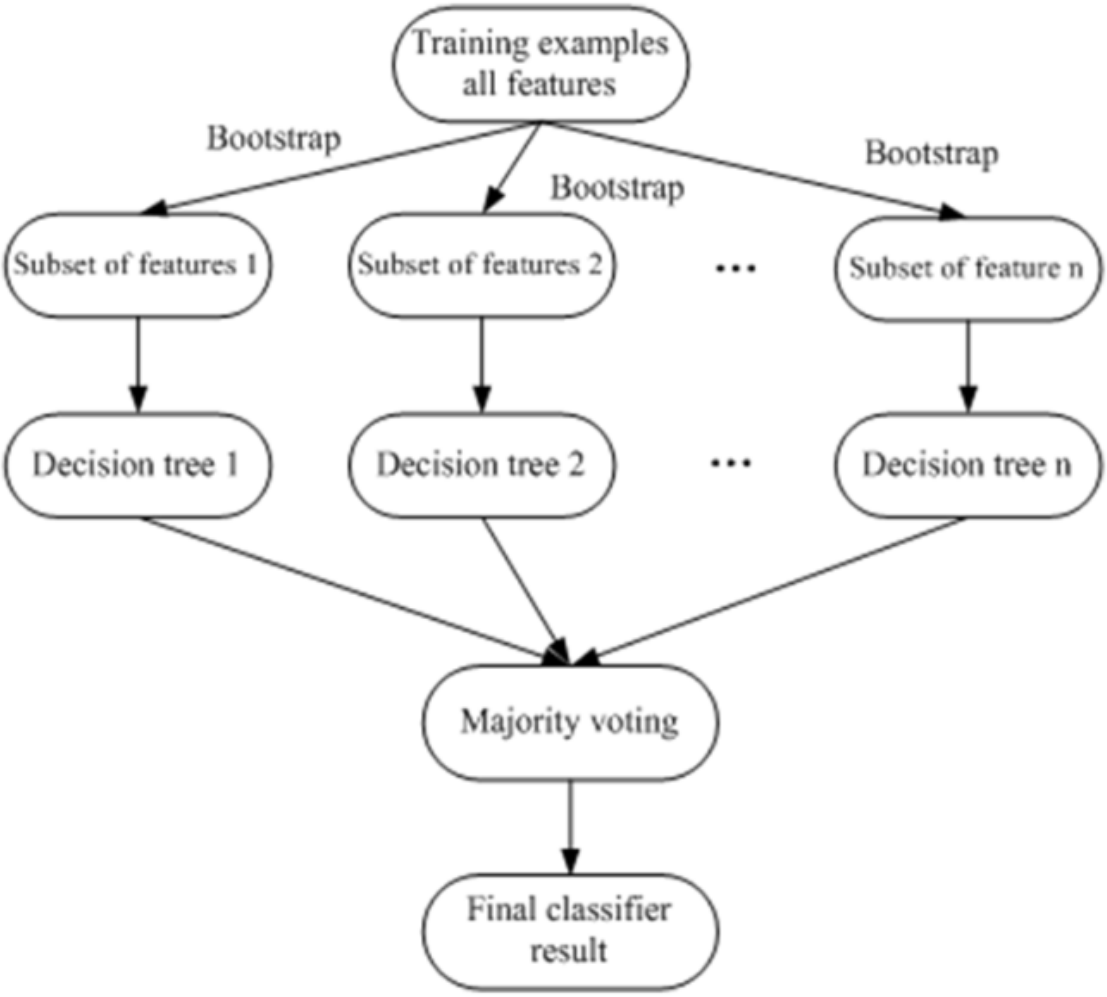


Figure 2.2: Sample Random Forest (Zhang & Lv, 2015)

Random Forest has proven to be highly effective in various audio classification tasks. (Grama & Rusu, 2017) utilized RF to classify audio signals for a wildlife intruder detection system, achieving an impressive overall correct classification rate of 99.25%. Their study highlighted that RF was an excellent choice for their multiclass problem with imbalanced data. Similarly, (Zhang & Lv, 2015) found that Random Forest outperformed other ensemble methods like Bagging and AdaBoost for environmental audio classification, noting its stability and strong performance even with smaller training sets.

They also leveraged RF's built-in capability to assess feature importance using the Gini Index to select the most relevant features for their model.

In a direct comparison for audio classification, (Ansari et al., 2021) evaluated the performance of RF against SVM. Their results showed that the performance of each model could vary depending on the data; for one subject with less training data, Random Forest was more accurate than SVM. This suggests that RF can be a very strong performer, particularly in scenarios with limited data. For vehicle sound analysis specifically, both Wieczorkowska et al. (2018) and Kubera et al. (2015) successfully applied Random Forest classifiers, alongside SVMs, to achieve robust vehicle and engine type classification.

2.3.3 Convolutional Neural Network (CNN)

A Convolutional Neural Network (CNN) is a class of deep learning models most commonly applied to analysing visual imagery. Inspired by the organization of the animal visual cortex, CNNs are designed to automatically and adaptively learn spatial hierarchies of features from input data. A typical CNN architecture consists of an input layer, multiple hidden layers (including convolutional layers, pooling layers, and fully connected layers), and an output layer (Zaman et al., 2023).

The key innovation of CNNs is the convolutional layer, which uses a set of learnable filters (or kernels) to detect specific features like edges, textures, and patterns. As the input data passes through the network, these filters become activated when they detect their corresponding feature. Subsequent layers build upon the patterns detected in previous

layers to learn more complex structures. This hierarchical feature learning makes CNNs exceptionally powerful for tasks where local patterns are important.

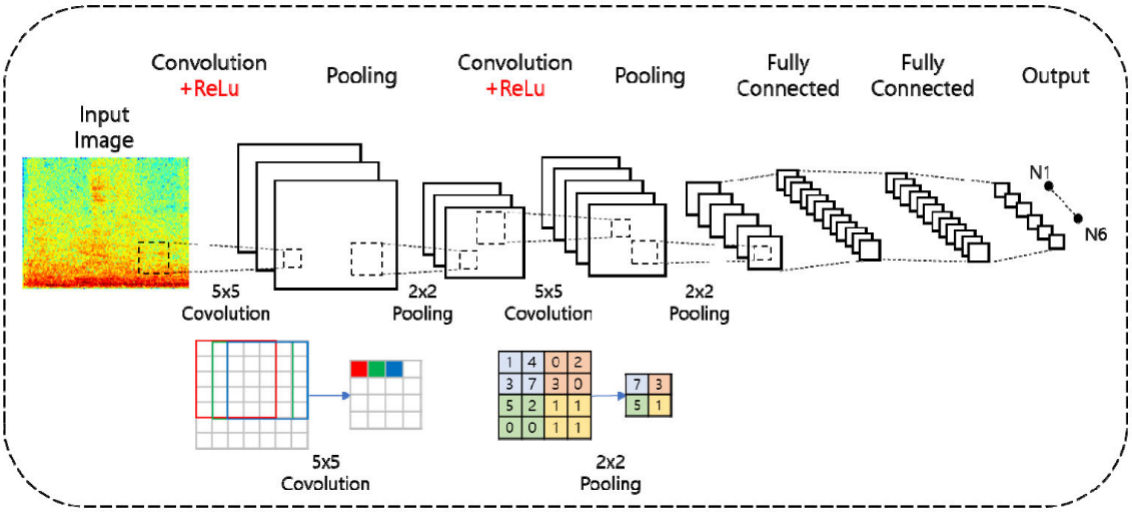


Figure 2.3: Sample CNN structure (Zaman et al., 2023)

For audio classification, the standard approach is to convert the one-dimensional (1D) raw audio waveform into a two-dimensional (2D) representation, most commonly a **spectrogram** or **Mel-spectrogram**. This 2D representation, which visualizes the spectrum of frequencies as they vary over time, can be treated as an "image" and fed into a CNN. This allows the network to learn spectro-temporal patterns that are characteristic of different sounds (Hershey et al., 2017; Palanisamy et al., 2020). While end-to-end models that operate directly on raw waveforms exist (Huang & Leanos, 2018), the spectrogram-based approach has become dominant due to its proven effectiveness.

The power of using standard CNN architectures designed for image classification (e.g., AlexNet, VGG, ResNet, DenseNet) for audio tasks has been thoroughly

demonstrated. (Hershey et al., 2017) showed that these architectures achieve excellent results on large-scale audio classification. Furthermore, (Palanisamy et al., 2020) argued that fine-tuning ImageNet-pretrained models on audio spectrograms can achieve state-of-the-art results, suggesting that the features learned from natural images are highly transferable to the "images" of sound. Early work in vehicle sound classification by (George, Cyril, et al., 2013) utilized simpler Artificial Neural Networks (ANNs), but the field has since progressed to adopt more complex and powerful deep learning models like CNNs. A comprehensive survey by (Zaman et al., 2023) confirms that CNNs are currently the most popular and effective deep learning architecture for a wide array of audio classification tasks.

2.4 Review on Past Research

As mentioned before, there aren't that many articles about or related with identifying vehicles using only acoustic signatures when compared to other usages of audio analysis such as music recognition and bird identification. There were roughly 10 or more articles and conference papers that were found to be in topic with this Final Year Project. Five articles/conference paper were selected after carefully perusing through the internet and selecting the best and most relevant to this Final Year Project.

The study by (George, Cyril, et al., 2013) explores the feasibility of leveraging vehicle sound signatures for detection and classification purposes using artificial neural networks (ANN), which closely aligns with the objectives of this Final Year Project. Their study explored the potential of using sound signatures for vehicle detection and classification,

serves as an important foundational reference by demonstrating the feasibility of acoustic-based vehicle identification.

The research by (George, Mary, et al., 2013) primarily continues the study of acoustic-based vehicle detection and classification has shown significant promise in addressing challenges where traditional visual methods are impractical. This study expanded the investigation by comparing the performance of ANNs and k-Nearest Neighbours (k-NN) algorithms for acoustic vehicle detection and classification. This research provided valuable insights into the accuracy of different machine learning methods for this task.

Moreover, the study by (Kozhisseri & Bikdash, 2009) delves into the intricate domain of acoustic vehicle identification exploring the potential of spectral features to distinguish between civilian vehicles using acoustic sensors. Their work provides a foundational framework for vehicle classification using spectral features extracted from acoustic sensors, emphasizing robust and precise identification under varied conditions. Spectral features, derived from the frequency domain representation of acoustic signals, offer a powerful approach to identifying distinct vehicle types. Their research aligns with the broader objective of this final year project by investigating the feasibility of leveraging sound signatures for vehicle classification, which align closely with this project's focus on distinguishing stock car models based on car sound analysis.

Furthermore, the research on acoustic vehicle classification presented by Yassin et al. (2023) explored the integration of Mel-Frequency Cepstral Coefficients (MFCCs) and Long Short-Term Memory (LSTM) neural networks to classify vehicles into broad categories (e.g., motorcycles, cars, trucks, and "no traffic"). Their methodology and findings offer valuable

insights directly relevant to this Final Year Project (FYP) by exploring the potential of Deep Learning models to accurately classify vehicles based on their acoustic signatures.

Finally, the study by Kubera et al. (2015) introduced a hierarchical classification system for distinguishing between various vehicle types using audio signals. This framework leveraged spectral and time-domain features, combined with multiple machine learning models, to classify vehicles into broad categories (cars, trucks, tractors) and further refine these into subclasses (e.g., small cars, vans, large trucks). The following study aligns closely with the current Final Year Project objectives of identifying stock car models via sound analysis.

2.5 Comparison and Review of Methodology and Key Findings

The comprehensive analysis of methodologies and key findings across the reviewed studies reveals intricate patterns, significant variations, and evolving approaches in acoustic vehicle identification research. This detailed examination spans data collection methods, feature extraction techniques, classification approaches, and their respective outcomes, providing crucial insights for future research directions in vehicle sound analysis.

2.5.1 Data Collection and Preprocessing

In terms of data collection methodologies, the studies exhibited notable diversity in their approaches while maintaining certain common elements. (George, Cyril, et al., 2013) conducted their recordings on two-lane undivided roads with moderate traffic, employing both video and stereo audio recordings through a Sony Handycam and Zoom H4n recorder.

This dual-recording approach provided complementary data streams for validation purposes. In contrast, (George, Mary, et al., 2013) focused on minimizing environmental interference by strategically placing roadside microphones along straight stretches of road, demonstrating an emphasis on signal quality over quantity. (Kozhisseri & Bikdash, 2009) took a markedly different approach by conducting controlled recordings of idling engines with open hoods, prioritizing the isolation of pure engine sounds without the contamination of road noise or environmental factors. This controlled approach, while limiting real-world applicability, provided valuable baseline data for understanding core engine acoustic signatures. (Yassin et al., 2023) expanded the environmental scope significantly by collecting data across multiple road types (one country road and three urban roads) under varying weather conditions (wet and dry), introducing environmental variability as a key consideration in their methodology. (Kubera et al., 2015) focused their recordings in a suburban area near Lublin, Poland, conducting hour-long sessions that captured a wide range of vehicle types and ambient conditions. The technical specifications across these studies showed some convergence, with 48 kHz emerging as the standard sampling rate, though bit depth varied from 16-bit to 24-bit depending on the study's specific requirements and equipment capabilities. This inclusion of real-world variability underscores the importance of designing systems resilient to environmental changes, enhancing model generalizability and robustness. To address potential inconsistencies in audio quality, advanced preprocessing steps like noise filtering and normalization were universally employed. Techniques such as smoothed logarithmic energy extraction and horn sound elimination were critical in isolating relevant acoustic events, as seen in studies by (Kubera et al., 2015) and (George, Mary, et al., 2013).

2.5.2 Feature Extraction

Feature extraction techniques demonstrated both commonality and innovation across the studies. Mel-Frequency Cepstral Coefficients (MFCCs) emerged as the de facto standard feature extraction method, with implementations varying in the number of coefficients used (typically 13, though some studies experimented with different configurations). The widespread adoption of MFCCs can be attributed to their proven effectiveness in capturing spectral characteristics that closely align with human auditory perception. Energy-based features formed another crucial category, with studies implementing various approaches including short-time energy, logarithmic energy, and smoothed logarithmic energy. (Kozhisseri & Bikdash, 2009) innovative use of tristimulus and M-stimulus responses borrowed from music theory represented a novel approach to feature extraction, demonstrating the potential for cross-disciplinary methodologies in vehicle sound analysis. Their method of grouping harmonics into three categories (T_1, T_2, and T_3) provided a unique perspective on engine sound characterization. (Kubera et al., 2015) expanded the feature set further by incorporating zero-crossing rate and detailed audio spectrum features (centroid, spread, and roll-off frequencies), demonstrating the value of combining multiple feature types for improved classification accuracy. However, several studies have highlighted the value of integrating advanced spectral features to complement MFCCs. (Kozhisseri & Bikdash, 2009) explored power spectral density (PSD) and tristimulus values, which proved particularly effective in distinguishing vehicles with similar engine configurations. PSD's ability to identify spectral peaks and tristimulus' representation of harmonic groupings provide additional layers of discrimination, enabling finer granularity in classification tasks. Other innovative approaches include the use of Mel-spectrograms, which visually represent

frequency components over time. Studies like (Kubera et al., 2015) leveraged these visual features to enhance CNN-based model performance, capturing both spatial and temporal patterns within audio signals. Complementary features such as spectral centroid, zero-crossing rate (ZCR), and harmonic feature analysis further enriched the feature sets, as evidenced in (Kubera et al., 2015).

2.5.3 Machine Learning Model and Classification

The development of classification approaches across these studies reflects the broader progression in machine learning applications. Traditional machine learning models, including Support Vector Machines (SVM), K-Nearest Neighbours (KNN), Random Forest (RF), and Decision Trees, were extensively evaluated across different studies. (George, Mary, et al., 2013) comparative analysis of ANN and KNN provided valuable insights into the relative strengths and limitations of these approaches, with ANN significantly outperforming KNN (70-73% versus ~50% accuracy). The introduction of more sophisticated neural network architectures, particularly in later studies, marked a significant advancement in classification capabilities. (Yassin et al., 2023) implementation of Long Short-Term Memory (LSTM) networks achieved impressive accuracy rates of 82-86.2%, demonstrating the potential of deep learning approaches in this domain. (Kubera et al., 2015) hierarchical classification strategy represented a particularly innovative approach, combining the strengths of both binary and multi-class classification methods to achieve superior results, with some specific categories reaching accuracy rates up to 99%.

The performance metrics across studies revealed critical insights into the factors influencing classification accuracy. Classification accuracy varied across studies, heavily influenced by the quality of preprocessing, feature selection, and model design. Early works, such as (George, Cyril, et al., 2013), achieved moderate accuracies of around 67%, primarily due to the limitations of simpler models and constrained feature sets. Data quality emerged as a paramount consideration, with controlled recordings consistently yielding superior results compared to real-world conditions. This was particularly evident (Kozhisseri & Bikdash, 2009) study, where the controlled environment of idling engines provided highly reliable classification results. The challenge of distinguishing between vehicles with similar engine configurations was particularly noted in (Kozhisseri & Bikdash, 2009) work, where overlapping harmonic patterns complicated classification tasks. The impact of environmental conditions was thoroughly documented by Yassin et al., whose wet versus dry road conditions demonstrated the importance of accounting for environmental variables in classification systems. Frame size analysis, particularly detailed in Kubera et al.'s work, revealed that longer frames (e.g., 330 ms) generally provided better frequency resolution and higher accuracy compared to shorter frames (15 ms), though at the cost of increased processing time.

A significant pattern emerged regarding the relationship between model complexity and computational requirements. Deep learning approaches, while offering superior performance in many cases, demanded substantial computational resources and larger training datasets. This was particularly evident in (Yassin et al., 2023) LSTM implementation, which required significant GPU resources (GTX 1080 Ti) and substantial RAM (64GB). This highlights an important practical consideration for real-world implementations, where

resource constraints might necessitate trade-offs between model complexity and performance.

2.5.4 Evolution of Methodologies in Vehicle Sound Analysis

The evolution of methodological sophistication across these studies reveals several important advancements. There are a clear movement toward more complex, hierarchical classification approaches that can better handle the nuanced differences between vehicle sounds. The increasing adoption of deep learning techniques, while promising, highlights the need for balanced approaches that consider both performance and practical implementation constraints. The emphasis on data quality and controlled collection methods underscores the importance of robust dataset development for future research.

The synthesis of findings across these studies provides valuable guidance for future research in acoustic vehicle identification, particularly in the context of stock car model identification through sound analysis. The success of hierarchical approaches suggests that future systems might benefit from multi-level classification strategies that can progressively refine vehicle identification. The demonstrated value of combining multiple feature types indicates that future research should continue to explore novel feature extraction methods while building upon proven techniques like MFCCs. The challenges identified in these studies, particularly regarding noise interference and computational requirements, highlight areas requiring focused attention in future research.

Looking forward, several key research directions emerge from this analysis. The development of larger, more diverse datasets that capture a wider range of vehicle types and

environmental conditions appears crucial for advancing the field. The potential for hybrid approaches that combine the strengths of different classification methods while managing computational requirements presents a promising avenue for investigation. The integration of real-time classification capabilities while maintaining high accuracy remains an important goal for practical applications. Additionally, the exploration of novel feature extraction methods and their combination with established techniques could lead to improved classification accuracy for similar vehicle types, addressing one of the persistent challenges identified across studies.

2.6 Summary of Previous Research

Table 2.1: Summary of Previous Research

Study	**Methodology**	**Key Findings**
Exploring Sound Signature for Vehicle Detection and Classification Using ANN (George, Cyril, et al., 2013)	1. Data collection on a two-lane road with moderate traffic using Sony Handycam and Zoom H4n. 2. Sound data analysis and labeling using WaveSurfer software. 3. Feature extraction using energy-based features and MFCCs. 4. Classification using Multi-Layer Feedforward Neural Network (MLFFNN). Refer to Figure 2.4	1. Effectiveness of smoothed logarithmic energy in identifying vehicle sounds. 2. Possible confusion between vehicles with similar engine types. 3. Role of MFCCs in capturing spectral characteristics. 4. Overall vehicle classification accuracy of approximately 67%.

Table 2.1: continued

Vehicle Detection and Classification from Acoustic Signal Using ANN and KNN (George, Mary, et al., 2013)	1. Roadside vehicle sound recordings for minimal background noise. 2. Preprocessing includes smoothed logarithmic energy extraction and horn sound elimination. 3. Feature extraction using MFCCs from detected acoustic peaks. 4. Classification using Artificial Neural Networks (ANN) and K-Nearest Neighbors (KNN).	1. ANN achieved higher accuracy (73.42%) compared to KNN (50.62%). 2. Importance of preprocessing in enhancing detection accuracy.

Table 2.1: continued

Spectral Features for the Classification of Civilian Vehicles using Acoustic Sensors (Kozhisseri & Bikdash, 2009)	1. Engine sound recording from idling cars with open hoods. 2. Fundamental frequency estimation using Burg's method, Auto-correlation method, and Yin algorithm (Figure 2.3). 3. Investigation of tristimulus and generalized M-stimulus responses as potential features.	1. Successful use of Power Spectral Density (PSD) for spectral analysis. 2. Tristimulus and M-stimulus responses effectively distinguish engine sounds, even in noisy environments. 3. Importance of accurate fundamental frequency (f0) estimation for capturing engine sound characteristics.

Table 2.1: continued

Acoustic Vehicle Classification Using Mel-Frequency Features with Long Short-Term Memory Neural Networks (Yassin et al., 2023)	1. Data collection from roads in Ilmenau, Germany, under different conditions (dry/wet, urban/country roads) with a sampling rate of 48 kHz. 2. Data processing: stereo to mono audio, downsampling to 22 kHz, 2-second duration. 3. Feature extraction using MFCCs (13 coefficients). 4. LSTM neural network design and implementation. Refer to Figure 2.1 and 2.2	1. Achieved classification accuracies between 82% and 86.2%, varying with hidden layers and initialization seeds. 2. Accurate identification of cars and no traffic. 3. Poor recognition of truck and motorcycle sounds.

Table 2.1: continued

Audio-Based Hierarchic Vehicle Classification for Intelligent Transportation Systems (Kubera et al., 2015)	1. 1-hour data collection in a suburban area near Lublin, Poland, including cars, trucks, and tractors. 2. Feature extraction using spectral and time-domain features. 3. Classification using Decision Tree, Random Forest, MLP, and SVM. 4. Comparison of single multi-class, binary, and hierarchical classifiers.	1. Binary classifiers outperformed multi-class classifiers. 2. Longer frames resulted in higher accuracy. 3. Hierarchical classifier achieved the highest accuracy.

2.7 Summary

This chapter reviewed key concepts and advancements relevant to identifying stock car models using machine learning and acoustic signatures. It traced the evolution of audio identification from speech recognition to modern classification techniques, highlighting studies that validated the feasibility of acoustic-based vehicle identification using models like ANN, SVM, and LSTM. Key components such as data collection, feature extraction (e.g., MFCCs, spectral centroid, harmonic patterns, PSD), and model development were discussed, alongside challenges like noise interference, category ambiguity, and computational demands. Despite these hurdles, prior research demonstrated promising accuracies, offering a strong theoretical foundation for the development of a robust audio-based vehicle identification system in this project.

Chapter 3: Methodology

3.1 Introduction

This chapter discusses the project methodology that employs for this project. In total there are five steps to be done to successfully complete this research project.

3.2 Project Methodology

Project Methodology provides a work plan and simultaneously explaining the activities needed for the FYP's completion. The first phase will include the overall findings from Chapter 2: Literature Review. Phase 2 and 3 of Data Collection and Preprocessing will explain on how the audio data are collected and how it they will be processed. Phase 4 of Machine Learning will dive more into the processing, comparison, evaluation of Data Mining. Phase 5 of Product Development has been decided to be a Web based system using the Prototyping model. The UML Diagram and data and function will also be related here. The last phase of Documentation is a detailed report that summarise the key findings of this Final Year Project.

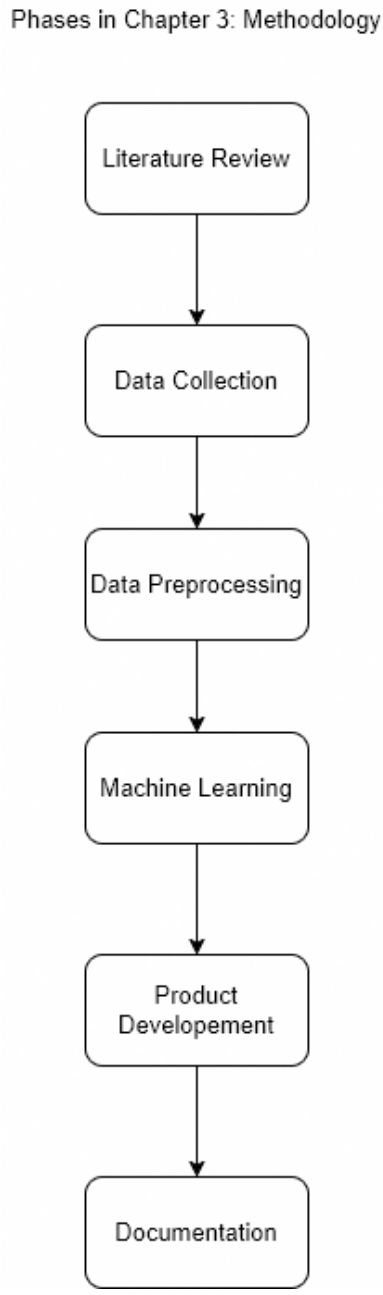


Figure 3.1: Overview of Research Methodology

3.3 Phase 1: Literature Review

This phase is an extensive review of relevant literature and research is conducted to explore the potential of machine learning techniques for stock car model identification through acoustic signatures. Key areas include the evolution of acoustic recognition, audio classification techniques, and related vehicle sound identification research. Furthermore, the literature review examines past research efforts, including studies on general vehicle detection and specific classifications based on spectral features. These studies provide valuable insights into critical components such as data collection, feature extraction, and model development. Multiple machine learning models were tested and implemented by the researchers. Commonly used features, including Mel-Frequency Cepstral Coefficients (MFCCs), spectral centroid, harmonic patterns, and Power Spectral Density (PSD), are instrumental in capturing unique acoustic signatures. The review also identifies challenges such as noise interference, ambiguity between similar vehicle categories, and computational demands of advanced models like deep learning. Despite these challenges, the reviewed studies demonstrate promising classification accuracies, affirming the potential of leveraging machine learning for nuanced audio-based applications. This literature review provides a robust theoretical framework for the project, emphasizing the significance of accurate feature extraction, diverse data collection, and appropriate machine learning techniques. The detailed review is written in Chapter 2 of the report.

3.4 Phase 2: Data Collection

Specifically for this research, the dataset of car sounds is obtained directly by recording the vehicle sounds of cars inside of Forza Horizon 5. The developers of Forza Horizon 5 have meticulously crafted each and every car to sound as close as possible to their real-world counterparts making it a reliable and consistent source of dataset. This also serves as a workaround for the nigh impossible task of collecting cars sounds in the real world without some kind of insider access as going through this route would require knowing an individual or a conglomerate with a massive car collection and a be kind enough to allow recordings of more than 100 cars which no doubt will also be insanely expensive. The recordings inside of Forza Horizon 5 allows for a single student to have access to a scalable and cost-effective means for obtaining a comprehensive dataset of diverse car models. To ensure consistency and reliability of the dataset, a standardized recording protocol is implemented across all samples. All recordings are conducted in the same in-game location under controlled conditions, with weather settings maintained at clear conditions and time of day set to noon to minimize environmental variations. The audio is captured at a 48 kHz sampling rate with 24-bit depth, in stereo format. Multiple recordings will be done to obtain data variation.

3.5 Phase 3: Data Preprocessing

Once raw recordings are collected, they undergo an intensive preprocessing phase to ensure uniformity and quality. First, all audio files are converted to a standardized format, such as FLAC, then inspected for any anomalies or recording errors. This involves using

visualization tools to identify any irregular spikes or dips in the audio waveforms that may indicate unwanted noise or clipping. The audio recordings are trimmed to remove any unnecessary silence or gaps at the beginning and end. This ensures that the analysis focuses solely on the relevant car sound information and avoiding tainting the produced ML models.

To enhance the robustness of the resulting model, data augmentation techniques are employed. These include time stretching by ±10%, pitch shifting by ±2 semitones and audio gain of ±6db. Quality control measures are implemented throughout the process, verification of feature extraction consistency, and cross-validation of augmented data quality. All preprocessing parameters and steps are meticulously documented to ensure reproducibility.

Once cleaned and processed, the audio data will be prepared for feature extraction. This involves defining relevant features that capture essential characteristics of the car sounds. Techniques such as Mel-Frequency Cepstral Coefficients (MFCCs), Zero Crossing Rate (ZCR) Spectral Centroid, Log-Mel Spectrogram, Chroma Features and Tonnetz Features will be utilized to convert audio signals into numerical representations that machine learning algorithms can interpret effectively.

3.6 Phase 4: Machine Learning

Based on the literature review and the specific requirements of identifying stock car models through acoustic signatures, three machine learning approaches have been selected: Convolutional Neural Networks (CNN) as a representative deep learning approach, Support Vector Machines (SVM), and Random Forest (RF). These models were chosen based on

their demonstrated success in related audio classification tasks and their complementary strengths in handling spectral data.

The Convolutional Neural Network (CNN) is selected as the deep learning model due to its exceptional performance in processing spectral representations of audio data, as demonstrated in the work of Yassin et al. (2023). CNNs are a specialized type of deep learning model designed for analysing image-like data, such as the spectrograms generated from car sounds. They have shown remarkable success in various audio classification tasks due to their ability to automatically learn hierarchical feature representations from the data. Convolutional Neural Networks (CNNs) represent the deep learning approach in this study, leveraging their ability to learn spatial hierarchies directly from raw input data such as spectrograms. This model is expected to yield superior performance in identifying subtle differences between car sounds due to its ability to learn high-level abstractions from raw audio data.

SVMs are a powerful class of supervised learning algorithms used for classification and regression tasks. They are particularly effective in handling high-dimensional data, making them suitable for analysing the complex features extracted from car sounds. Support Vector Machines (SVM) is chosen as the second model based on its strong performance in vehicle classification tasks, as shown in Kubera et al. (2015). SVMs are particularly suitable for this project as they can classify car sounds by finding the optimal hyperplane in feature space. In the context of this project, this model is expected to perform well with the extracted features such as MFCCs and spectral characteristics, especially when the dataset is not excessively large.

Random Forest (RF) is selected as the third model due to its effectiveness in handling complex audio classification tasks and its inherent ability to rank feature importance. By aggregating multiple decision trees, RF reduces overfitting and provides reliable predictions. This model excels in scenarios with noisy data and complex feature interactions, making it ideal for the variability in acoustic patterns across car models. Random Forest's ability to handle multiple feature types simultaneously makes it particularly suitable for combining different acoustic features like MFCCs, spectral centroids, and harmonic features.

For dataset splitting and evaluation, a stratified k-fold cross-validation approach will be implemented with k=5, ensuring each fold maintains the same distribution of car models as the complete dataset. This approach is particularly important given the potential imbalance in the number of recordings available for different car models. To ensure the models' reliability and generalization capability, the dataset is split into training and testing sets using an 80/20 ratio. This means that 80% of the data is used to train the models, while 20% is reserved for testing their performance on unseen data. Using this common selection of split helps prevent overfitting, where the models perform well on the training data but poorly on new, unseen data.

To compare the performance of these models, several appropriate evaluation metrics were considered and accuracy was ultimately chosen as the single most important metric. This metrics will provide a comprehensive view of each model's effectiveness in classifying stock car models based on their acoustic signatures. Accuracy is the measurement of the overall correctness of the model's predictions which will be sufficient enough to determine the effectiveness of the application for this Final Year Project.

By combining traditional machine learning approaches with deep learning techniques, this project aims to identify the optimal model for stock car classification through car sounds. The inclusion of CNNs ensures that the system benefits from state-of-the-art advancements in audio analysis, while SVM and RF provide robust and interpretable benchmarks for comparison. This comprehensive evaluation approach will enable objective comparison between the models and ensure the selection of the most suitable approach for the stock car identification system, considering both performance and practical implementation constraints.

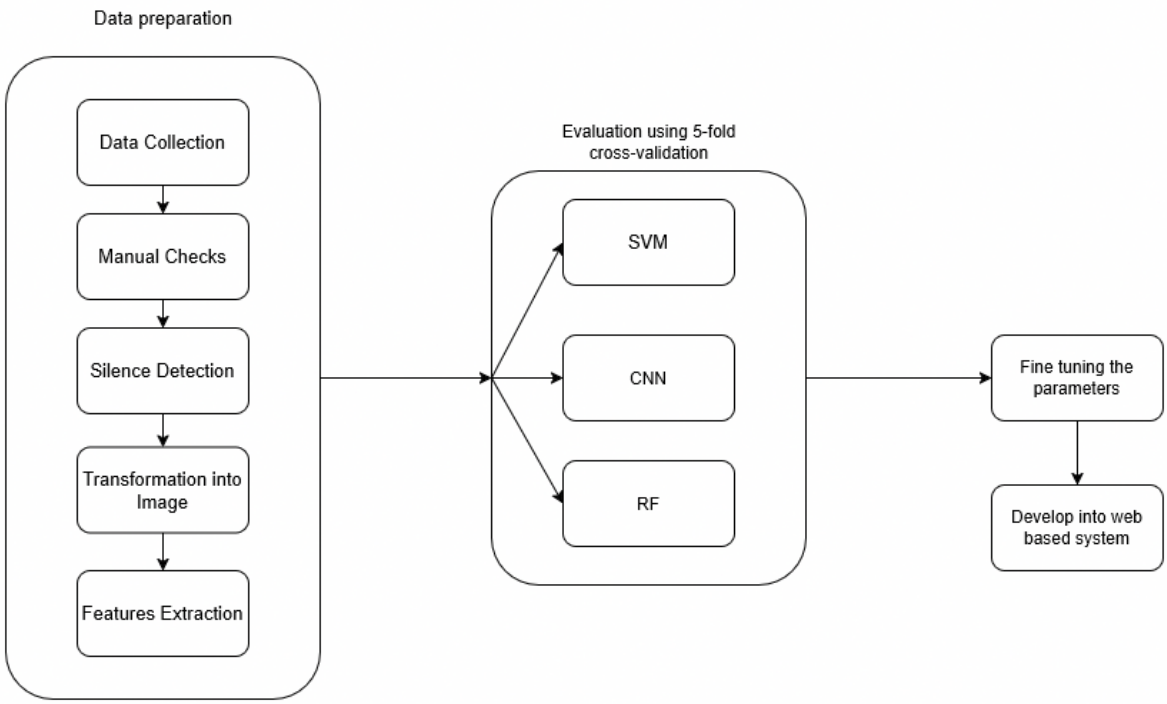


Figure 3.2: Overview of the data preparation and machine learning algorithms evaluation before the system development

3.7 Phase 5: Product Development

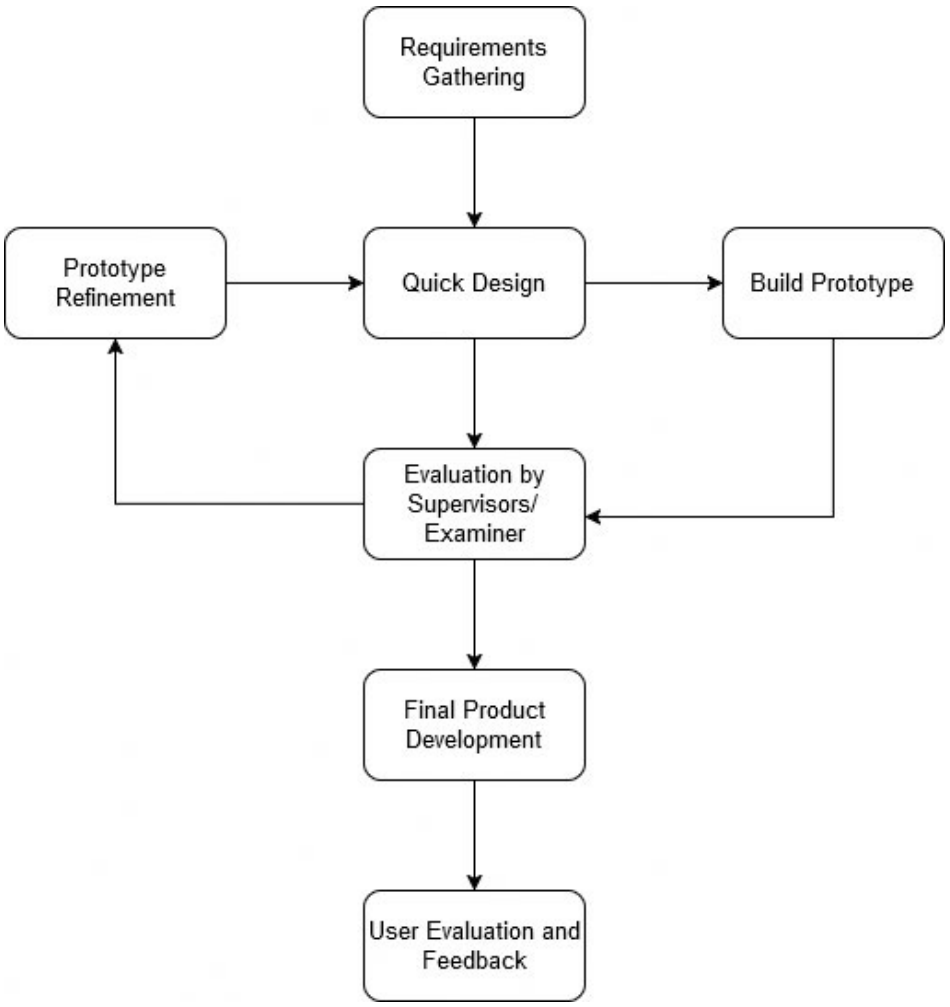


Figure 3.3: Flow of the Prototyping Model

In the context of this research, product development will follow the prototyping model, which emphasizes iterative design and user feedback to refine the machine learning-based identification system for stock car models. The prototyping model is particularly suitable as it allows for early validation of the audio processing and machine learning components, enables iterative refinement based on user feedback, further reduces development risks through continuous testing and evaluation coupled with flexibility to

adapt to changing requirements. The process consists of seven stages: Requirements Gathering, Quick Design, Build Prototype, Evaluation by Supervisors/Examiners, and Prototype Refinement. Early prototypes will undergo multiple iterations, refining previous stages until they reach a sufficient level of maturity. Once the prototype is refined, it will undergo User Evaluation and Feedback, before moving on to the Final Product Development stage.

3.7.1 Stage 1 - Requirements Gathering

The first stage involves a comprehensive analysis of the system requirements, establishing the foundation for the entire development process. The requirements analysis focuses on several critical aspects: the audio recording and processing capabilities needed to handle car sound data, the computational requirements for implementing machine learning models, and the user interface specifications that will make the system accessible and user-friendly. Collaborative sessions with stakeholders, including project supervisors and domain experts, help refine these requirements and prioritize features for subsequent development phases. Furthermore, methodology on the ways to evaluate the application with users will also be analysed, finding the best methods for feedback and evaluation to be received.

3.7.1.1 Requirements for Application Development

The development of the application for this project will focus on a web-based platform to ensure accessibility, scalability, and cross-platform compatibility. This choice leverages the universal availability of web browsers, eliminating the need for platform-

specific installations while maintaining a user-friendly interface. The primary goal is to design a system capable of handling audio recordings, performing preprocessing, and classifying stock car models based on sound signatures, all within a seamless web environment.

The application will consist of two main components: the front-end interface and the back-end processing system. The front-end will be developed using modern web technologies such as HTML, and CSS, ensuring a responsive and intuitive user experience. Users will interact with the application via a clean and minimalistic interface to upload car sounds, view the classification results, and access related functionalities. Tools like Figma/Draw.io will be utilized during the design phase to prototype and refine the user interface based on feedback.

On the back end, the core functionality of audio preprocessing and classification will be implemented using Python and its robust ecosystem of libraries. Additionally, machine learning models, will be trained and hosted locally through Django Framework in Visual Studio Code. By adopting this specific architecture, the application should be much easier to be created and tested which aligns with the prototyping model.

3.7.2 Stage 2 - Prototyping

The prototype for the web-based application will be designed and developed after gathering the initial requirements. The prototyping phase will mostly concentrate on the core functionalities which will be uploading the audio file, preprocessing of car sound data, and real-time classification of stock car models. Ensuring a seamless user experience and

functional compatibility across different web browsers will be key priorities during this stage. The iterative design approach will be employed to refine the prototype based on evaluations and feedback provided by the supervisor. User interface elements, workflow efficiency, and back-end integration with machine learning models will be tested and updated to meet the project's objectives. This phase will conclude once a significant portion of the application's requirements has been implemented, and critical bugs have been addressed to ensure stable and consistent performance.

3.7.2.1 Quick Design

The Quick Design phase of this project focuses on establishing the foundational architecture and components necessary for the car sound analysis system. This initial design acts as a blueprint for the first prototype, outlining key user interface elements such as the audio input mechanism, the display area for classification results, and basic controls to initiate the classification process. The quick design is intentionally kept streamlined to facilitate rapid prototyping and early feedback. The key elements of the quick design include User Interface (UI), Core Functionality Integration, and Placeholder Components.

The free online tool of Draw.io is a cross-platform graph drawing software application developed in HTML5 and JavaScript. Its interface can be used to create diagrams such as flowcharts, wireframes, UML diagrams, organizational charts, and network diagrams. Draw.io is very sufficient in creating UML Diagrams that will aid us in visualising system structure and behaviour, documenting requirements and design, and also allowing for better

analysis. The high-level overview of components they provide helps in designing software architecture which are essential in complex systems.

A basic interface will be developed to facilitate interaction with the system. Users will be able to upload audio files, initiate the classification process, and view results in a clear and organized manner. Core Functionality Integration should include fundamental features such as audio file uploading, preprocessing, and classification that will be integrated into the prototype. Placeholder machine learning models may be employed during this phase to simulate classification outputs. These placeholders will provide mock results to validate the end-to-end flow without requiring fully trained models initially. The prototype will include a feedback collection mechanism, such as a form or survey, enabling users to provide input on their experience.

The design emphasizes modularity and scalability, ensuring that new features or improvements can be integrated without major system refactoring. By focusing on essential functionalities and gathering early feedback, this phase ensures that the development process remains aligned with user expectations and technical requirements. The prototyping phase will conclude once a significant portion of the application's requirements have been identified and addressed through its functionalities and bug fixes.

3.7.2.1.1 Use Case Diagram

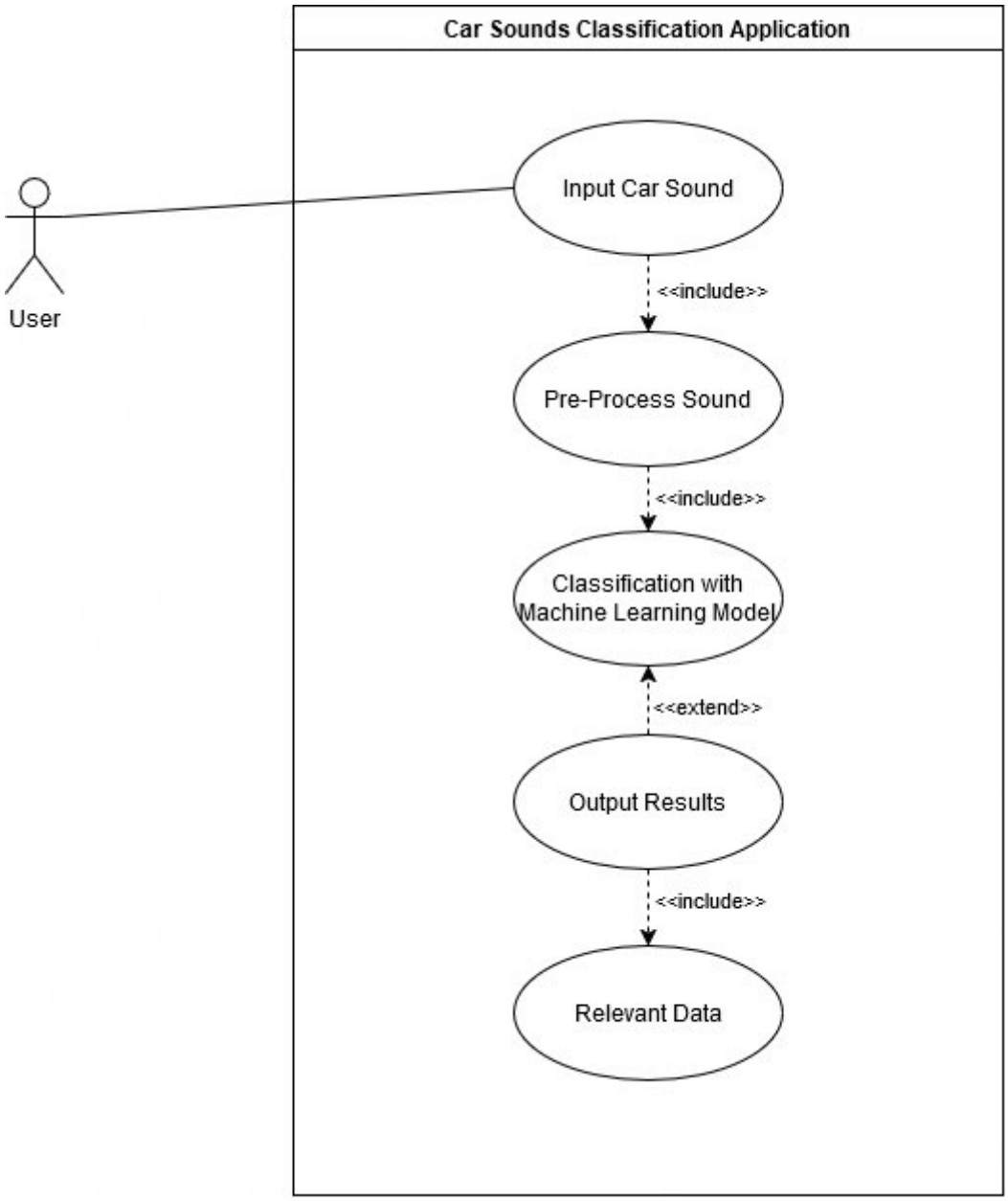


Figure 3.4: Use Case Diagram

The use case diagram above illustrates the interaction between the user and the application, highlighting the key functionalities of the proposed system.

Initially, the user will input the car sound into the system. The system will then preprocess the audio to extract relevant features, optimizing it for accurate classification. After preprocessing, the audio will be passed through machine learning models for classification. Finally, the results, along with all pertinent data, will be displayed to the user in a timely manner.

Shown below is the Use Case Specification for the use case diagram.

Use Case Specification: Car Sound Classification System

1. **Use Case Name**

Car Sound Classification

2. **Actors**

- Primary Actor: User
- Secondary Actor: Machine Learning Model

3. Preconditions

- The user must have an audio file of a car sound.
- The system must be running and ready to process input.

4. **Flow of Events**

Basic Flow

1. The user accesses the system via a web interface.
2. The user uploads a car sound audio file.
3. The system preprocesses the audio file (e.g., Silence detection, feature extraction).
4. The processed audio data is analysed using a machine learning model.
5. The system classifies the car model based on the extracted features.
6. The classification result is displayed to the user, and possible matches.

5**. Alternative Flows**

- File Format Error: If the uploaded file format is not supported, the system notifies the user and prompts for a valid format.

6. **Postconditions**

- The system stores the classification result for potential model improvement.
- The user receives the classification result.

3.7.2.1.2 Class Diagram

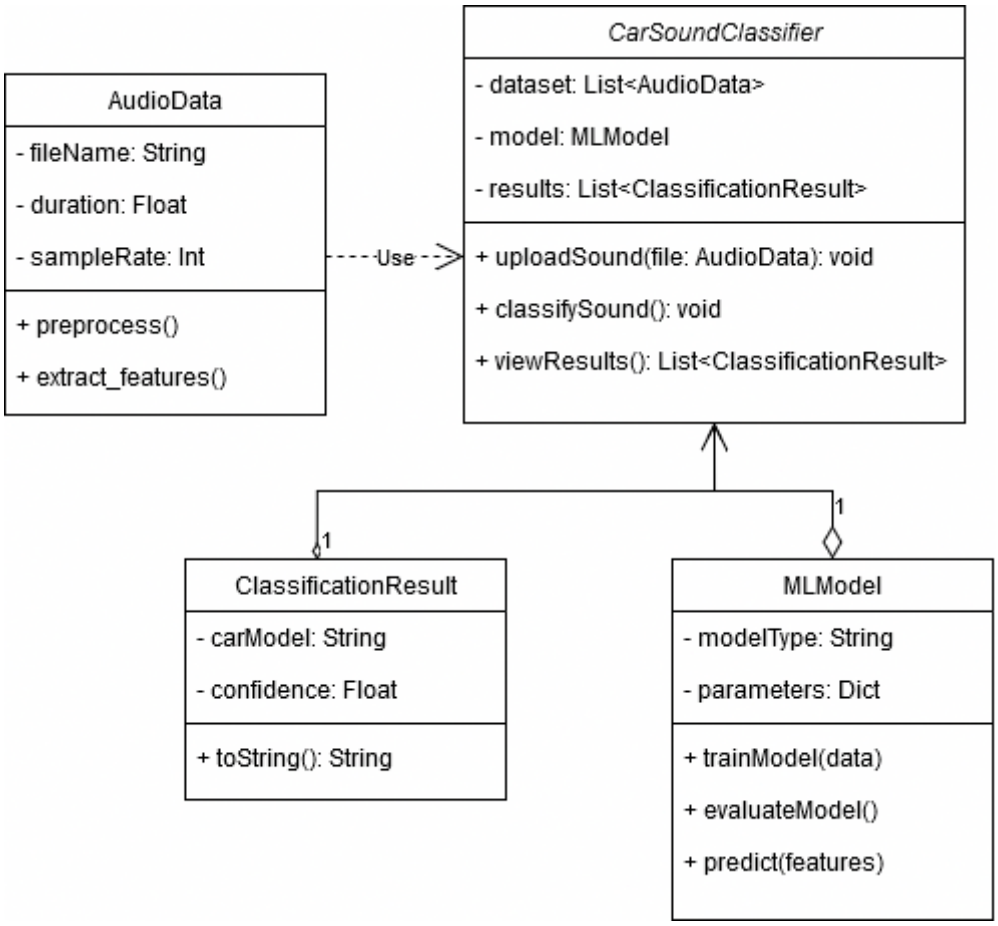


Figure 3.5: Class Diagram

A class diagram is a type of static structure diagram used in object-oriented modeling to represent the structure of a system by showing its classes, attributes, methods (operations), and the relationships between the classes.

The class diagram above illustrates the structure of the machine learning-based identification system for stock car models. It consists of four primary classes: CarSoundClassifier, AudioData, MLModel and ClassificationResult. This modular design

ensures separation of concerns and supports scalability, as new features like alternative machine learning models or enhanced result formatting can be easily integrated.

The CarSoundClassifier is the central class in the diagram and manages the overall classification system. It contains three key attributes: dataset (a collection of audio data), model (an instance of the MLModel class), and results (a list of classification results). The primary operations include uploading sounds, classifying them, and viewing the results.

The MLModel class encapsulates the machine learning model used for classification. Key attributes include modelType (e.g., SVM, RF, CNN), accuracy, and parameters (a dictionary of parameters). The class provides methods allow for training the model on provided data, evaluating its performance using various metrics, and making predictions based on extracted features, which then returns a ClassificationResult.

Finally, the ClassificationResult class stores the outcome of a classification. Its attributes include carModel (the predicted model) and confidence (the prediction certainty). The toString method generates a readable summary of the classification.

3.7.2.1.3 Sequence Diagram

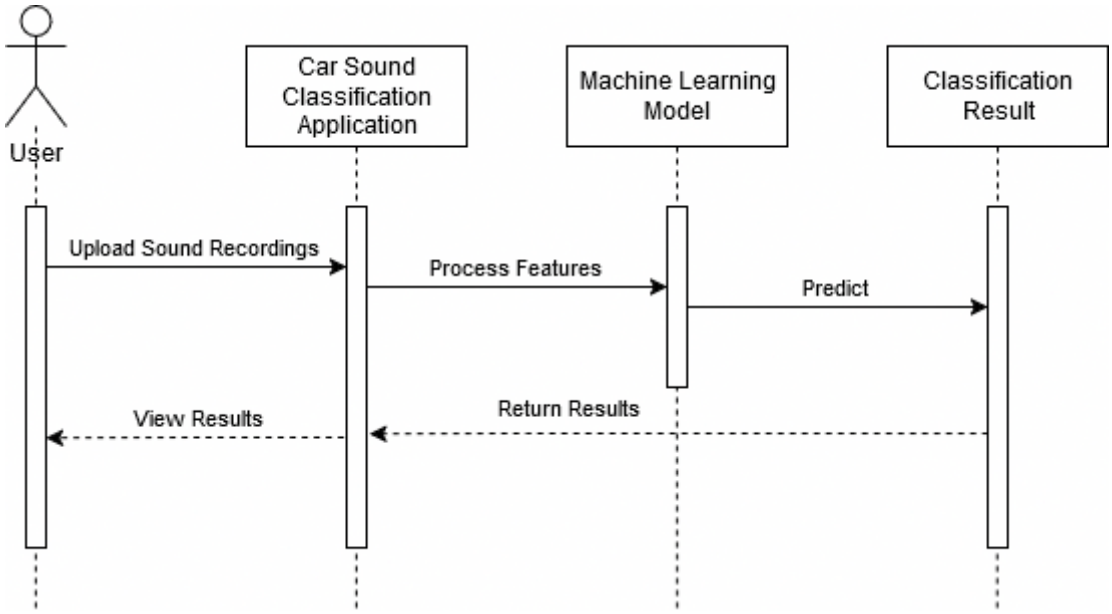


Figure 3.6: Sequence Diagram

A sequence diagram is a type of interaction diagram in UML (Unified Modeling Language) that visually represents how objects or components interact over time. It shows the sequence of messages exchanged between participants (objects, classes, or components) in a particular use case or scenario. As illustrated in Figure 3.5, the sequence diagram consists of three objects, Car Sound Classification Application, Machine Learning Model and Classification Result.

The sequence begins with the user uploading sound recordings containing car sounds onto Car Sound Classification Application. This triggers it to process the sound recordings and transmit the processed features to the Machine Learning Model object. The Machine Learning Model processes the input and creates a new ClassificationResult object which

contains the prediction and all the relevant data. The results are then sent back to the application, allowing the user to view the final classification results.

3.7.2.1.4 State Diagram

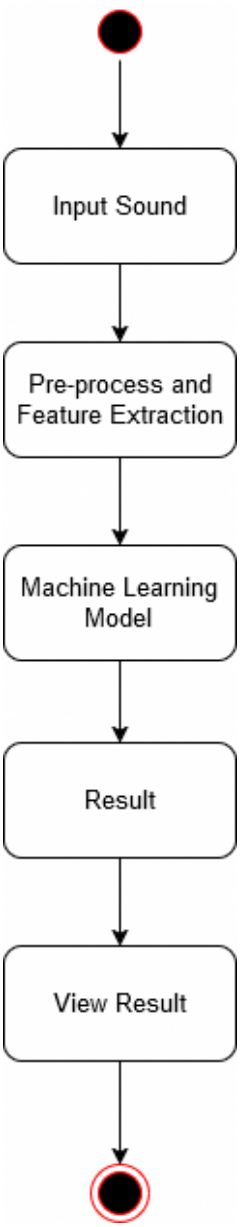


Figure 3.7: State Diagram

A state diagram, also known as a state machine diagram or state chart diagram, is a visual representation of the different states that an object or system can be in, and the transitions between those states. It's a powerful tool for understanding and designing complex systems, particularly those that involve dynamic behaviour and changing conditions

Figure 3.6 is a relatively simple state diagram that consists of 7 states which are the initial state, Input Sound, Pre-process and Feature Extraction, Machine Learning Model, Result and View Result. The application starts first with the "Input Sound" state, where the application awaits audio input from the user and would be done by uploading an audio file.

Next, the process moves to "Pre-process and Feature Extraction." Here, the application will process the data through multiple steps as mentioned before. It then extracts key features from the sound, such as Mel-Frequency Cepstral Coefficients (MFCCs) plus other spectral characteristics, which are essential for the machine learning model to analyse. The extracted features are then passed to the "Machine Learning Model" state. In this state, the trained machine learning model, be it an SVM, KNN, or CNN, processes the features to predict the car model that produced the sound.

The "Result" state represents the outcome of the classification. The application determines the predicted car model based on the output of the machine learning model. Finally, the "View Result" state displays the predicted car model to the user. This could be a simple text display, or it might include additional information like the confidence level of the prediction or a visual representation of the analysed sound. The diagram ends with a final state, indicating the completion of the classification process.

3.7.2.2 Mock Up

Figma, the free and online interface design tool was selected to create the UI mock-up. These early mock-ups shown below will be incredibly helpful in offering a visual representation of the system's structure and functionality. These designs provide both users and developers with a clear understanding of the system's intended appearance and operation upon completion. This will contribute to enhancing the likelihood of the system's success and its broader adoption by students and researchers.

The simple design provided below adheres to modern design languages of minimalism with functionality while still maintaining a pleasing overall design that will appeal to everyone. The mock-ups consist of 2 pages, the main page and the result page. The main page is the page where users would first encounter the web app and it only has one function which is to upload the audio file. User will perform this function by either clicking the upload button or the word itself. The second page is the result page where once the audio file has been uploaded and the results having been generated, it will then be shown on this page. The overall UI is meant to be easy to understand and be used by everyone of all ages.

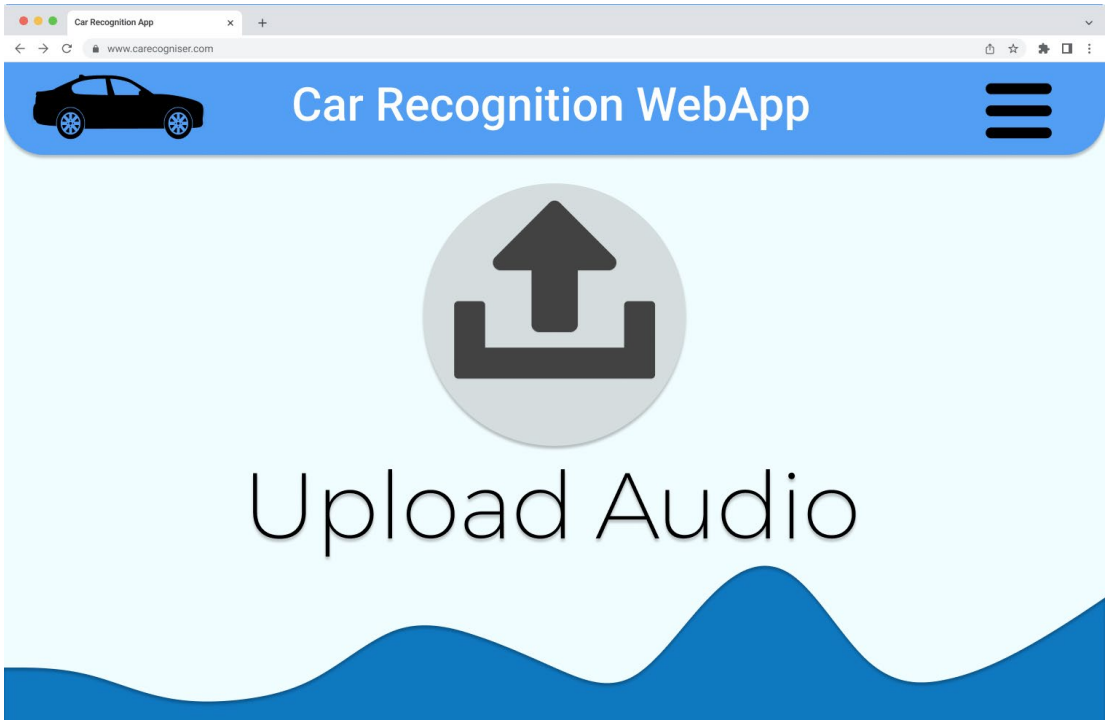


Figure 3.8: Main Page

Figure 3.9: Results Page

3.7.2.3 Build Prototype

Prototype of application will be rapidly built after each quick design iteration. Subsequent prototype should be even more complete and successfully complements previous prototypes by incorporating each design's proven elements. The prototype will then get send off to be evaluated by supervisors/examiners. If the prototype is considered unfinished and needs more polishing, the prototyping cycle will be repeated until it is finally complete.

3.7.2.4 Evaluation by Supervisors/Examiners

The evaluation phase will involve presenting the developed prototype to supervisors and examiners for feedback and assessment. The primary objective of this stage is to ensure that the prototype aligns with the defined requirements and meets the expected standards of functionality, usability, and performance. Supervisors and examiners will assess various aspects of the application, including its ability to accurately classify car sounds, the efficiency of its audio preprocessing methods, and the overall user experience of the web interface. Feedback from this evaluation will be collected systematically through structured testing sessions and direct discussions. The feedback gathered will act as shining beacon to guide subsequent iterations of the prototype.

3.7.2.5 Prototype Refinement

Any room for improvement, including enhancements to the machine learning model, user interface, or system reliability, will be documented and prioritized for implementation.

The prototype will then undergo further refinement until it achieves a level of reliability and functionality suitable for real-world application.

Key areas of refinement will mostly be in improving the accuracy and efficiency of the machine learning models, optimizing the audio preprocessing pipeline, and resolving any identified bugs or inconsistencies. Adjustments to the user interface will also be made to enhance accessibility and ensure a seamless user experience. For example, elements such as file upload mechanisms, result displays, and system responsiveness will be reviewed and refined to ensure they function intuitively and efficiently.

This iterative aspect ensures the final product meets the stated technical and functional expectations and achieves the project's overall objectives. Once the refinements are complete, the prototype will be ready for final evaluation and preparation for deployment as a fully functional web-based application.

3.7.3 Stage 3 - Final Product

The final product stage of product development involves incorporating the insights and feedback gathered from the previous stages to create the final product. This includes refining the user interface, optimizing the performance of the machine learning models, and ensuring the system meets all the defined requirements by thoroughly testing to ensure accuracy, efficiency, and user-friendliness. The Final Product phase ensures the delivery of a polished and reliable car sound classification application. By integrating feedback from earlier stages and focusing on user-centric design, the system is positioned to meet both technical and practical requirements effectively.

3.7.4 Stage 4 - User Evaluation and Feedback

In this phase, a usability test will be conducted by presenting the final product to a select group of users for evaluation and feedback. The evaluation process will employ a mixed-methods approach, incorporating both structured testing protocols and open-ended feedback sessions. Participants will be asked to interact with the system through a series of predetermined tasks, such as uploading different car sound samples and interpreting the classification results. Their interactions will be monitored to assess metrics such as task completion time, error rates, and the intuitiveness of the user interface. Additionally, users will be asked to rate their experience across various dimensions, including ease of use, system responsiveness, and result clarity, using a standardized 5-point Likert scale in a SUS questionnaire. A Google Form survey will be created and sent to users to gauge their overall satisfaction with the system and to collect qualitative insights on potential improvements. The survey will include both quantitative questions (such as ratings on system performance) and open-ended prompts, encouraging users to provide detailed feedback on their experience, challenges faced, and suggestions for enhancing the usability and functionality of the prototype. As illustrated in Figure 3.10, the figure shows a list of questions taken directly from Appendix A. The complete SUS questionnaire is in Appendix A.

		Strong disagree 1	2	3	4	Strongly agree 5
1.	I think I would like to use this web-based system more frequently					
2.	I think the web-based system is too complicated					
3.	I think the web-based system is simple to use					
4.	I would need help from a technical person to use this web-based system					

Figure 3.10: SUS Questionnaire

3.8 Phase 6: Documentation

Once the preceding methodological stages have been successfully completed, the next step will be to initiate the documentation phase of the project. Documentation serves as a crucial step in the Final Year Project, providing a detailed repository of the entire research process. This documentation consists of a thesis report, research paper and poster that encompasses all aspects of the project, from the initial literature review and methodology to the results, evaluations, and conclusions. The documentation is structured in a clear and concise manner, ensuring readability and ease of navigation. It includes relevant diagrams,

charts, and tables to illustrate key findings and processes. The documentation also adheres to established academic standards, including proper citations and references. It serves as a comprehensive guide for anyone interested in understanding or replicating the research.

3.9 Hardware Requirement

The hardware requirements for the proposed system:

Table 3.1: Hardware Requirements

No.	**Hardware**	**Specifications**
1.	CPU	AMD Ryzen 5 4500U
2.	RAM	8GB 3200 MT/s
3.	GPU	AMD Integrated Radeon Graphics
4.	Operating Systems	Windows 11 24H2

3.10 Software Requirement

The software requirements for the proposed system:

Table 3.2: Software Requirements

No.	**Software**	**Requirement**
1.	IDE	Microsoft Visual Studio Code
2.	Front End	CSS, HTML

Table 3.2: Continued

3.	Machine Learning Language and Library	Python, Python Libraries
4.	Web Browser	Microsoft Edge, Mozilla Firefox

3.11 Summary

This project employed a multi-phased methodology for developing a machine learning algorithm capable of identifying car models based on their acoustic signatures and subsequently creating a user-friendly application that was used to perform the aforementioned actions. The initial phase involved a comprehensive literature review to analyse existing research and methodologies within the field of car sound classification. This was followed by a meticulous data preparation phase, encompassing data acquisition, rigorous processing, and strategic augmentation to enhance model performance. The core of the project lay in the evaluation phase, where three distinct machine learning models were rigorously assessed for their effectiveness in accurately classifying car sounds. Building upon these findings, the project transitioned into the application development phase, focusing on prototyping a user-friendly application that seamlessly integrated the developed classification algorithm. Finally, the project culminated in the documentation phase, encompassing the creation of a comprehensive thesis report, a research paper, and an engaging poster to effectively communicate the project's findings.

Chapter 4: Implementation

4.1 Introduction

This chapter provides a detailed explanation of the implementation process for the Web-Based Car Recognition system using Machine Learning techniques. It outlines the tools and methodologies employed in the system's development, including the creation of the front-end graphical user interface using HTML, CSS and also Django Framework. Additionally, this chapter will explain the implementation of Python and the integration of the three machine learning techniques (SVM, RF, CNN) within the system.

4.2 Hardware and Software Environments

The table 3.1 below is the hardware and software environment used for the implementation of Web-based Car Recognition system

Table 4.1: Hardware and Software Environment

No.	**Hardware**	**Specifications**
1.	CPU	AMD Ryzen 5 4500U
2.	RAM	8GB 3200 MT/s
3.	GPU	AMD Integrated Radeon Graphics
4.	Operating Systems	Windows 11 24H2
5.	Integrated Development Environment	Microsoft Visual Studio Code
6.	Front end	HTML, CSS, Django Framework

Table 4.1: Continued

7.	Programming Language	Python
8.	Web Browser	Mozilla Firefox

The web-based system was developed using a personal thin and light laptop equipped with an AMD 4500u CPU, a max wattage of 25W, base clock of 2,38 GHz, maximum turbo of 3.99 GHz and 8 GB of RAM while running the latest version or Windows 11 OS. The GPU, an Integrated Radeon Graphics was not utilised at all since it does not have CUDA cores, which is the industry method of training machine learning models. The machine learning models were primarily trained using pure CPU power. Table 4.1 contains the summarisation of Hardware and Software Environment. The personal laptop provided enough power to handle the whole development process plus the training of the machine learning models.

4.3 Machine Learning Implementation

This section provides a detailed explanation of the augmentation techniques applied to the audio dataset, followed by the preprocessing steps and feature extraction methods. Additionally, the implementation of various machine learning algorithms, including Support Vector Machine (SVM), Convolutional Neural Networks (CNN), and Random Forest (RF), is discussed.

4.3.1 Data Augmentation

Some of the datasets will be untouched, same as it was obtained from the recordings while the others will be recordings that will undergo data augmentation to introduce variability into the data. The *Audio_augmentation* script meticulously automates the process of audio data augmentation, a vital technique for enhancing the robustness and generalization capabilities of audio-based machine learning models. Operating on audio files within the specified input directory, the script generates a configurable number of augmented versions for each original audio file.

The core of its functionality lies in the application of a sequential augmentation pipeline, defined using the audiomentations library. This pipeline, as currently configured, applies random time stretching (speed variation), pitch shifting, and gain adjustments to the audio signals, introducing subtle yet significant variations. Each augmented version is then saved into a newly created, version-specific subdirectory within the designated output base directory, ensuring organized storage of the expanded dataset. The filenames of the augmented files are systematically constructed to reflect their origin and augmentation version. By applying these transformations, the script effectively increases the size and diversity of your audio dataset, thereby potentially improving the performance and real-world applicability of any machine learning models trained on this augmented data.

Several important considerations underpin the design and application of this script within the project. Firstly, the selection and parameterization of augmentation techniques are tailored to the specific characteristics of the audio data and the requirements of the target application. Over-augmentation, a potential pitfall, can introduce unrealistic or detrimental artifacts, ultimately hindering model performance. Finally, the realism of the applied

augmentations is paramount; the generated synthetic data should closely approximate real-world variations to ensure effective model generalization. This automated approach streamlines the creation of augmented audio data, saving time and effort in preparing datasets for the Final Year Project.

```
# Define augmentation pipeline
augment = Compose([
    TimeStretch(min_rate=0.9, max_rate=1.10, p=0.6),
    PitchShift(min_semitones=-2, max_semitones=2, p=0.7),
    Gain(min_gain_db=-6, max_gain_db=6, p=0.6)
])
```

Figure 4.1: Augmentation Pipeline

4.3.2 Pre-processing and Feature Extraction

In the process of preparing audio data for machine learning, pre-processing and feature extraction are essential steps that transform raw recordings into structured numerical inputs suitable for model training. The pre-processing stage begins with the removal of silent segments from the audio files. This is accomplished using the *remove_silence* function, which utilizes the *librosa.effects.split* method to detect and eliminate segments of the waveform that fall below a specified decibel threshold, typically considered non-informative. The remaining non-silent audio segments are then concatenated into a single continuous waveform, effectively reducing noise and focusing the analysis on relevant sound content.

Following silence removal, the next phase is feature extraction, handled by the *extract_features* function. This function loads the cleaned audio file and computes a diverse set of acoustic features that collectively capture the characteristics of the sound. These

features include Mel-Frequency Cepstral Coefficients (MFCCs), which are widely used in speech recognition due to their ability to model the human auditory perception. Thirteen MFCCs are computed, and their means are taken over time to form a compact representation. The Zero Crossing Rate (ZCR), a measure of signal noisiness based on the rate at which the audio waveform crosses the zero-amplitude line, is also calculated. Another feature, the Spectral Centroid, indicates the "center of mass" of the audio spectrum and is often associated with the brightness of a sound.

Additionally, the Log-Mel Spectrogram is extracted by applying a mel-scale filter bank to the power spectrogram and then converting it to a decibel scale, providing a detailed frequency representation that mimics human hearing. The means of 128 mel bands are taken to reduce dimensionality. Chroma features, which summarize the intensity of the 12 pitch classes of music (one for each note in an octave), and Tonnetz features, which capture tonal and harmonic relationships within the audio, are also computed. All these features are averaged over time and concatenated into a single feature vector, providing a fixed-length numerical representation of each audio file.

Once features have been extracted from all audio files in a specified directory, they are processed concurrently using multithreading to improve efficiency. The resulting feature vectors, along with filenames, are stored in a structured DataFrame. To prepare the data for machine learning, the features are standardized using *StandardScaler* to ensure that each feature contributes equally during model training. The fitted scaler is saved to disk for use during inference or on future datasets. Finally, the entire dataset of features is saved in both CSV format for readability and a binary NumPy file for computational efficiency. This

pipeline ensures that each audio file is consistently processed and transformed into a robust, machine-readable format.

```python
def remove_silence(y, sr, top_db=30):
    """Removes silent parts from the audio signal."""
    non_silent_intervals = librosa.effects.split(y, top_db=top_db)
    y_trimmed = np.concatenate([y[start:end] for start, end in non_silent_intervals])
    return y_trimmed
```

Figure 4.2: Pre-processing

```python
def extract_features(audio_path, sr=22050, n_mfcc=13):
    """Extracts audio features including MFCCs, ZCR, spectral centroid, log-mel spectrogram, chroma, and tonnetz."""

    # Load audio file
    y, sr = librosa.load(audio_path, sr=sr, mono=True)

    # Remove silence
    y = remove_silence(y, sr)

    # Compute MFCCs
    mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=n_mfcc)
    mfccs_mean = np.mean(mfccs, axis=1)

    # Compute Zero Crossing Rate (ZCR)
    zcr = librosa.feature.zero_crossing_rate(y)
    zcr_mean = np.mean(zcr)

    # Compute Spectral Centroid
    spectral_centroid = librosa.feature.spectral_centroid(y=y, sr=sr)
    spectral_centroid_mean = np.mean(spectral_centroid)

    # Compute Log-Mel Spectrogram
    mel_spectrogram = librosa.feature.melspectrogram(y=y, sr=sr)
    log_mel_spectrogram = librosa.power_to_db(mel_spectrogram, ref=np.max)
    log_mel_spectrogram_mean = np.mean(log_mel_spectrogram, axis=1)

    # Compute Chroma Features
    chroma = librosa.feature.chroma_stft(y=y, sr=sr)
    chroma_mean = np.mean(chroma, axis=1)

    # Compute Tonnetz Features
    tonnetz = librosa.feature.tonnetz(y=librosa.effects.harmonic(y), sr=sr)
    tonnetz_mean = np.mean(tonnetz, axis=1)

    # Concatenate all features into a single feature vector
    feature_vector = np.hstack([mfccs_mean, [zcr_mean, spectral_centroid_mean], log_mel_spectrogram_mean, chroma_mean, tonnetz_mean])

    return feature_vector
```

Figure 4.3: Feature Extraction

```
# Save extracted features to CSV and NumPy file
audio_features_df.to_csv("audio_features.csv", index=False)
np.save("audio_features.npy", audio_features_df.iloc[:, 1:].to_numpy())
```

Figure 4.4: CSV and NumPy saving

4.3.3 Model Training – Support Vector Machine (SVM)

To classify the car audio recordings based on their extracted acoustic features, a Support Vector Machine (SVM) model was developed and trained as part of the machine learning pipeline. The process began with the loading of extracted features from the audio files through the previously generated CSV file. These features included Mel-Frequency Cepstral Coefficients (MFCCs), zero-crossing rate, spectral centroid, log-mel spectrogram, chroma, and tonnetz features, capturing various timbral, tonal, and frequency-based characteristics of the audio. Each audio sample was thus represented as a 162-dimensional feature vector. The corresponding car label was derived from the filename and encoded into numeric form using a LabelEncoder to prepare it for supervised learning.

Prior to model training, the dataset was split into training and testing sets with an 80:20 ratio which will apply for all subsequent models while ensuring class balance through stratified sampling. Given that SVMs are sensitive to feature scaling, all features were standardized using StandardScaler to have zero mean and unit variance. To identify the optimal SVM configuration, a grid search with 5-fold cross-validation was performed using GridSearchCV. The hyperparameter search space included various values for the regularization parameter C, kernel type (linear, rbf, poly), and kernel coefficient gamma.

This exhaustive search ensured that the model could generalize well by preventing both underfitting and overfitting.

Once the best combination of parameters was identified, the optimal SVM model was fitted on the training data and subsequently evaluated on the test set. The evaluation was performed using classification accuracy, providing a quantitative measure of the model's predictive performance. The final trained SVM model was serialized and saved using Joblib, allowing for efficient reuse during inference.

```
# Load audio features from CSV file
csv_path = os.path.abspath(r"C:\Users\Isaiah Ho\Desktop\Information System Year 4\TMF4913 Sem 1 2425 (G01) Final Year P
data = pd.read_csv(csv_path)

# Separate features and labels
X = data.iloc[:, 1:]  # Drop 'filename'
y = data['filename'].apply(lambda x: os.path.splitext(x)[0])  # Use full car name without extension

# Encode labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# Save the LabelEncoder for later use
joblib.dump(label_encoder, "label_encoder.pkl")

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42, stratify=y_encoded)

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train SVM Model with GridSearchCV
print("\nTraining SVM Model with GridSearchCV...")
param_grid = {
    'C': [0.1, 1, 10, 20],
    'gamma': ['scale', 'auto', 0.001, 0.01, 0.1, 1],
    'kernel': ['rbf', 'linear', 'poly']
}

grid_search = GridSearchCV(SVC(), param_grid, cv=5, scoring='accuracy', verbose=2)
grid_search.fit(X_train, y_train)
```

Figure 4.5: SVM Model

4.3.4 Model Training – Random Forest (RF)

In addition to the Support Vector Machine (SVM) model, a Random Forest (RF) classifier was implemented to evaluate the performance of an ensemble-based approach for the car audio classification task. The Random Forest algorithm is well-suited for

classification problems involving high-dimensional feature spaces, such as those derived from audio signals. It operates by constructing an ensemble of decision trees during training and aggregating their predictions to produce a final classification output, thereby reducing variance and improving generalization.

For this project, the RF model was configured after multiple test runs which found out that 100 decision trees (n_estimators=100) and a fixed random seed (random_state=42) are able to give the best values. To rigorously assess the model's generalization capability, 5-fold cross-validation was conducted on the training set. This technique partitions the data into five equal subsets, iteratively using four folds for training and one for validation. Accuracy was selected as the performance metric, and the average cross-validation score was computed to provide an unbiased estimate of model performance across different data partitions.

The results from the 5-fold cross-validation yielded a set of accuracy scores for each fold, along with a mean accuracy value representing the overall performance of the RF classifier on unseen validation data. Following cross-validation, the RF model was retrained on the full training set to maximize the use of available labeled data. It was then evaluated on the separate test set to provide a final assessment of its predictive capability. The test accuracy was computed and compared to the results of other models to determine the effectiveness of the RF classifier in distinguishing between the car audio recordings.

This ensemble-based approach offered several advantages, including robustness to overfitting, interpretability of feature importance, and minimal requirement for hyperparameter tuning.

```
# Train Random Forest Model with 5-fold Cross Validation
print("\nTraining Random Forest Model with 5-Fold Cross Validation...")
start_time = time.time()
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_cv_scores = cross_val_score(rf_model, X_train, y_train, cv=5, scoring='accuracy', verbose=2)
rf_model.fit(X_train, y_train)
rf_time = time.time() - start_time

print("Random Forest 5-Fold CV Accuracies:", rf_cv_scores)
print("Random Forest Mean CV Accuracy:", np.mean(rf_cv_scores))
y_pred_rf = rf_model.predict(X_test)
print("Random Forest Test Accuracy:", accuracy_score(y_test, y_pred_rf))
print(f"Random Forest Training Time: {rf_time:.2f} seconds")
```

Figure 4.6: Random Forest Model

4.3.5 Model Training – Convolutional Neural Network (CNN)

In addition to traditional machine learning classifiers, a deep learning approach was explored using a Convolutional Neural Network (CNN) to directly model the complex patterns within the extracted audio features. CNNs are particularly effective at learning local patterns in structured input data, such as time-series or spectrogram-like audio representations, making them suitable for this task. The input to the CNN was the same 162-dimensional feature vector per audio sample, reshaped into a one-dimensional format suitable for 1D convolutional processing.

The initial CNN architecture was constructed using the TensorFlow Keras API. It comprised two convolutional layers with 32 and 64 filters respectively, each followed by a max-pooling layer to reduce dimensionality and control overfitting. The resulting feature maps were flattened and passed through a dense fully connected layer with 64 units and ReLU activation, followed by a final output layer with a softmax activation function to predict the probability distribution over the car classes. The model was compiled using the Adam optimizer and trained with a categorical cross-entropy loss function appropriate for

multi-class classification. Training was conducted over 25 epochs using a batch size of 32, with validation performed on a held-out test set to monitor performance and mitigate overfitting.

To further enhance the model's performance, hyperparameter tuning was carried out using Keras Tuner with a random search strategy. The search space included key parameters such as the number of convolutional filters, kernel sizes, dense layer units, dropout rates, and learning rates. The tuner evaluated multiple configurations based on validation accuracy and identified the best-performing set of hyperparameters. These optimal settings were then used to construct a refined CNN model.

To rigorously validate the performance of the CNN, a 5-fold cross-validation was performed using the tuned architecture. The dataset was split into five stratified folds to ensure class distribution consistency. For each fold, the CNN model was re-initialized and trained on four folds while validated on the remaining fold. This process was repeated across all five folds, and the validation accuracies were recorded. The average of these accuracies provided a reliable estimate of the model’s generalization ability across unseen data. Finally, the trained CNN model was saved in the Keras .keras format, allowing for future deployment and reuse.

```
# Train CNN Model with Progress Bar
print("\nTraining CNN Model...")
cnn_model.fit(X_train_cnn, y_train, epochs=25, batch_size=32, validation_data=(X_test_cnn, y_test), verbose=1)

# Hyperparameter Tuning for CNN Model
def build_model(hp):
    model = keras.Sequential()
    model.add(layers.Input(shape=(X_train.shape[1], 1)))
    model.add(layers.Conv1D(filters=hp.Choice('filters', [32, 64, 128]), kernel_size=hp.Choice('kernel_size', [3, 5, 7]), activation='relu'))
    model.add(layers.MaxPooling1D(pool_size=2))
    model.add(layers.Conv1D(filters=hp.Choice('filters_2', [64, 128, 256]), kernel_size=hp.Choice('kernel_size_2', [3, 5]), activation='relu'))
    model.add(layers.MaxPooling1D(pool_size=2))
    model.add(layers.Flatten())
    model.add(layers.Dense(units=hp.Choice('dense_units', [64, 128, 256]), activation='relu'))
    model.add(layers.Dropout(rate=hp.Choice('dropout', [0.2, 0.3, 0.5])))
    model.add(layers.Dense(len(np.unique(y_encoded)), activation='softmax'))
    model.compile(optimizer=keras.optimizers.Adam(learning_rate=hp.Choice('learning_rate', [1e-2, 1e-3, 1e-4])),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

Figure 4.7: CNN Model

4.4 Web-based System

Establishing an appropriate development environment is always a crucial step in any software project, as it ensures the availability of the necessary tools and frameworks for effective development and deployment. The following subsections provide a comprehensive guide to installing and configuring these essential components to create a robust and efficient development environment. This section outlines the comprehensive development process of the web-based system, detailing the setup of the frameworks and tools utilized, the integrated development environment (IDE) employed, the model inference, the design and implementation of the graphical user interface (GUI), and the deployment procedures.

4.4.1 Django Framework

The first step in setting up the environment is installing Django Framework onto the personal laptop. The process first requires the installation of Python which at the point of writing this report and considering the compatibility of other dependencies is the version of

3.12.9 by downloading the windows executable from the Python website and installing it. The next step is to install Django Framework through the Windows Terminal by copying and pasting the command: python -m pip install Django.

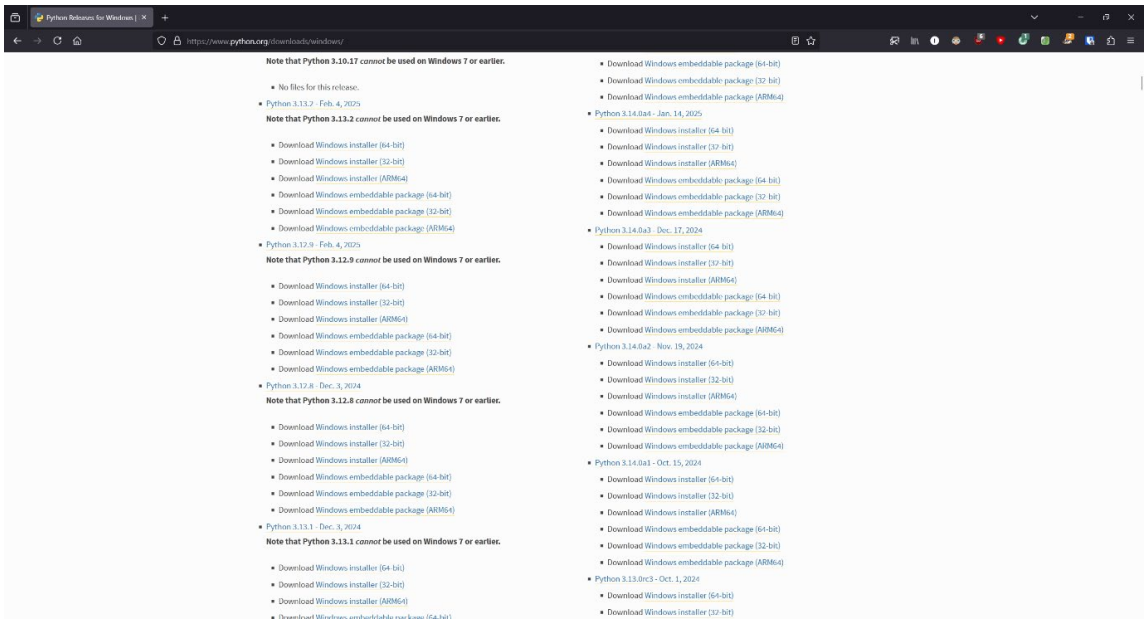


Figure 4.8: The webpage for downloading Python

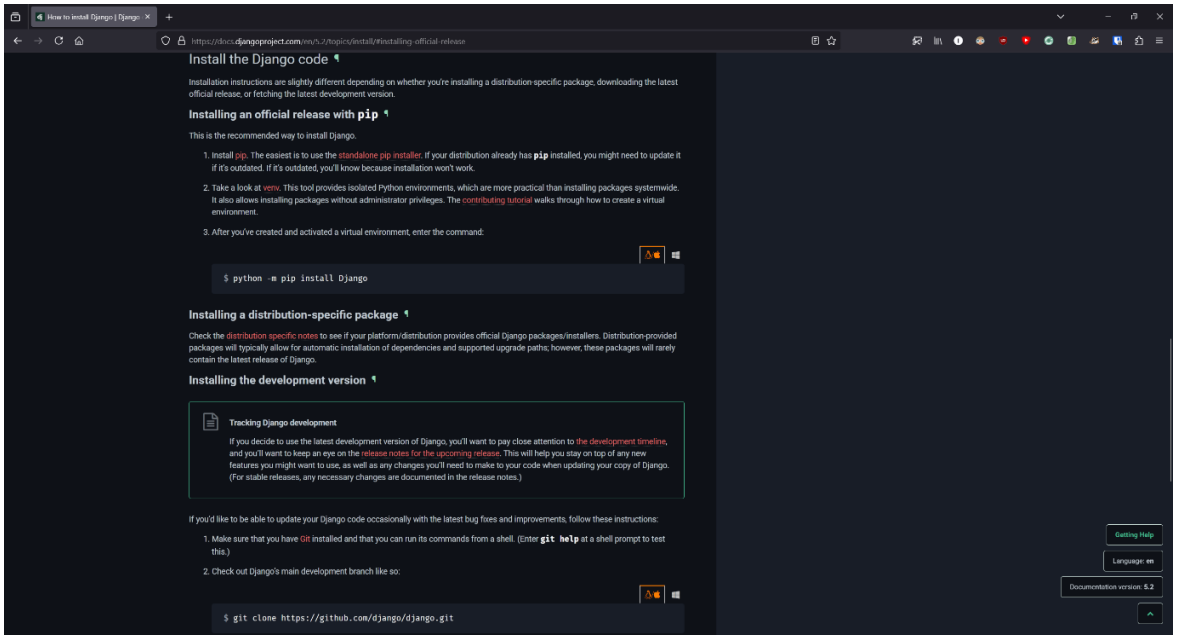


Figure 4.9: Django Framework installation command

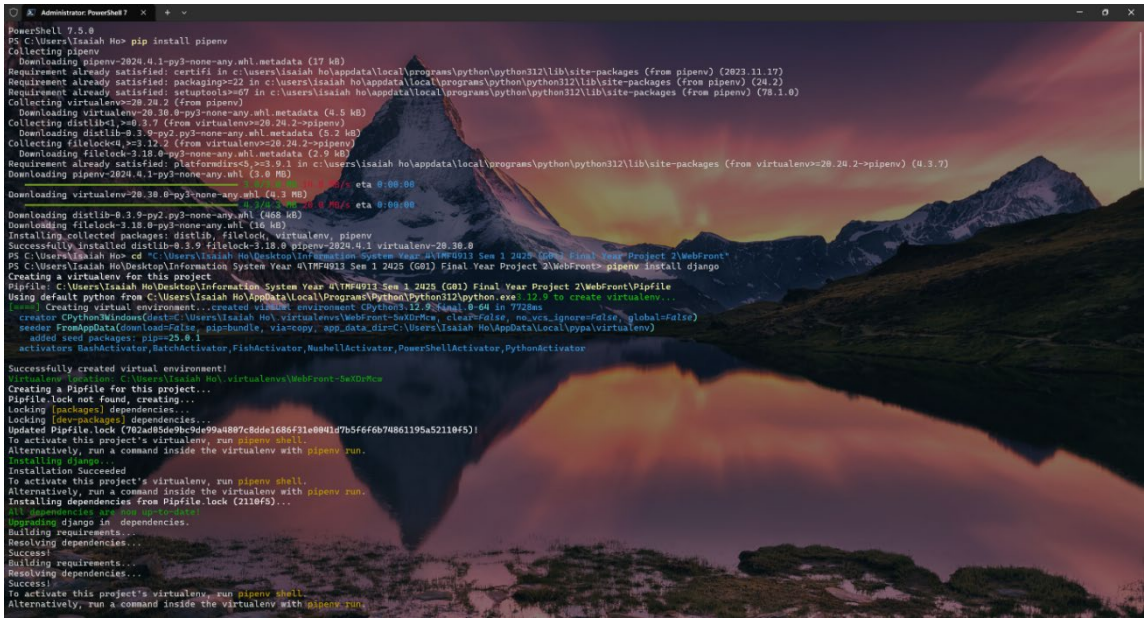


Figure 4.10: Creating a test Django framework project

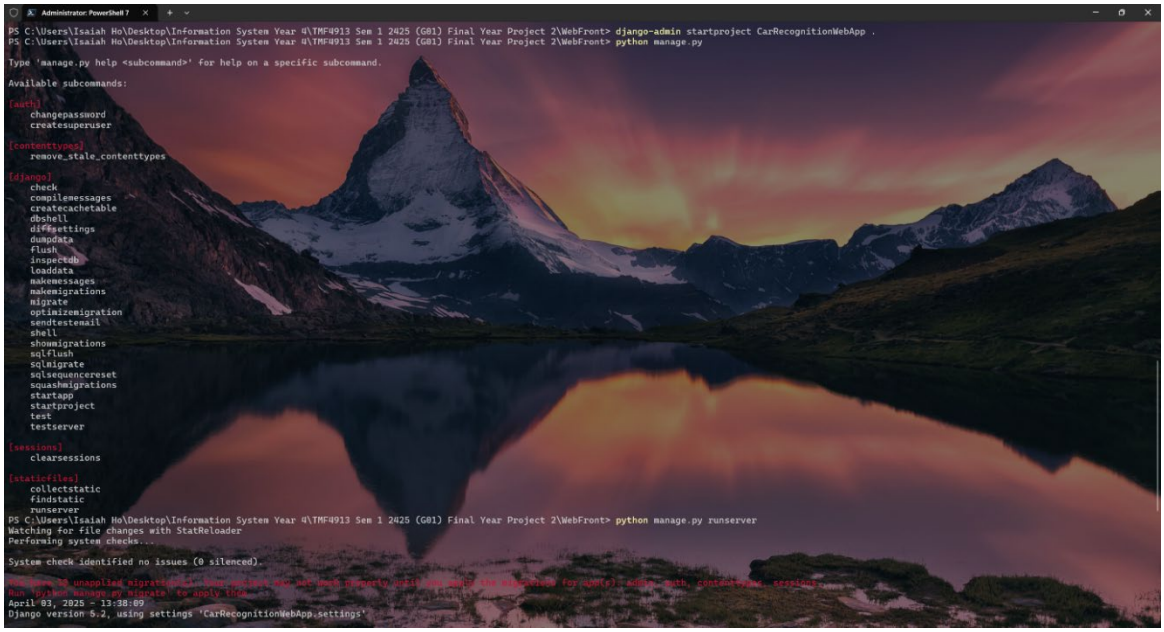


Figure 4.11: Running the test Django Framework

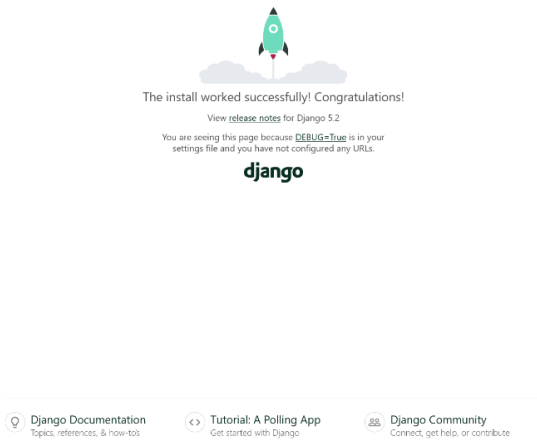


Figure 4.12: The successful installation of the test Django framework

One of the main reason Django framework was chosen and not Flask and FastAPI is its easy to have a website running in a short time which is crucial to this Final Year Project.

With Django and all the required dependencies installed, we are able create a web application with a strong back-end capable of handling user input, processing it with machine learning techniques, and delivering the results back to the user. The combination of Django's robust features and Python's large libraries of machine learning algorithms ensured that the entire workflow—from data processing to user interaction—ran smoothly, making the system both effective and user-friendly.

4.4.2 Visual Studio Code

The Integrated Development Environment (IDE) selected for this Final Year Project is Microsoft Visual Studio Code (VS Code). This choice was made due to its lightweight nature, robust support for a wide range of programming languages, and its rich ecosystem of extensions, all of which contribute to a streamlined and efficient development process. VS Code offers built-in support for Python and HTML/CSS, along with customizable features that cater perfectly to the needs of a web-based project incorporating machine learning and Django.

The installer for Visual Studio Code was downloaded directly from the official website and installed on a personal laptop to serve as the primary development machine. After installation, the Python extension was added to enable support for writing, running, and debugging Python code seamlessly within the editor. To ensure isolation and manage dependencies effectively, a virtual environment was created within VS Code. This environment was specifically set up for Django, allowing for cleaner package management and avoiding conflicts with system-wide Python packages.

Subsequently, a new Django project named "WebFront" was initialized within the virtual environment. This project acts as the central development framework and server for the web-based article summarization system. It provides the structure for developing both the back-end logic and the front-end interface. Mandatory libraries that are required for running the machine learning models such as NumPy, Pandas, Librosa, Scikit-learn, TensorFlow and Keras were also installed to ensure proper execution.

To further improve the development experience, several additional VS Code extensions were installed. These include tools for linting, code formatting, template support, and Copilot integration. Collectively, these enhancements significantly boost productivity, reduce errors, and provide a more intuitive and responsive development workflow.

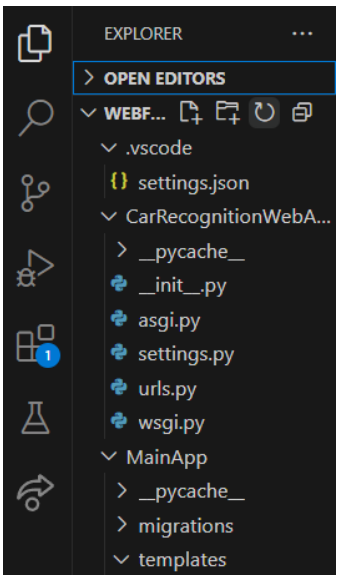


Figure 4.13: Screenshot of Django Project in Visual Studio Code

4.4.3 Model Inference

The views.py file serves as the backend logic of the Django-based web application, responsible for managing user interactions, processing audio input, invoking trained machine learning models, and returning predictions to the frontend interface. Upon server initialization, several critical components are loaded into memory, including the pre-trained Convolutional Neural Network (CNN) model saved in Keras format, as well as serialized Random Forest and Support Vector Machine (SVM) models using joblib. Additionally, a *LabelEncoder* is loaded to decode numeric predictions into readable class labels, and a StandardScaler is used to normalize input features consistently with the preprocessing pipeline applied during model training. These resources are loaded once to ensure efficient and consistent inference throughout the application's runtime.

When a user uploads an audio file via the frontend interface, the *upload_audio* function is triggered. This function handles the POST request, saves the uploaded file temporarily using Django's default_storage, and passes the file path to the core prediction function. The file is then processed by *predict_car*, a function that orchestrates the end-to-end inference process. This function first invokes extract_features, which performs comprehensive preprocessing and feature extraction using the librosa library. The audio signal is loaded and trimmed to remove silent segments, enhancing the relevance of extracted features. Several audio descriptors are computed, including Mel-Frequency Cepstral Coefficients (MFCCs), zero crossing rate, spectral centroid, chroma features, tonal centroid features (tonnetz), and log-Mel spectrograms. These features are concatenated into a single fixed-length vector and normalized to align with the training data distribution.

Following feature extraction, the vector is reshaped to accommodate the input requirements of each model— (1, 161) for traditional machine learning models and (1, 161, 1) for the CNN. The processed input is then fed into all three models sequentially. The CNN outputs a probability distribution over all possible car classes, from which the class with the highest probability is selected. The Random Forest and SVM models directly produce class label predictions. Each numeric prediction is decoded into a human-readable label using the label encoder. The predictions from all three models are stored in a dictionary and returned to the *upload_audio* function, which saves them in the session context before redirecting the user to the results page.

Finally, the *results_page* function retrieves the prediction results and the original file name from the session and renders them using the results.html template. This provides the user with a clear and informative summary of the classification outcome.

```python
def predict_car(audio_path):
    # Extract features
    features = extract_features(audio_path)
    print(f"Features shape for CNN: {features.shape}")  # Debug: Check feature shape for CNN

    # CNN Prediction
    cnn_pred_index = int(np.argmax(cnn_model.predict(features)))
    cnn_pred_label = label_encoder.inverse_transform([cnn_pred_index])[0]
    print(f"CNN Prediction: {cnn_pred_label}")  # Debug: Check CNN prediction

    # Reshape features for RF and SVM models
    features_flat = features.reshape(1, -1)  # Shape (1, 161)

    # Random Forest Prediction
    rf_pred_index = int(rf_model.predict(features_flat)[0])
    rf_pred_label = label_encoder.inverse_transform([rf_pred_index])[0]
    print(f"Random Forest Prediction: {rf_pred_label}")  # Debug: Check RF prediction

    # SVM Prediction
    svm_pred_index = int(svm_model.predict(features_flat)[0])
    svm_pred_label = label_encoder.inverse_transform([svm_pred_index])[0]
    print(f"SVM Prediction: {svm_pred_label}")  # Debug: Check SVM prediction

    # Return predictions
    predictions = {
        "CNN_Model": cnn_pred_label,
        "Random_Forest_Model": rf_pred_label,
        "SVM_Model": svm_pred_label
    }
    return predictions
```

Figure 4.14: Model Inference

4.4.4 Graphical User Interface - GUI

The Car Recognition WebApp is an advanced, web-based intelligent system designed to accurately classify cars using machine learning algorithms. Built with the Django framework, a robust and high-level Python web framework, HTML and CSS with design following modern skeuomorphism, the application offers a seamless and user-friendly interface, making it easy for users to upload audio files and receive accurate car classifications.

Upon uploading an audio file—either by simple file selection or by dragging and dropping—the system employs a variety of powerful machine learning techniques to process and analyse the data. The web app supports multiple algorithms, including Support Vector Machines (SVM), Random Forest (RF), and Convolutional Neural Networks (CNN), each tailored to provide optimal classification results. This ensures high accuracy and reliability when classifying different car sounds.

The web app is designed to be highly responsive, making it accessible across a wide range of devices—from desktops to mobile phones—ensuring a smooth user experience regardless of the device used.

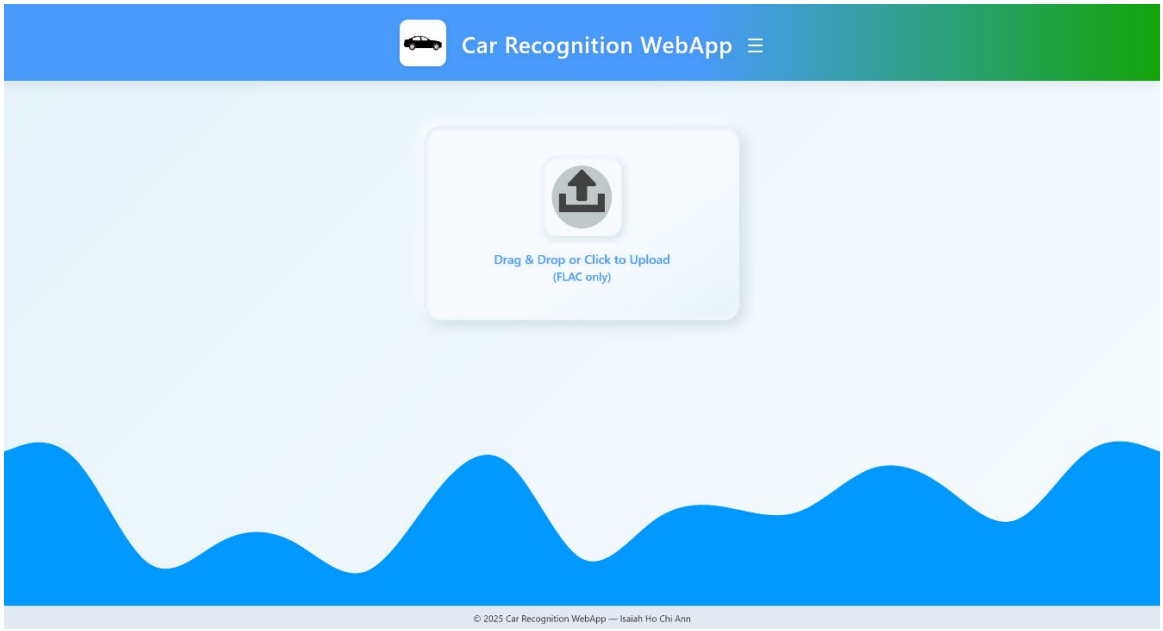


Figure 4.15: Main Page

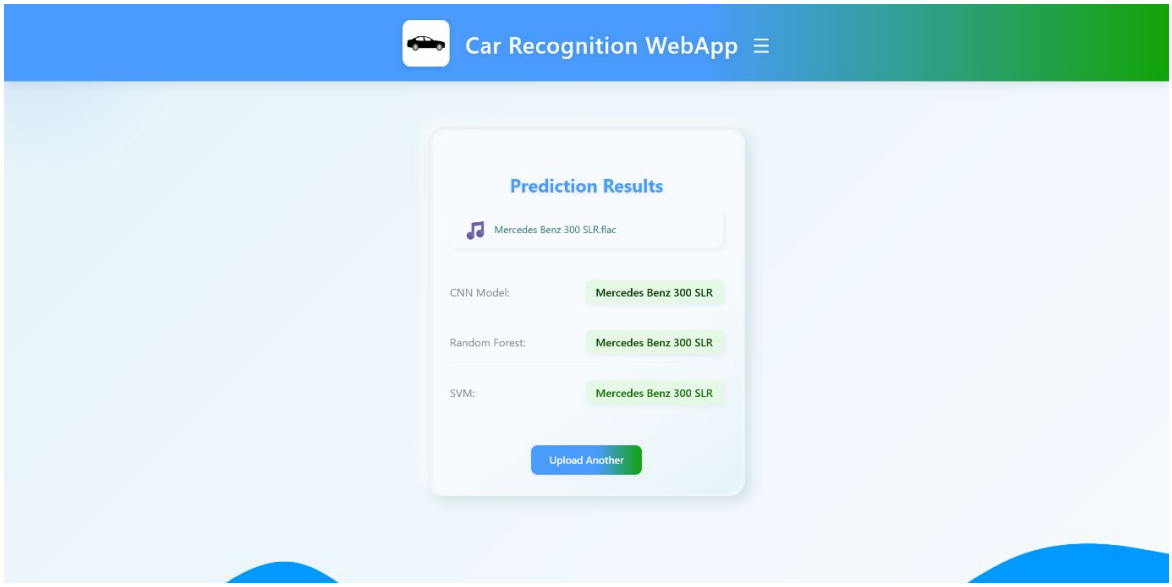


Figure 4.16: Results Page

4.4.5 Deployment

To run the Django-based Car Recognition Web Application, the user must first ensure that Python—preferably version 3.12, as specified in the Pipfile—is installed on the system, along with pip for package management. After navigating to the project directory via the terminal, the user should install the necessary dependencies by executing “pip install -r requirements.txt”, which will install all required libraries such as librosa, numpy, scikit-learn, tensorflow, keras-tuner and more. These packages are essential for audio preprocessing, model training, and inference as defined in the project’s scripts.

It is also necessary to confirm that all machine learning models (cnn_model.keras, rf_model.pkl, svm_model.pkl, and label_encoder.pkl) and the corresponding scaler (scaler.pkl) have been pre-trained and are saved in the correct directory structure, as referenced in the backend code. Once the environment is properly configured, the Django project can be initialized by running python manage.py migrate to set up the required database schema.

The development server can then be launched locally in your specified terminal of choice using the python manage.py runserver command. By opening a web browser and navigating to http://127.0.0.1:8000/, users will access the main interface of the application. This page allows for the upload of .flac audio files containing car engine sounds. Upon upload, the system processes the audio by extracting relevant features and passing them through the pre-trained machine learning models. The predicted car model results are then displayed on a user-friendly results page. By executing these instructions, the web-based system is now successfully launched and access is now available via the designated IP address and port.

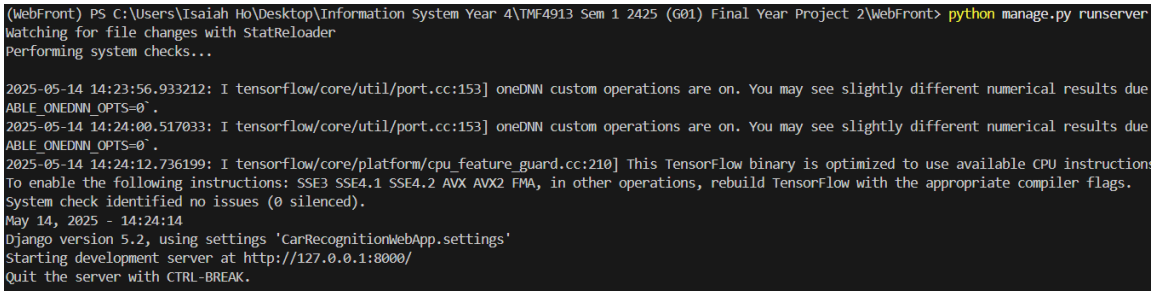


Figure 4.17: Running the web-based project

4.5 Summary

This chapter detailed the implementation of the machine learning algorithms and also the web-based stock car model recognition system using machine learning and acoustic analysis. The ML implementation covers the key steps involved in the implementation of three machine learning algorithms, Support Vector Machine (SVM), Convolutional Neural Networks (CNN), and Random Forest (RF) to classify and analyse the audio features. The Web-based system then began by establishing the development environment using Django and Visual Studio Code on a personal laptop, with all necessary dependencies configured for Python-based audio classification. The system's core modules included the front-end GUI, an audio data augmentation pipeline, preprocessing and feature extraction using MFCCs, and the integration of machine learning models, SVM, Random Forest, and CNN into the web framework. The user interface was designed for ease of use and responsiveness, supporting intuitive audio file uploads and delivering accurate classification results. The system was deployed locally, ensuring users could access its functionality through a web browser. All models and supporting files were organized and loaded effectively to support inference. In general, the chapter illustrates the transformation of theoretical machine learning concepts into a functional, user-ready web application.

Chapter 5: Results

5.1 Introduction

This chapter presents a comprehensive examination of the evaluation, testing, and outcomes of the developed system. It starts by highlighting and explaining the significance of evaluation and testing in determining the system's effectiveness and overall functionality, then end with the results stating what the system has successfully achieved and also what it hasn't achieved.

5.2 Model Evaluation

This subchapter presents the evaluation of three machine learning models—Support Vector Machine (SVM), Random Forest (RF), and Convolutional Neural Network (CNN)—applied to the task of car model classification using car sound features. All models were trained on standardized Mel-frequency cepstral coefficients (MFCCs), Zero Crossing Rate (ZCR) Spectral Centroid, Log-Mel Spectrogram, Chroma Features and Tonnetz Features extracted from the audio recordings. The evaluation was conducted using 5-fold cross-validation (CV) as the primary validation method. The goal was to assess each model's performance in terms of accuracy, hyperparameter choices, training times, generalizability, and consistency.

5.2.1 Support Vector Machine (SVM)

The SVM model was tuned using GridSearchCV to optimize three key hyperparameters: C, gamma, and kernel which began with a pipeline with four values of C (0.1, 1, 10, 20), six options for gamma (scale, auto, 0.001, 0.01, 0.1, 1), and three kernel types (rbf, linear, poly). After exhaustive tuning, the best configuration found was a linear kernel with C = 10 and gamma = scale. Under these settings, the model achieved a mean 5-fold CV accuracy of 95.72%, the highest among the three candidates. This strong cross-validation result indicates that the SVM is highly effective at generalizing across training folds. The SVM's total training time was 110.29 seconds, which reflects both the cost of grid-search and fitting on the full training set. This runtime is moderate: slower than Random Forest but markedly faster than the CNN.

Table 5.1: SVM Results

Best SVM Parameters	**SVM Mean CV Accuracy**	**SVM Training Time**
'C': 10, 'Gamma': 'Scale', 'Kernel': 'Linear'	95.72%	110.29 seconds

5.2.2 Random Forest (RF)

The Random Forest model was trained with 100 estimators and evaluated using both 5-fold CV and a separate test set. It used bootstrapped sampling and default feature splits. Rather than tuning many hyperparameters, it relied on ensemble's inherent regularization. The 5-fold CV results showed individual fold accuracies of 93.10%, 93.49%, 91.57%, 93.49%, and 93.46%, yielding a mean cross-validation accuracy of 93.02%. Although its CV

accuracy was the lowest among the three, it achieved the highest test accuracy of 96.94%, highlighting its strong generalization capability on unseen data. The narrow gap between CV and test performance also implies minimal overfitting. The Random Forest model also had the shortest training time at 81.99 seconds, making it an efficient option when balancing performance and training speed.

Table 5.2: RF Results

Random Forest 5-Fold CV Accuracies	**Random Forest Mean CV Accuracy**	**Random Forest Test Accuracy**	**Random Forest Training Time**
93.10%, 93.49%, 91.57%, 93.49%, 93.46%	93.02%.	96.94%,	81.99 seconds

5.2.3 Convolutional Neural Network (CNN)

The CNN architecture comprised two 1D convolutional layers (32 filters with kernel size 3, then 64 filters), each followed by max-pooling, a dense ReLU layer of 64 units, and a SoftMax output layer. Hyperparameters were further optimized via Keras Tuner's random search, exploring Conv filters: 32, 64, 128; Kernel sizes: 3, 5, 7; Dense units: 64, 128, 256; Dropout: 0.2, 0.3, 0.5; and Learning rates: 1e-2, 1e-3, 1e-4. The CNN was evaluated using stratified 5-fold cross-validation. The individual validation accuracies across the five folds were 95.11%, 96.32%, 95.40%, 95.09%, and 94.48%, resulting in a second-best mean CV accuracy of 95.28%. The best single-fold accuracy reached 96.32%, close to that of the SVM. Additionally, hyperparameter tuning using Keras Tuner selected the optimal configuration as 32 and 64 filters, a kernel size of 5, 256 dense units, a dropout rate of 0.2, and a learning

rate of 0.001 using the Adam optimizer. The total training time for the CNN was 2 minutes and 39 seconds, the longest among the three models, which is expected given its higher computational complexity.

Table 5.3: CNN Results

Best CNN Parameters	**CNN 5-Fold CV Accuracies**	**CNN Mean CV Accuracy**	**CNN Training Time**
'filters': 32, 'kernel_size': 5, 'filters_2': 64, 'kernel_size_2': 5, 'dense_units': 256, 'dropout': 0.2, 'learning_rate': 0.001	95.11%, 96.32%, 95.40%, 95.09%, 94.48%,	95.28%	2 minutes 39 seconds

5.3 Testing Unseen Data

This section details the evaluation of the models to new unseen data, obtained in the same way from the previous data. It serves as a way to determine if the models can successfully make accurate predictions beyond its training examples. To further assess the system's generalizability and robustness, three entirely new datasets—referred to as Unseen Data 1, 2, and 3—were evaluated again. When presented with three entirely new datasets ("Unseen Data 1–3"), all models experienced a modest accuracy decline, as might be expected with out-of-sample recordings. Here, the SVM again led with a mean unseen accuracy of 94.28 %, while the CNN achieved 94.28 % and RF trailed at 93.85 %. These results demonstrate that the SVM generalizes better to new acoustic variations, though all classifiers maintain performance well above 93 % accuracy.

Table 5.1: continued

Table 5.4: Accuracy Summary on Unseen Datasets

Dataset	**CNN Accuracy**	**SVM Accuracy**	**RF Accuracy**
Unseen Data 1	94.42 % (220 / 233)	93.13 % (217 / 233)	94.42 % (220 / 233)
Unseen Data 2	94.42 % (220 / 233)	95.28 % (222 / 233)	92.70 % (216 / 233)
Unseen Data 3	93.99 % (219 / 233)	94.42 % (220 / 233)	94.42 % (220 / 233)

The consistency and minimal performance degradation of the SVM and CNN models suggest they are the preferred classifier for live use, offering near-real-time, high-accuracy predictions across diverse engine sounds. The slightly lower yet still strong results of the RF model indicate that further enhancements—such as data augmentation or ensemble methods—could close the remaining gap.

5.4 Web Based System Evaluation with Real World Audio Data

Following the evaluation of the machine learning models using pre-segmented and clean datasets, a final phase of testing was conducted using real-world audio samples to assess the practical performance and limitations of the system in uncontrolled environments. The aim of this test was to determine how well the system could recognize actual car models

from authentic, non-simulated sound recordings. A total of 20 external car sound samples were sourced for this purpose.

These real-world audio samples were obtained from various YouTube videos, as no public or standardized external car sound database currently exists. The decision to use YouTube as a source was based on the lack of accessible datasets that matched the system's input requirements—specifically, high-quality, externally recorded exhaust and engine sounds. However, this approach immediately presented several critical challenges.

The first major issue was identifying suitable recordings that featured the **external** sound of a vehicle. Many car-related videos focus on in-cabin experiences or engine bay recordings, which differ acoustically from external noises. Internal recordings, while still capturing engine revs and acceleration, contain reflections and resonances from within the car cabin and do not match the sound profile the system was trained to recognize. Therefore, these internal sounds are fundamentally incompatible with the WebApp, which expects the clear, unfiltered acoustic signature of an external engine and exhaust note.

Even after identifying videos with appropriate external perspectives, the **second significant challenge** was the quality and consistency of the audio recordings. Real-world recordings found online are often plagued by background noise, speech overlays, wind interference, temperature, ambient city or highway noise, and inconsistent mic quality. Many videos include commentary, music, or environmental sounds, making it difficult to isolate the car's exhaust note from the surrounding audio environment. Furthermore, the compression applied by platforms such as YouTube degrades audio fidelity, and this loss of

frequency detail can affect the system's ability to accurately analyse the sound's spectral features.

Another contributing factor to classification difficulty was the **variation in microphone types and recording techniques** used in different videos. Some videos were recorded with high-end DSLR microphones or professional-grade audio equipment, while others used basic smartphone microphones, each capturing sound differently. Inconsistent mic distances, gain settings, and post-processing also impacted the audio profiles. These disparities introduced inconsistencies in pitch clarity, amplitude, and frequency resolution the key features used by the machine learning models for classification.

In addition to recording hardware, the **recording environment** itself had a noticeable effect on the outcome. For example, sounds recorded in open parking lots, garages, racetracks, or residential streets have different reverb characteristics, noise floors, and sound propagation behaviours. These environmental differences alter the spectral and temporal characteristics of the sound input, further reducing compatibility with the training data, which was collected under controlled conditions.

Furthermore, a usually unnoticed factor that emerged during testing was the **variation in driver behaviour**. The way a driver accelerates, revs the engine, or shifts gears can dramatically change the car sound. For example, a high-rev launch produces a sound pattern distinct from gentle acceleration or cruising. These driving style differences introduced variability that the model had not encountered during training, where all samples followed a standardized throttle input and rev cycle.

As a result of all these factors, the WebApp was unable to successfully classify all of the 20 real-world audio recordings. The classification results were either completely incorrect or the system failed to generate any meaningful prediction. The background interference, inconsistent sound quality, and environmental noise heavily impacted the system's ability to extract relevant features for reliable classification. This performance gap highlights a limitation of the current implementation, which was optimized for clean, consistent, repeatable, high-quality datasets.

5.5 System Usability Scale (SUS)

The System Usability Scale (SUS) is a widely recognized tool designed to evaluate the usability of a system, product, or service. It provides a standardized method for assessing the ease of use, efficiency, and overall user satisfaction. Comprising 10 statements, the SUS captures user perceptions regarding various usability factors such as learnability, task completion, and the overall user experience. The scale generates a numerical score that ranges from 0 to 100, with higher scores indicating more favourable usability perceptions. It is frequently utilized in usability testing to provide valuable insights into how users interact with a system, helping to identify areas for improvement in user interface design, functionality, and overall user experience. A detailed view of SUS questionnaire is available in Appendix C.

Respondent's Answers:

The SUS questionnaire with 10 questions was created in Google Form and sent out to the respondents to gauge their opinions. Show below are the responses from 25 participants.

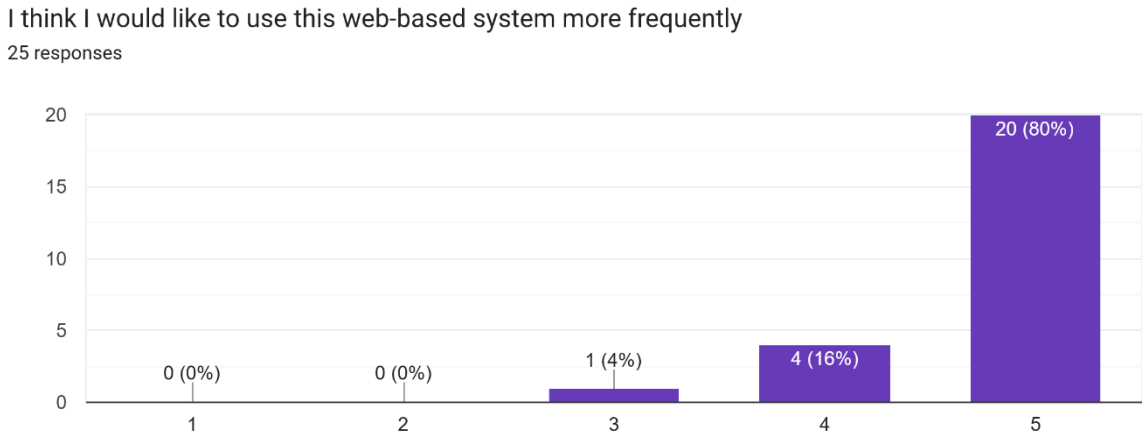


Figure 5.1: SUS Question 1 Results

In Figure 5.1 with the question, "I think I would like to use this web-based system more frequently", 80% of respondents selected 5, 16% selected 4 while only 1 selected 3, indicating participants will use the web-based system again.

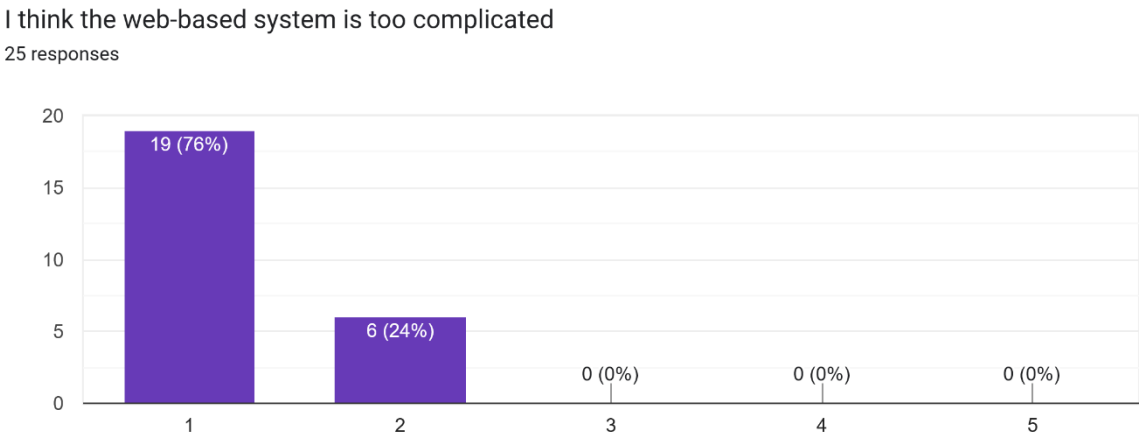


Figure 5.2: SUS Question 2 Results

In Figure 5.2 with the question, “I think the web-based system is too complicated”, 76% of respondents answers 1 and 24% answered 2. This shows that the web-based system is not complicated.

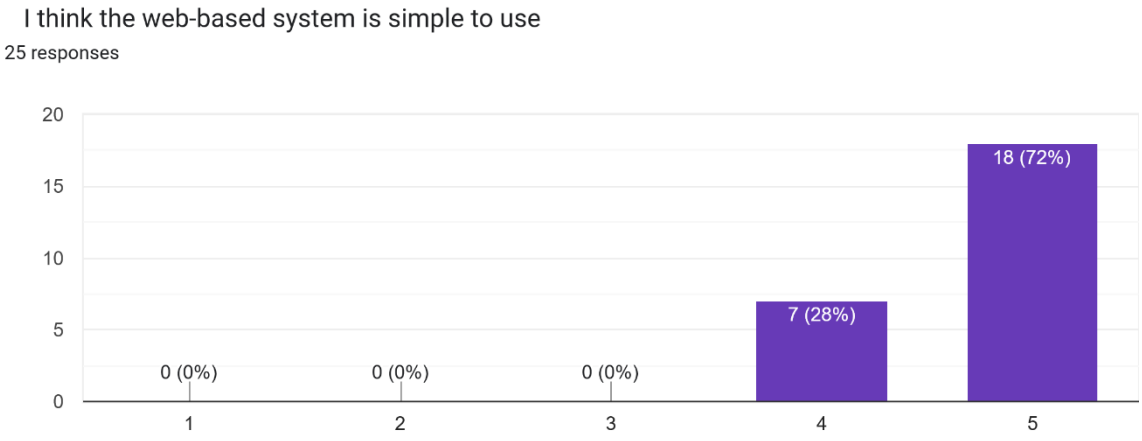


Figure 5.3: SUS Question 3 Results

In Figure 5.3 with the question, “I think the web-based system is simple to use”, 72% of people selected 5 and 28% selected 4. This indicated that the web-based system is generally simple to use.

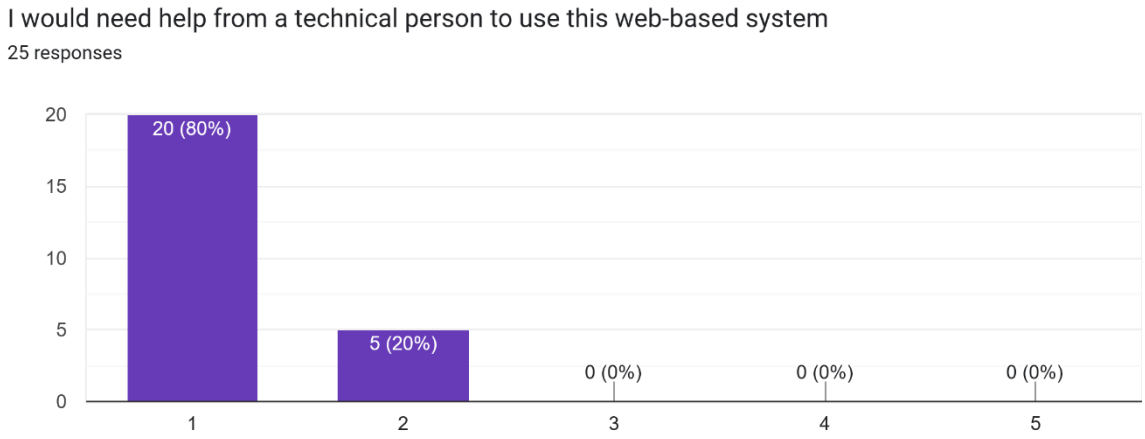


Figure 5.4: SUS Question 4 Results

In Figure 5.4 with the question, “I would need help from a technical person to use this web-based system”, 80% of respondents answered 1 and only 20% answered 1, indicating that the participants will not need technical help.

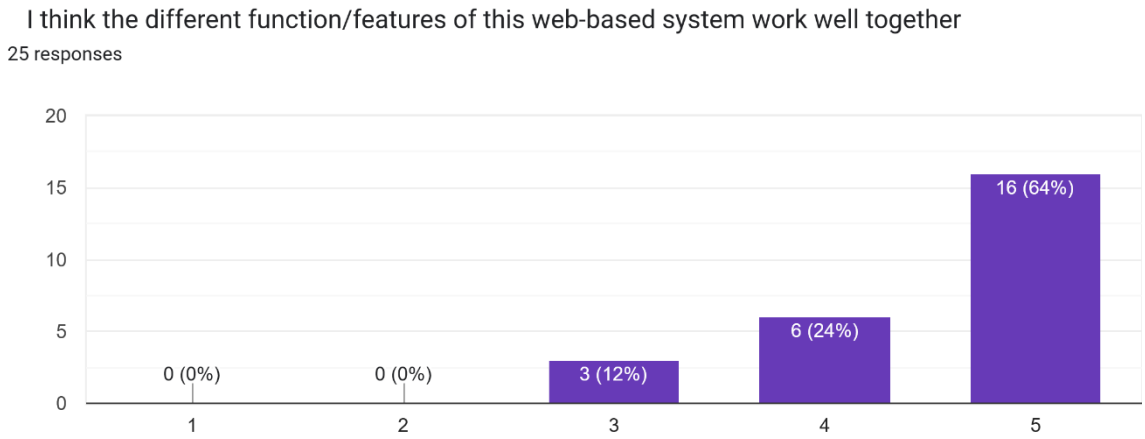


Figure 5.5: SUS Question 5 Results

In Figure 5.5 with the question, “I think the different function/features of this web-based system work well together”, 64% of respondents answered 5, 24% of respondents answered 4, and 12% answered 3, demonstrating that the different functions do work well together.

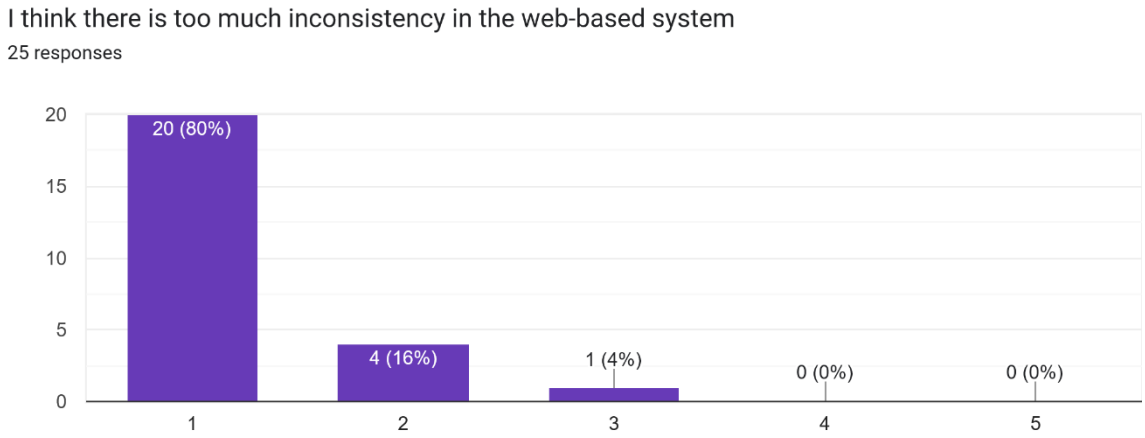


Figure 5.6: SUS Question 6 Results

In Figure 5.6 with the questions, “I think there is too much inconsistency in the web-based system”, 80% of the respondents selected 1, 16% selected 2 and only 4% selected 3, indicating that the web based system is consistent.

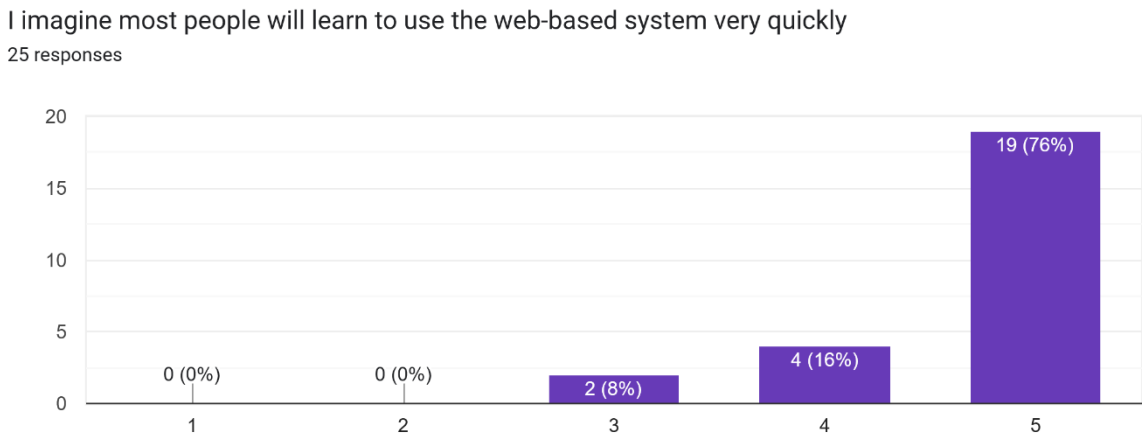


Figure 5.7: SUS Question 7 Results

In Figure 5.7 with the question, “I imagine most people will learn to use the web-based system very quickly”, 76% of the respondents answered 5, 16% of the respondents answered 4 while only 8% chose 3. This indicates that a majority of respondents learned to use it quite quickly.

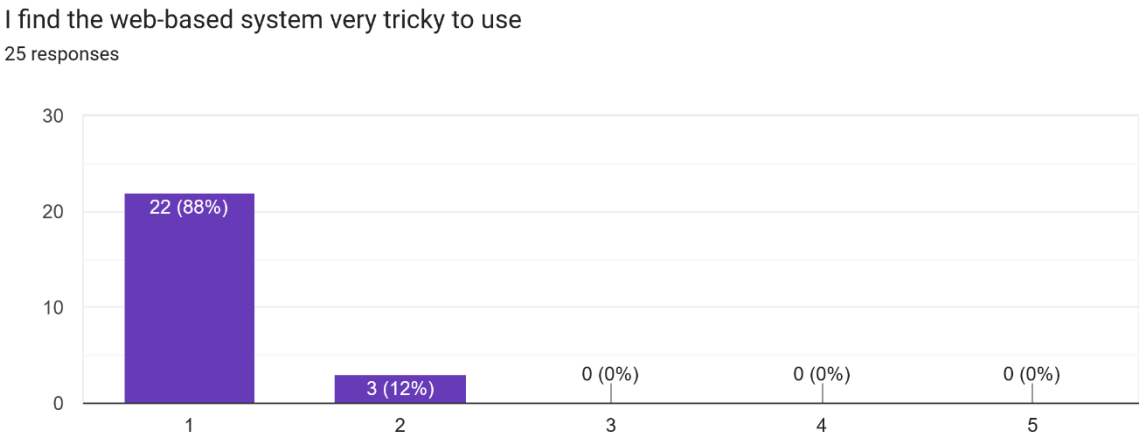


Figure 5.8: SUS Question 8 Results

In Figure 5.8 with the question, “I find the web-based system very tricky to use”, 88% of respondents chose 1 and 12% chose 2, signifying that most participants find the web-based system not tricky.

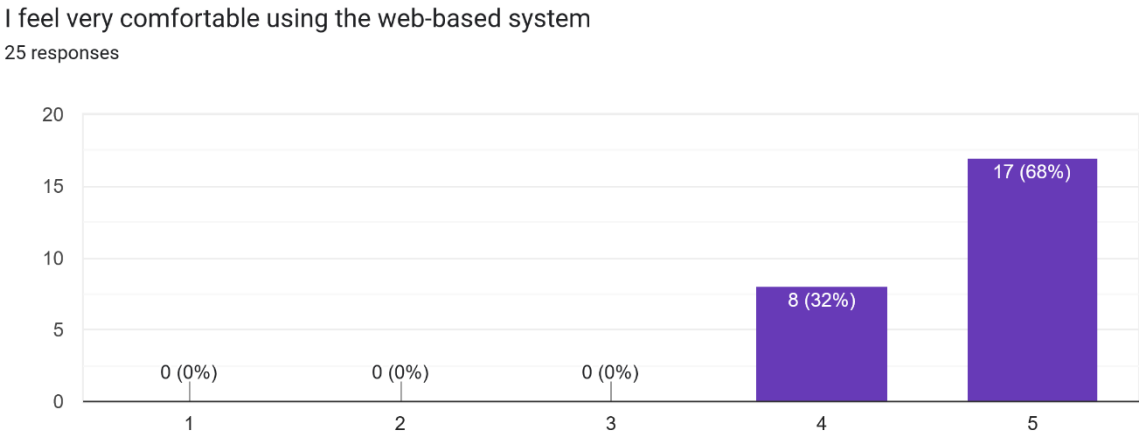


Figure 5.9: SUS Question 9 Results

In Figure 5.9 with the question, “I feel very comfortable using the web-based system”, 68% of respondents selected 5 and 32% of respondents selected 4. This indicates that most respondents are indeed comfortable with the web-based system.

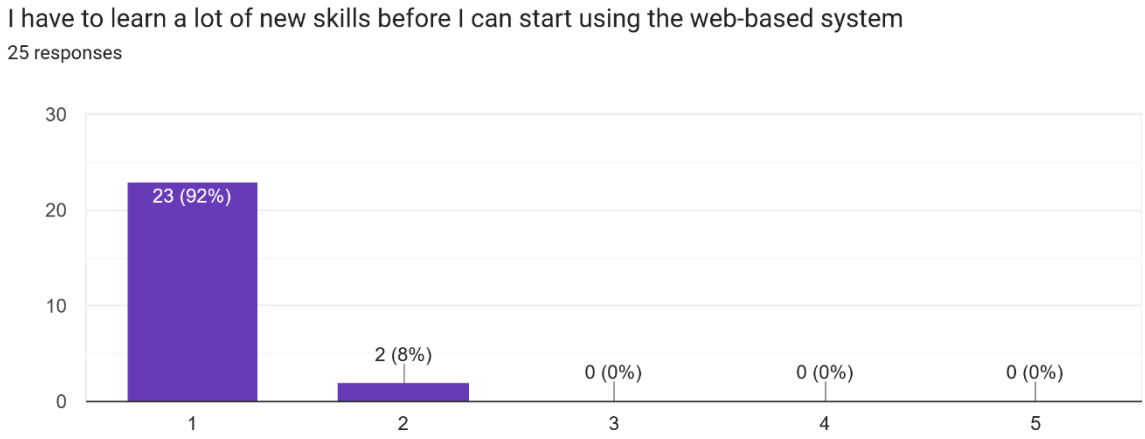


Figure 5.10: SUS Question 10 Results

In Figure 5.10 with the question, “I have to learn a lot of new skills before I can start using the web-based system”, 92% of respondents answered with 1 and only 8% answered with 4, indicating that most respondents are already skilful enough to use it.

5.6 SUS Results

This section details the results from the System Usability Scale (SUS) questionnaire, which was employed to evaluate the overall usability of the developed system. The responses were gathered from a group of 25 participants who interacted with the system. Each participant rated different aspects of the system's usability using a 5-point Likert scale ranging from Strongly disagree to Strongly agree.

Table 5.5: Participant's SUS Score

P/Q	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	SUS Score
P1	5	1	5	1	5	1	5	1	5	1	100.0
P2	5	1	4	1	4	1	4	1	5	1	92.5
P3	5	2	5	1	5	1	4	1	4	1	92.5
P4	5	1	5	1	4	1	5	1	5	1	97.5
P5	4	1	4	1	4	1	4	1	4	1	87.5
P6	5	1	5	1	5	1	5	2	4	2	92.5
P7	5	2	5	2	5	2	5	2	5	1	90.0
P8	4	2	4	2	4	2	4	2	4	2	75.0
P9	5	1	5	1	5	1	5	1	5	1	100.0
P10	5	1	5	1	5	1	5	1	5	1	100.0
P11	5	1	5	1	5	1	5	1	5	1	100.0
P12	4	2	5	2	3	3	3	1	4	1	75.0

Table 5.5: Continued

P13	**5**	**1**	**5**	**1**	**5**	**2**	**5**	**1**	**5**	**1**	**97.5**
P14	5	1	5	1	5	1	5	1	5	1	100.0
P15	5	2	5	1	5	1	5	1	5	1	97.5
P16	5	1	4	1	5	1	5	1	5	1	97.5
P17	5	1	5	1	5	1	5	1	5	1	100.0
P18	5	1	5	1	3	2	5	1	5	1	92.5
P19	5	1	5	1	5	1	5	1	5	1	100.0
P20	3	2	4	2	3	1	3	1	4	1	75.0
P21	4	1	4	1	5	1	5	1	5	1	95.0
P22	5	1	4	1	4	1	5	1	5	1	95.0
P23	5	1	5	2	4	1	5	1	4	1	92.5
P24	5	1	5	1	5	1	5	1	4	1	97.5
P25	5	1	5	1	5	1	5	1	5	1	100.0
Total											2342.50

$$\text{Overall SUS Score} = \frac{\sum_{i=1}^{n} \text{Individual SUS Score}_i}{n}$$

Where:

- ∑Individual SUS Scorei is the sum of all individual SUS scores.
- n is the total number of participants.

The overall SUS score for the web-based system is 2342.50/25 = 93.70

A System Usability Scale (SUS) score of 93.70 signifies an exceptionally high level of usability for the system. This score places the system well within the "Excellent" category on the SUS scale, which ranges from 0 to 100. Such a high score indicates that users find your system remarkably easy to use, highly intuitive, and are very likely to recommend it to others. The outcome is highly admirable, demonstrating a strong positive user experience.

5.7 Preferred Evaluation

The preferred evaluation is used to determine the end user satisfaction and usability scoring and opinion of the specified system. A combination of live feedback sessions and questionnaire was administered to friends and family, to create variability.

Per the live feedback sessions, a majority of the participants were generally very satisfied with the system. Early testers were however quick to point out that the UI/UX of the system's early design was too childish/ugly and "not that nice", therefore the system's design was overhauled to the current one and the feedback has been unanimously positive. The participants found the system to be easy to use, had minimal learning curve required to operate it once you first started using it. Functionality of the system was praised as it did was it promised to perform, no hassle and contributed to a consistent user experience from beginning to end. Furthermore, the participants were quite satisfied with the speed of the system as most runs only took less than 5 seconds to finish, indicating the system was lightning quick and fulfils modern user's requirements. There were however some complaints, one was the size of the upload button and the other one was the colour of the

font. These insights offer important feedback to evaluate the system's strengths and identify areas for improvement, helping to boost overall usefulness.

Based on the handed-out questionnaire, all participants agreed that the system was simple to use. However, only 8% believed that it required significant skill to use, suggesting that most users found it intuitive and easy to get started with. The system was also praised for its design and performed very responsively, with every respondent highlighting its speed and performance. Furthermore, most participants agreed that the system produced consistent and accurate results. As for the open-ended questions, responses were overwhelmingly positive, showing satisfaction with the system. These results were consistent with feedback received during the live session, reinforcing that the system is more than satisfactory.

The evaluation provided crucial insights into the system's strengths and areas for potential improvement. The feedback from both live sessions and the questionnaire closely aligned, confirming that the system meets or exceeds user expectations in key areas such as usability, design, and performance. Overall, the system is highly recommended and well-received, with strong potential for continued user satisfaction.

5.8 Summary

This chapter presented a thorough evaluation of the car model recognition system, highlighting its strengths, performance metrics, and areas requiring improvement. The evaluation began with a comparative analysis of three machine learning models—Support Vector Machine (SVM), Random Forest (RF), and Convolutional Neural Network (CNN)—

trained using audio-based features such as MFCCs, Log-Mel spectrograms, and tonal characteristics. Among the three, the SVM achieved the highest average 5-fold cross-validation accuracy of 95.72%, demonstrating strong generalization capability with efficient training time. The CNN followed closely with a 95.28% mean accuracy, while the Random Forest is trailing behind with a lower CV accuracy (93.02%).

The performance of the models was again put to the test and further validated in another trial. Accuracy evaluations across three unseen datasets showed that the SVM model again outperformed or equalled the others, achieving an overall accuracy of 94.28%. CNN and RF also performed well, with accuracies of 94.28% and 93.85%, respectively. These results confirmed that all three models are reliable, with SVM and CNN being the most consistent across diverse inputs.

To explore real-world applicability, 20 car sound clips were collected from YouTube and tested with the WebApp. However, results showed significant challenges due to background noise, inconsistent recording quality, environmental acoustics, and varied driving behaviour. These uncontrolled variables drastically reduced classification accuracy and exposed the limitations of the current system, which was trained primarily on clean and consistent data.

Finally, user-based evaluations through live feedback sessions and questionnaires demonstrated that the system is well-received by its target audience. Users praised its ease of use, speed, and responsiveness, with minor design complaints noted and addressed. Feedback showed strong satisfaction in both functionality and user interface, indicating the system’s readiness for wider usage and future expansion.

To sum up, while the system achieves high performance in controlled environments and through its WebApp, real-world usage highlighted critical areas for improvement. Nevertheless, user feedback confirmed the system's value and potential, positioning it as a practical and user-friendly application in the realm of car sound classification.

Chapter 6: Conclusion and Future Works

6.1 Achievements

The Final Year Project has successfully achieved and exceeded the three primary objective that were initially planned in Chapter 1. The key achievements are detailed as follows:

6.1.1 Comprehensive and Diverse Dataset Collection

A comprehensive and high-quality dataset was successfully curated, consisting of audio recordings from 233 distinct stock car models. This was achieved by leveraging the realistic and consistent sound engineering of the video game Forza Horizon 5, providing a cost-effective and scalable method for data acquisition. To enhance the dataset's robustness and prepare it for machine learning, strategic data augmentation techniques such as time stretching, pitch shifting, and gain adjustments were systematically applied.

6.1.2 Development of an Effective Audio Processing and Classification Pipeline

An end-to-end audio analysis pipeline was developed using Python and its extensive libraries. This pipeline incorporated sophisticated methods for audio preprocessing, including the removal of silent segments to isolate relevant sound data. A rich set of acoustic features was extracted, including Mel-Frequency Cepstral Coefficients (MFCCs), Zero Crossing Rate (ZCR), Spectral Centroid, Log-Mel Spectrogram, Chroma,

and Tonnetz features, to create a detailed numerical representation of each car's unique sound signature.

6.1.3 Successful Implementation of Machine Learning Models

Three distinct machine learning models—Support Vector Machine (SVM), Random Forest (RF), and a Convolutional Neural Network (CNN)—were successfully designed, trained, and evaluated. Through rigorous 5-fold cross-validation, the models demonstrated high classification accuracy on the controlled dataset, with the SVM achieving a mean accuracy of 95.72%, the CNN 95.28%, and the RF 93.02%. These results confirm the viability of using acoustic signatures for accurate vehicle identification in controlled settings.

6.1.4 Creation of a Functional and User-Friendly Web Application

The project culminated in the development of a fully functional web-based application. Built with the Django framework, the application provides an intuitive and responsive user interface that allows users to easily upload car audio files and receive classification results from all three trained models in near real-time. The system's usability was validated through a System Usability Scale (SUS) analysis, achieving an exceptionally high score of 93.70, indicating an "Excellent" level of user satisfaction and ease of use.

6.2 Limitations

Various limitations were discovered from using and operating the system. One of the limitations that was announced beforehand was the inability of the system to classify cars that had been modified specifically to augment the perceived car sounds. This includes cars with modified exhaust system, cars with engine swap, modified intake systems, modified ECU tuning, and the addition of turbochargers or superchargers. Furthermore, the system is completely unable to classify Electric Vehicle (EV) as all these cars do not produce any distinct and unique sounds due to the lack of an engine and exhaust system. Moreover, car audio obtained from internet sources have varying amount of background noise, speech, low-quality recordings, unknown variability and vast unpredictability are known to cause inaccuracy in the system.

6.3 Future Works

Future project efforts will prioritize addressing the third limitation: online car audio sources. The first two limitations, which involve inherent sound production issues, cannot be improved or fixed by an external system. Therefore, more effort will be dedicated to making the system more adaptable to uncontrolled, online, variable car sounds. The gathered feedback also indicates that users would want images to be shown in the results page as an inclusion for visual aids, indicating it as another priority.

References:

Ansari, Md. R., Tumpa, S. A., Raya, J. A. F., & Murshed, M. N. (2021). Comparison between Support Vector Machine and Random Forest for Audio Classification. *2021 International Conference on Electronics, Communications and Information Technology (ICECIT)*, 1–4. https://doi.org/10.1109/ICECIT54077.2021.9641152

Breiman, L. (2001). Random Forests. *Machine Learning*, *45*(1), 5–32. https://doi.org/10.1023/A:1010933404324

Bulatovic, N., & Djukanovic, S. (2022). Mel-spectrogram features for acoustic vehicle detection and speed estimation. *2022 26th International Conference on Information Technology (IT)*, 1–4. https://doi.org/10.1109/IT54280.2022.9743540

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297. https://doi.org/10.1007/BF00994018

George, J., Cyril, A., I. Koshy, B., & Mary, L. (2013). Exploring Sound Signature for Vehicle Detection and Classification Using ANN. *International Journal on Soft Computing*, *4*(2), 29–36. https://doi.org/10.5121/ijsc.2013.4203

George, J., Mary, L., & Riyas K S. (2013). Vehicle detection and classification from acoustic signal using ANN and KNN. *2013 International Conference on Control Communication and Computing (ICCC)*, 436–439. https://doi.org/10.1109/ICCC.2013.6731694

Grama, L., & Rusu, C. (2017). Audio signal classification using Linear Predictive Coding and Random Forests. *2017 International Conference on Speech Technology and*

Human-Computer Dialogue (SpeD), 1–9. https://doi.org/10.1109/SPED.2017.7990431

Hershey, S., Chaudhuri, S., Ellis, D. P. W., Gemmeke, J. F., Jansen, A., Moore, R. C., Plakal, M., Platt, D., Saurous, R. A., Seybold, B., Slaney, M., Weiss, R. J., & Wilson, K. (2017). CNN architectures for large-scale audio classification. *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 131–135. https://doi.org/10.1109/ICASSP.2017.7952132

Huadong Wu, Siegel, M., & Khosla, P. (1999). Vehicle sound signature recognition by frequency vector principal component analysis. *IEEE Transactions on Instrumentation and Measurement*, *48*(5), 1005–1009. https://doi.org/10.1109/19.799662

Huang, J. J., & Leanos, J. J. A. (2018). *AclNet: Efficient end-to-end audio classification CNN* (No. arXiv:1811.06669). arXiv. https://doi.org/10.48550/arXiv.1811.06669

Kozhisseri, S., & Bikdash, M. (2009). Spectral features for the classification of civilian vehicles using acoustic sensors. *2009 IEEE Workshop on Computational Intelligence in Vehicles and Vehicular Systems*, 93–100. https://doi.org/10.1109/CIVVS.2009.4938729

Kubera, E., Wieczorkowska, A., & Skrzypiec, K. (2015). Audio-Based Hierarchic Vehicle Classification for Intelligent Transportation Systems. In F. Esposito, O. Pivert, M.-S. Hacid, Z. W. Rás, & S. Ferilli (Eds.), *Foundations of Intelligent Systems* (Vol. 9384, pp. 343–352). Springer International Publishing. https://doi.org/10.1007/978-3-319-25252-0_37

Lu, L., Zhang, H.-J., & Li, S. Z. (2003). Content-based audio classification and segmentation by using support vector machines. *Multimedia Systems*, *8*(6), 482–492. https://doi.org/10.1007/s00530-002-0065-0

Palanisamy, K., Singhania, D., & Yao, A. (2020). *Rethinking CNN Models for Audio Classification* (No. arXiv:2007.11154). arXiv. https://doi.org/10.48550/arXiv.2007.11154

Rai, P., Golchha, V., Srivastava, A., Vyas, G., & Mishra, S. (2016). An automatic classification of bird species using audio feature extraction and support vector machines. *2016 International Conference on Inventive Computation Technologies (ICICT)*, 1–5. https://doi.org/10.1109/INVENTIVE.2016.7823241

Rong, F. (2016). Audio Classification Method Based on Machine Learning. *2016 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS)*, 81–84. https://doi.org/10.1109/ICITBS.2016.98

Sprengel, E., Jaggi, M., Kilcher, Y., & Hofmann, T. (2016). *Audio Based Bird Species Identification using Deep Learning Techniques*.

Wieczorkowska, A., Kubera, E., Słowik, T., & Skrzypiec, K. (2018). Spectral features for audio based vehicle and engine classification. *Journal of Intelligent Information Systems*, *50*(2), 265–290. https://doi.org/10.1007/s10844-017-0459-2

Wu, D. (2019). An Audio Classification Approach Based on Machine Learning. *2019 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS)*, 626–629. https://doi.org/10.1109/ICITBS.2019.00156

Yassin, A. I., Mohd Shariff, K. K., Kechik, M. A., Ali, A. M., & Megat Amin, M. S. (2023). Acoustic Vehicle Classification Using Mel-Frequency Features with Long Short-

Term Memory Neural Networks. *TEM Journal*, 1490–1496. https://doi.org/10.18421/TEM123-29

Zaman, K., Sah, M., Direkoglu, C., & Unoki, M. (2023). A Survey of Audio Classification Using Deep Learning. *IEEE Access*, *11*, 106620–106649. https://doi.org/10.1109/ACCESS.2023.3318015

Zhang, Y., & Lv, D. (2015). Selected Features for Classifying Environmental Audio Data with Random Forest. *The Open Automation and Control Systems Journal*, *7*(1), 135–142. https://doi.org/10.2174/1874444301507010135

Appendix A

System Usability Scale (SUS)

		Strongly disagree 1	2	3	4	Strongly agree 5
1.	I think I would like to use this web-based system more frequently					
2.	I think the web-based system is too complicated					
3.	I think the web-based system is simple to use					
4.	I would need help from a technical person to use this web-based system					
5.	I think the different function/features of this web-based system work well together					
6.	I think there is too much inconsistency in the web-based system					
7.	I imagine most people will learn to use the web-based system very quickly					
8.	I find the web-based system very tricky to use					
9.	I feel very comfortable using the web-based system					
10.	I have to learn a lot of new skills before I can start using the web-based system					

Open Ended Questions

1. What are your main opinions on the app.
2. Give your opinion on the design.
3. What else do you think is missing from the app.

Appendix B

Source Code of Car Recognition WebApp, Audio_Augmentation, Pre_Feature_Extraction, Car_Audio_ML

WebFront/manage.py

```
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE',
'CarRecognitionWebApp.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and
"
            "available on your PYTHONPATH environment variable? Did
you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()
```

requirements.txt

```
absl-py==2.2.1
asgiref==3.8.1
astunparse==1.6.3
```

```
audioread==3.0.1
certifi==2025.1.31
cffi==1.17.1
charset-normalizer==3.4.1
decorator==5.2.1
Django==5.2
flatbuffers==25.2.10
gast==0.6.0
google-pasta==0.2.0
grpcio==1.71.0
h5py==3.13.0
idna==3.10
joblib==1.4.2
keras==3.9.2
lazy_loader==0.4
libclang==18.1.1
librosa==0.11.0
llvmlite==0.44.0
Markdown==3.7
markdown-it-py==3.0.0
MarkupSafe==3.0.2
mdurl==0.1.2
ml_dtypes==0.5.1
msgpack==1.1.0
namex==0.0.8
numba==0.61.0
numpy==2.1.3
opt_einsum==3.4.0
optree==0.14.1
packaging==24.2
platformdirs==4.3.7
pooch==1.8.2
protobuf==5.29.4
pycparser==2.22
Pygments==2.19.1
requests==2.32.3
rich==14.0.0
scikit-learn==1.6.1
scipy==1.15.2
setuptools==78.1.0
```

```
six==1.17.0
soundfile==0.13.1
soxr==0.5.0.post1
sqlparse==0.5.3
tensorboard==2.19.0
tensorboard-data-server==0.7.2
tensorflow==2.19.0
termcolor==3.0.1
threadpoolctl==3.6.0
typing_extensions==4.13.0
tzdata==2025.2
urllib3==2.3.0
Werkzeug==3.1.3
wheel==0.45.1
wrapt==1.17.2
```

WebFront/CarRecognitionWebApp/settings.py

```
"""
Django settings for CarRecognitionWebApp project.

Generated by 'django-admin startproject' using Django 5.2.

For more information on this file, see
https://docs.djangoproject.com/en/5.2/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/5.2/ref/settings/
"""

from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
# See
https://docs.djangoproject.com/en/5.2/howto/deployment/checklist/
```

```
# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-
90_$pao*r2v!ixit#c_j7@w#x&(*7pnmvvi(%@^+2wy5v*^*4c'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'MainApp',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'CarRecognitionWebApp.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
```

```
            'context_processors': [
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages'
,
            ],
        },
    },
]

WSGI_APPLICATION = 'CarRecognitionWebApp.wsgi.application'


# Database
# https://docs.djangoproject.com/en/5.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}


# Password validation
# https://docs.djangoproject.com/en/5.2/ref/settings/#auth-password-
validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValid
ator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
```

```
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/5.2/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/5.2/howto/static-files/

STATIC_URL = 'static/'
STATICFILES_DIRS = [BASE_DIR / 'MainApp/templates/static']  # if you
use a global static folder

# Default primary key field type
# https://docs.djangoproject.com/en/5.2/ref/settings/#default-auto-
field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

CarRecognitionWebApp/urls.py

```
"""
URL configuration for CarRecognitionWebApp project.
```

```
The `urlpatterns` list routes URLs to views. For more information
please see:
    https://docs.djangoproject.com/en/5.2/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(),
name='home')
Including another URLconf
    1. Import the include() function: from django.urls import
include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('MainApp.urls')),
]
```

MainApp/urls.py

```
from django.urls import path
from .views import upload_audio
from .views import results_page

urlpatterns = [
    path('', upload_audio, name='upload_audio'),
    path('results/', results_page, name='results'),
]
```

MainApp/views.py

```
from django.shortcuts import render, redirect
from django.http import HttpResponse

import os
import numpy as np
import librosa
import pickle
from joblib import load
import tensorflow as tf
from django.shortcuts import render
from django.core.files.storage import default_storage
from django.core.files.base import ContentFile
from django.http import JsonResponse
from sklearn.preprocessing import StandardScaler

# Load Machine Learning Models
cnn_model = tf.keras.models.load_model(r"C:\Users\Isaiah
Ho\Desktop\Information System Year 4\TMF4913 Sem 1 2425 (G01) Final
Year Project 2\Preprocessing and Feature Extraction\ML Models\Run
16\cnn_model.keras")
with open(r"C:\Users\Isaiah Ho\Desktop\Information System Year
4\TMF4913 Sem 1 2425 (G01) Final Year Project 2\Preprocessing and
Feature Extraction\ML Models\Run 16\rf_model.pkl", "rb") as f:
    rf_model = load(f)
with open(r"C:\Users\Isaiah Ho\Desktop\Information System Year
4\TMF4913 Sem 1 2425 (G01) Final Year Project 2\Preprocessing and
Feature Extraction\ML Models\Run 16\svm_model.pkl", "rb") as f:
    svm_model = load(f)

# Load the LabelEncoder
label_encoder = load(r"C:\Users\Isaiah Ho\Desktop\Information System
Year 4\TMF4913 Sem 1 2425 (G01) Final Year Project 2\Preprocessing
and Feature Extraction\ML Models\Run 16\label_encoder.pkl")


def remove_silence(y, sr, top_db=30):
    """Removes silent parts from the audio signal."""
    non_silent_intervals = librosa.effects.split(y, top_db=top_db)
    y_trimmed = np.concatenate([y[start:end] for start, end in
non_silent_intervals])
    return y_trimmed
```

```
def extract_features(audio_path):
    """Extracts audio features including MFCCs, ZCR, spectral
centroid, log-mel spectrogram, chroma, and tonnetz."""
    # Load audio file
    y, sr = librosa.load(audio_path, sr=22050, mono=True)

    # Remove silence
    y = remove_silence(y, sr)

    # Compute MFCCs
    mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)
    mfccs_mean = np.mean(mfccs, axis=1)

    # Compute Zero Crossing Rate (ZCR)
    zcr = librosa.feature.zero_crossing_rate(y)
    zcr_mean = np.mean(zcr)

    # Compute Spectral Centroid
    spectral_centroid = librosa.feature.spectral_centroid(y=y, sr=sr)
    spectral_centroid_mean = np.mean(spectral_centroid)

    # Compute Log-Mel Spectrogram
    mel_spectrogram = librosa.feature.melspectrogram(y=y, sr=sr)
    log_mel_spectrogram = librosa.power_to_db(mel_spectrogram,
ref=np.max)
    log_mel_spectrogram_mean = np.mean(log_mel_spectrogram, axis=1)

    # Compute Chroma Features
    chroma = librosa.feature.chroma_stft(y=y, sr=sr)
    chroma_mean = np.mean(chroma, axis=1)

    # Compute Tonnetz Features
    tonnetz = librosa.feature.tonnetz(y=librosa.effects.harmonic(y),
sr=sr)
    tonnetz_mean = np.mean(tonnetz, axis=1)

    # Concatenate all features into a single feature vector
```

```
    feature_vector = np.hstack([mfccs_mean, [zcr_mean,
spectral_centroid_mean], log_mel_spectrogram_mean, chroma_mean,
tonnetz_mean])

    # Normalize features using the pre-fitted scaler
    scaler = load(r"C:\Users\Isaiah Ho\Desktop\Information System
Year 4\TMF4913 Sem 1 2425 (G01) Final Year Project 2\Preprocessing
and Feature Extraction\scaler.pkl")
    feature_vector = scaler.transform(feature_vector.reshape(1, -
1)).flatten()

    # Reshape to match the CNN model's input shape (1, 161, 1)
    feature_vector = feature_vector.reshape(1, 161, 1)

    # Log the feature vector for debugging
    print(f"Extracted feature vector: {feature_vector}")
    print(f"Feature vector shape: {feature_vector.shape}")

    return feature_vector.reshape(1, -1)


def predict_car(audio_path):
    # Extract features
    features = extract_features(audio_path)
    print(f"Features shape for CNN: {features.shape}")  # Debug:
Check feature shape for CNN

    # CNN Prediction
    cnn_pred_index = int(np.argmax(cnn_model.predict(features)))
    cnn_pred_label =
label_encoder.inverse_transform([cnn_pred_index])[0]
    print(f"CNN Prediction: {cnn_pred_label}")  # Debug: Check CNN
prediction

    # Reshape features for RF and SVM models
    features_flat = features.reshape(1, -1)  # Shape (1, 161)

    # Random Forest Prediction
    rf_pred_index = int(rf_model.predict(features_flat)[0])
    rf_pred_label =
label_encoder.inverse_transform([rf_pred_index])[0]
```

```
    print(f"Random Forest Prediction: {rf_pred_label}")  # Debug:
Check RF prediction

    # SVM Prediction
    svm_pred_index = int(svm_model.predict(features_flat)[0])
    svm_pred_label =
label_encoder.inverse_transform([svm_pred_index])[0]
    print(f"SVM Prediction: {svm_pred_label}")  # Debug: Check SVM
prediction

    # Return predictions
    predictions = {
        "CNN_Model": cnn_pred_label,
        "Random_Forest_Model": rf_pred_label,
        "SVM_Model": svm_pred_label
    }
    return predictions

def upload_audio(request):
    if request.method == "POST" and request.FILES.get("audio_file"):
        audio_file = request.FILES["audio_file"]
        file_path = default_storage.save("uploads/" +
audio_file.name, ContentFile(audio_file.read()))
        full_path = os.path.join(default_storage.location, file_path)
        results = predict_car(full_path)

        # Store results and file name in the session
        request.session['results'] = results
        request.session['file_name'] = audio_file.name  # Store file
name
        return redirect('results')  # Redirect to the results page
    return render(request, "MLModels.html")

def results_page(request):
    results = request.session.get('results', {})
    file_name = request.session.get('file_name', 'Unknown File')  #
Default to 'Unknown File' if not set
    return render(request, "results.html", {"results": results,
"file_name": file_name})
```

MainApp/templates/MLModels.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Car Recognition WebApp</title>
    <style>
        body {
            margin: 0;
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-
serif;
            background: linear-gradient(135deg, #e6f3f9 0%, #f8fbff
100%);
            text-align: center;
            min-height: 100vh;
        }

        header {
            background: linear-gradient(90deg, #4a9cfc 60%, #13a309
100%);
            color: white;
            padding: 28px 0 22px 0;
            display: flex;
            align-items: center;
            justify-content: center;
            box-shadow: 0 4px 18px rgba(0,0,0,0.08);
            position: relative;
        }

        .header-content {
            display: flex;
            align-items: center;
            gap: 28px;
        }

        header img {
```

```
            width: 70px;
            height: 70px;
            object-fit: contain;
            border-radius: 14px;
            box-shadow: 0 2px 8px rgba(0,0,0,0.10);
            background: #fff;
            padding: 6px;
        }

        .header-title {
            text-decoration: none;
            color: inherit;
            display: flex;
            align-items: center;
        }

        header h1 {
            margin: 0;
            font-size: 2.5rem;
            font-family: 'Segoe UI Semibold', Helvetica, Arial, sans-
serif;
            letter-spacing: 1px;
            text-shadow: 0 2px 8px rgba(0,0,0,0.07);
        }

        .menu-icon {
            font-size: 2.2rem;
            margin-left: 24px;
            cursor: pointer;
            user-select: none;
            transition: color 0.2s;
        }
        .menu-icon:hover {
            color: #e6f9e6;
        }

        .upload-section {
            margin-top: 80px;
            padding-bottom: 0px;
            display: flex;
```

```
            justify-content: center;
            align-items: center;
        }

        /* Skeuomorphic Drop Area */
        .drop-area {
            display: flex;
            flex-direction: column;
            align-items: center;
            justify-content: center;
            border-radius: 28px;
            background: linear-gradient(145deg, #f8fbff 60%, #e6f3f9
100%);
            box-shadow:
                8px 8px 24px #d1e0e6,
                -8px -8px 24px #ffffff,
                inset 2px 2px 8px #e3eaf3,
                inset -2px -2px 8px #ffffff;
            padding: 54px 38px;
            cursor: pointer;
            transition: box-shadow 0.3s, background 0.3s;
            width: 480px;
            min-width: unset;
            min-height: unset;
            border: 1.5px solid #e3eaf3;
        }

        .drop-area.dragover {
            box-shadow:
                0 0 0 4px #b6f7c2,
                8px 8px 24px #d1e0e6,
                -8px -8px 24px #ffffff,
                inset 2px 2px 8px #e3eaf3,
                inset -2px -2px 8px #ffffff;
            background: linear-gradient(145deg, #e6f9e6 60%, #f8fbff
100%);
        }

        .upload-icon img {
            width: 120px;
```

```
    height: 120px;
    margin-bottom: 21px;
    border-radius: 24px;
    box-shadow:
        4px 4px 16px #d1e0e6,
        -4px -4px 16px #ffffff,
        inset 1px 1px 4px #e3eaf3,
        inset -1px -1px 4px #ffffff;
    background: #f8fbff;
    padding: 10px;
}

.upload-label {
    font-size: 22px;
    font-weight: 600;
    color: #4a9cfc;
    margin-bottom: 10px;
    text-align: center;
    text-shadow: 0 1px 0 #fff, 0 2px 4px #e3eaf3;
}

input[type="file"] {
    display: none;
}

.file-name {
    margin-top: 18px;
    font-size: 18px;
    color: #297374;
    word-break: break-all;
    background: #f8fbff;
    border-radius: 10px;
    box-shadow: 2px 2px 8px #e3eaf3, -2px -2px 8px #ffffff;
    padding: 8px 18px;
    display: inline-block;
}

@media (max-width: 600px) {
    header {
        flex-direction: column;
```

```
        padding: 18px 0 12px 0;
    }
    .header-content {
        flex-direction: column;
        gap: 10px;
    }
    header img {
        width: 48px;
        height: 48px;
    }
    header h1 {
        font-size: 1.4rem;
    }
    .menu-icon {
        font-size: 1.4rem;
        margin-left: 0;
    }
    .drop-area {
        width: 95vw;
        padding: 27px 6vw;
    }
    .upload-icon img {
        width: 90px;
        height: 90px;
    }
    .upload-label {
        font-size: 16px;
    }
}

.wave {
    display: block;
    width: 100%;
    margin-top: 50px;
}

.footer {
    margin-top: 0px;
    font-size: 16px;
    color: #333;
```

```
            background: #e3eaf3;
            padding: 10px;
            border-top: 0px solid #ddd;
            box-shadow: 0 -2px 8px #e3eaf3;
        }
    </style>
</head>
<body>
    <header>
        <div class="header-content">
            <a href="{% url 'upload_audio' %}">
                <img src="{% static 'images/Car.png' %}" alt="Car
Icon">
            </a>
            <a href="{% url 'upload_audio' %}" class="header-title">
                <h1>Car Recognition WebApp</h1>
            </a>
        </div>
        <div class="menu-icon">&#9776;</div>
    </header>

    <div class="upload-section">
        <form id="upload-form" action="" method="post"
enctype="multipart/form-data">
            {% csrf_token %}
            <label for="audio-upload" class="drop-area" id="drop-
area">
                <div class="upload-icon">
                    <img src="{% static 'images/Upload Audio
2.png' %}" alt="Upload Icon">
                </div>
                <span class="upload-label">Drag & Drop or Click to
Upload<br><span style="font-size:20px;">(FLAC only)</span></span>
                <input id="audio-upload" type="file"
name="audio_file" accept=".flac" required>
                <div id="file-name" class="file-name"
style="display:none;"></div>
            </label>
        </form>
    </div>
```

```
    <script>
        const dropArea = document.getElementById('drop-area');
        const fileInput = document.getElementById('audio-upload');
        const fileNameDisplay = document.getElementById('file-name');
        const form = document.getElementById('upload-form');

        // Drag & drop events
        dropArea.addEventListener('dragover', (e) => {
            e.preventDefault();
            dropArea.classList.add('dragover');
        });
        dropArea.addEventListener('dragleave', () => {
            dropArea.classList.remove('dragover');
        });
        dropArea.addEventListener('drop', (e) => {
            e.preventDefault();
            dropArea.classList.remove('dragover');
            const files = e.dataTransfer.files;
            if (files.length > 0 && files[0].name.endsWith('.flac'))
{
                fileInput.files = files;
                showFileName(files[0].name);
                form.submit();
            } else {
                alert('Please upload a .flac file.');
            }
        });

        // Show file name and auto-submit on file select
        fileInput.addEventListener('change', function() {
            if (this.files.length > 0) {
                if (this.files[0].name.endsWith('.flac')) {
                    showFileName(this.files[0].name);
                    form.submit();
                } else {
                    alert('Please upload a .flac file.');
                    this.value = '';
                    hideFileName();
                }
```

```
            } else {
                hideFileName();
            }
        });

        function showFileName(name) {
            fileNameDisplay.textContent = "Selected: " + name;
            fileNameDisplay.style.display = "inline-block";
        }
        function hideFileName() {
            fileNameDisplay.textContent = "";
            fileNameDisplay.style.display = "none";
        }
        // Hide on page load
        hideFileName();
    </script>

    <img class="wave" src="{% static 'images/waves.svg' %}"
alt="Wave">

    <footer class="footer">
        <span>&copy; 2025 Car Recognition WebApp &mdash; Isaiah Ho
Chi Ann</span>
    </footer>
</body>
</html>
```

MainApp/templates/results.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Prediction Results</title>
    <style>
        body {
            margin: 0;
```

```
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-
serif;
            background: linear-gradient(135deg, #f4fcff 0%, #e6f1ff
100%);
            text-align: center;
            min-height: 100vh;
            position: relative;
            overflow-x: hidden;
        }

        .bg-blob {
            position: fixed;
            top: -100px;
            left: -100px;
            width: 320px;
            height: 320px;
            background: #4a9cfc;
            opacity: 0.10;
            border-radius: 50%;
            filter: blur(60px);
            z-index: 0;
            pointer-events: none;
        }

        header {
            background: linear-gradient(90deg, #4a9cfc 60%, #13a309
100%);
            color: white;
            padding: 28px 0 22px 0;
            display: flex;
            align-items: center;
            justify-content: center;
            box-shadow: 0 4px 18px rgba(0,0,0,0.08);
            position: relative;
            z-index: 1;
        }

        .header-content {
            display: flex;
            align-items: center;
```

```
            gap: 28px;
        }

        header img {
            width: 70px;
            height: 70px;
            object-fit: contain;
            border-radius: 14px;
            box-shadow: 0 2px 8px rgba(0,0,0,0.10);
            background: #fff;
            padding: 6px;
        }

        .header-title {
            text-decoration: none;
            color: inherit;
            display: flex;
            align-items: center;
        }

        header h1 {
            margin: 0;
            font-size: 2.5rem;
            font-family: 'Segoe UI Semibold', Helvetica, Arial, sans-
serif;
            letter-spacing: 1px;
            text-shadow: 0 2px 8px rgba(0,0,0,0.07);
        }

        .menu-icon {
            font-size: 2.2rem;
            margin-left: 24px;
            cursor: pointer;
            user-select: none;
            transition: color 0.2s;
        }
        .menu-icon:hover {
            color: #e6f9e6;
        }
```

```
        /* Skeuomorphic Results Card */
        .results-card {
            margin: 80px auto 0 auto;
            background: linear-gradient(145deg, #f8fbff 60%, #e6f3f9
100%);
            border-radius: 28px;
            box-shadow:
                8px 8px 24px #d1e0e6,
                -8px -8px 24px #ffffff,
                inset 2px 2px 8px #e3eaf3,
                inset -2px -2px 8px #ffffff;
            padding: 54px 38px 38px 38px;
            max-width: 480px;
            text-align: left;
            display: flex;
            flex-direction: column;
            align-items: center;
            position: relative;
            z-index: 1;
        }

        .results-card h2 {
            font-size: 32px;
            margin-bottom: 24px;
            color: #4a9cfc;
            text-align: center;
            text-shadow: 0 1px 0 #fff, 0 2px 4px #e3eaf3;
        }

        .file-info {
            display: flex;
            align-items: center;
            margin-bottom: 24px;
            font-size: 1.1rem;
            color: #297374;
            background: #f8fbff;
            border-radius: 14px;
            box-shadow: 2px 2px 8px #e3eaf3, -2px -2px 8px #ffffff;
            padding: 12px 22px;
            width: 100%;
```

```
    box-sizing: border-box;
}
.file-icon {
    font-size: 2rem;
    margin-right: 12px;
}

.prediction-list {
    width: 100%;
}
.prediction {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin: 18px 0;
    font-size: 1.2rem;
    padding: 14px 0;
    border-bottom: 1.5px solid #e6f3f9;
}
.prediction:last-child {
    border-bottom: none;
}
.model-label {
    color: #888;
    font-weight: 500;
}
.model-result {
    font-weight: 600;
    color: #222;
    background: #f8fbff;
    border-radius: 10px;
    box-shadow: 2px 2px 8px #e3eaf3, -2px -2px 8px #ffffff;
    padding: 8px 18px;
    display: inline-block;
}
.model-result.highlight {
    color: #0b4804;
    background: #e6f9e6;
}
.model-result.highlight2 {
```

```
            color: #073601;
            background: #e6f9e6;
        }
        .model-result.highlight3 {
            color: #0e6d03;
            background: #e6f9e6;
        }

        .back-button-container {
            margin-top: 40px;
            width: 100%;
            display: flex;
            justify-content: center;
        }
        .back-button {
            background: linear-gradient(90deg, #4a9cfc 60%, #13a309
100%);
            color: white;
            padding: 14px 32px;
            font-size: 18px;
            border: none;
            border-radius: 12px;
            cursor: pointer;
            text-decoration: none;
            transition: background 0.3s;
            display: inline-flex;
            align-items: center;
            font-weight: 600;
            box-shadow: 0 2px 8px #e3eaf3, 0 2px 8px #ffffff;
        }
        .back-button:hover {
            background: linear-gradient(90deg, #13a309 60%, #4a9cfc
100%);
        }
        .back-button .icon {
            margin-right: 10px;
            font-size: 1.2em;
        }

        .wave {
```

```
    display: block;
    width: 100%;
    margin-top: 10px;
    z-index: 1;
    position: relative;
}

.footer {
    margin-top: 0px;
    font-size: 16px;
    color: #333;
    background: #e3eaf3;
    padding: 10px;
    border-top: 0px solid #ddd;
    box-shadow: 0 -2px 8px #e3eaf3;
}

@media (max-width: 600px) {
    header {
        flex-direction: column;
        padding: 18px 0 12px 0;
    }
    .header-content {
        flex-direction: column;
        gap: 10px;
    }
    header img {
        width: 48px;
        height: 48px;
    }
    header h1 {
        font-size: 1.4rem;
    }
    .menu-icon {
        font-size: 1.4rem;
        margin-left: 0;
    }
    .results-card {
        max-width: 98vw;
        padding: 18px 4vw 18px 4vw;
```

```
            }
            .file-info {
                font-size: 1rem;
                padding: 8px 6vw;
            }
            .prediction {
                font-size: 1rem;
            }
            .back-button {
                font-size: 16px;
                padding: 10px 18px;
            }
        }
    </style>
</head>
<body>
    <div class="bg-blob"></div>
    <header>
        <div class="header-content">
            <a href="{% url 'upload_audio' %}">
                <img src="{% static 'images/Car.png' %}" alt="Car
Icon">
            </a>
            <a href="{% url 'upload_audio' %}" class="header-title">
                <h1>Car Recognition WebApp</h1>
            </a>
        </div>
        <div class="menu-icon">&#9776;</div>
    </header>

    <div class="results-card">
        <h2>Prediction Results</h2>
        <div class="file-info">
            <span class="file-icon">🎵</span>
            <span class="file-name">{{ file_name }}</span>
        </div>
        <div class="prediction-list">
            <div class="prediction">
                <span class="model-label">CNN Model:</span>
```

```
                <span class="model-result
highlight2">{{ results.CNN_Model }}</span>
            </div>
            <div class="prediction">
                <span class="model-label">Random Forest:</span>
                <span class="model-result
highlight">{{ results.Random_Forest_Model }}</span>
            </div>
            <div class="prediction">
                <span class="model-label">SVM:</span>
                <span class="model-result
highlight3">{{ results.SVM_Model }}</span>
            </div>
        </div>
        <div class="back-button-container">
            <a href="{% url 'upload_audio' %}" class="back-button">
                Upload Another
            </a>
        </div>
    </div>

    <img class="wave" src="{% static 'images/wave2.svg' %}"
alt="Wave">

    <footer class="footer">
        <span>&copy; 2025 Car Recognition WebApp &mdash; Isaiah Ho
Chi Ann</span>
    </footer>
</body>
</html>
```

Audio_Augmentation.py

```
import os
import librosa
import soundfile as sf
from audiomentations import Compose, AddGaussianNoise, TimeStretch,
PitchShift, Gain
from pathlib import Path

# === CONFIGURATION ===
input_dir = r"C:\Users\Isaiah Ho\Desktop\Information System Year
4\TMF4913 Sem 1 2425 (G01) Final Year Project 2\Car Audio Recordings
3"          # Folder with original FLAC files
output_base_dir = r"C:\Users\Isaiah Ho\Desktop\Information System
Year 4\TMF4913 Sem 1 2425 (G01) Final Year Project 2\Car Audio
Recordings Unseen 4"    # Folder to store augmented audio
num_versions = 1             # How many augmented files per original

# Ensure output directory exists
os.makedirs(output_base_dir, exist_ok=True)

# Define augmentation pipeline
augment = Compose([
    TimeStretch(min_rate=0.9, max_rate=1.10, p=0.6),
    PitchShift(min_semitones=-2, max_semitones=2, p=0.7),
    Gain(min_gain_db=-6, max_gain_db=6, p=0.6)
])

# Process each file
for file in os.listdir(input_dir):
    if file.endswith(".flac"):
        file_path = os.path.join(input_dir, file)
        y, sr = librosa.load(file_path, sr=None)

        for version in range(1, num_versions + 1):
            # Apply augmentation
            y_aug = augment(samples=y, sample_rate=sr)

            # Create versioned output directory
            version_dir = os.path.join(output_base_dir,
f"version_{version}")
```

```
            os.makedirs(version_dir, exist_ok=True)

            # Save the augmented file
            output_filename =
f"{Path(file).stem}_aug_v{version}.flac"
            output_path = os.path.join(version_dir, output_filename)
            sf.write(output_path, y_aug, sr)

            print(f"Saved: {output_path}")

print("All augmentations complete.")
```

Pre_Feature_Extraction.py

```
import librosa
import librosa.display
import numpy as np
import matplotlib.pyplot as plt
import os
import pandas as pd
import concurrent.futures
from tqdm import tqdm
from sklearn.preprocessing import StandardScaler
from joblib import dump

def remove_silence(y, sr, top_db=30):
    """Removes silent parts from the audio signal."""
    non_silent_intervals = librosa.effects.split(y, top_db=top_db)
    y_trimmed = np.concatenate([y[start:end] for start, end in
non_silent_intervals])
    return y_trimmed

def extract_features(audio_path, sr=22050, n_mfcc=13):
    """Extracts audio features including MFCCs, ZCR, spectral
centroid, log-mel spectrogram, chroma, and tonnetz."""

    # Load audio file
    y, sr = librosa.load(audio_path, sr=sr, mono=True)

    # Remove silence
    y = remove_silence(y, sr)

    # Compute MFCCs
    mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=n_mfcc)
    mfccs_mean = np.mean(mfccs, axis=1)

    # Compute Zero Crossing Rate (ZCR)
    zcr = librosa.feature.zero_crossing_rate(y)
    zcr_mean = np.mean(zcr)

    # Compute Spectral Centroid
    spectral_centroid = librosa.feature.spectral_centroid(y=y, sr=sr)
    spectral_centroid_mean = np.mean(spectral_centroid)
```

```
    # Compute Log-Mel Spectrogram
    mel_spectrogram = librosa.feature.melspectrogram(y=y, sr=sr)
    log_mel_spectrogram = librosa.power_to_db(mel_spectrogram,
ref=np.max)
    log_mel_spectrogram_mean = np.mean(log_mel_spectrogram, axis=1)

    # Compute Chroma Features
    chroma = librosa.feature.chroma_stft(y=y, sr=sr)
    chroma_mean = np.mean(chroma, axis=1)

    # Compute Tonnetz Features
    tonnetz = librosa.feature.tonnetz(y=librosa.effects.harmonic(y),
sr=sr)
    tonnetz_mean = np.mean(tonnetz, axis=1)

    # Concatenate all features into a single feature vector
    feature_vector = np.hstack([mfccs_mean, [zcr_mean,
spectral_centroid_mean], log_mel_spectrogram_mean, chroma_mean,
tonnetz_mean])

    return feature_vector

def process_audio_file(file_path):
    """Processes a single audio file and extracts features."""
    features = extract_features(file_path)
    return [os.path.basename(file_path)] + features.tolist()

def process_audio_files(directory):
    """Processes all audio files in the given directory and extracts
features concurrently."""
    files = [os.path.join(directory, file) for file in
os.listdir(directory) if file.endswith(".flac")]
    feature_list = []

    with concurrent.futures.ThreadPoolExecutor() as executor:
        results = list(tqdm(executor.map(process_audio_file, files),
total=len(files), desc="Processing Audio Files"))
        feature_list.extend(results)
```

```
    # Convert extracted features into a DataFrame
    columns = ["filename"] + [f"mfcc_{i}" for i in range(13)] +
["zcr", "spectral_centroid"] + [f"log_mel_{i}" for i in range(128)] +
[f"chroma_{i}" for i in range(12)] + [f"tonnetz_{i}" for i in
range(6)]
    df = pd.DataFrame(feature_list, columns=columns)

    # Normalize features for scikit-learn compatibility
    scaler = StandardScaler()
    df.iloc[:, 1:] = scaler.fit_transform(df.iloc[:, 1:])
    dump(scaler, "scaler.pkl")  # Save the scaler to a file
    return df

# Define the directory containing audio files
directory = r"C:\Users\Isaiah Ho\Desktop\Information System Year
4\TMF4913 Sem 1 2425 (G01) Final Year Project 2\Car Audio Recordings
7"  # Change this to the correct directory path

# Process audio files and extract features
audio_features_df = process_audio_files(directory)

# Save extracted features to CSV and NumPy file
audio_features_df.to_csv("audio_features.csv", index=False)
np.save("audio_features.npy", audio_features_df.iloc[:,
1:].to_numpy())

print("Feature extraction complete. Saved to audio_features.csv and
audio_features.npy.")
```

Car_Audio_ML.py

```
import pandas as pd
import numpy as np
import os
import joblib
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tqdm import tqdm
from sklearn.model_selection import GridSearchCV
import keras_tuner as kt
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
import time

# Load audio features from CSV file
csv_path = os.path.abspath(r"C:\Users\Isaiah Ho\Desktop\Information
System Year 4\TMF4913 Sem 1 2425 (G01) Final Year Project
2\Preprocessing and Feature Extraction\audio_features.csv")  # Ensure
it's absolute path
data = pd.read_csv(csv_path)

# Separate features and labels
X = data.iloc[:, 1:]  # Drop 'filename'
y = data['filename'].apply(lambda x: os.path.splitext(x)[0])  # Use
full car name without extension

# Encode labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# Save the LabelEncoder for later use
joblib.dump(label_encoder, "label_encoder.pkl")
```

```
# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded,
test_size=0.2, random_state=42, stratify=y_encoded)

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train SVM Model with GridSearchCV
print("\nTraining SVM Model with GridSearchCV...")
start_time = time.time()
param_grid = {
    'C': [0.1, 1, 10, 20],
    'gamma': ['scale', 'auto', 0.001, 0.01, 0.1, 1],
    'kernel': ['rbf', 'linear', 'poly']
}

grid_search = GridSearchCV(SVC(), param_grid, cv=5,
scoring='accuracy', verbose=2)
grid_search.fit(X_train, y_train)
svm_time = time.time() - start_time

print("Best SVM Parameters:", grid_search.best_params_)
svm_model = grid_search.best_estimator_
y_pred_svm = svm_model.predict(X_test)
print("SVM Accuracy:", accuracy_score(y_test, y_pred_svm))
print(f"SVM Training Time: {svm_time:.2f} seconds")

# Train Random Forest Model with 5-fold Cross Validation
print("\nTraining Random Forest Model with 5-Fold Cross
Validation...")
start_time = time.time()
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_cv_scores = cross_val_score(rf_model, X_train, y_train, cv=5,
scoring='accuracy', verbose=2)
rf_model.fit(X_train, y_train)
rf_time = time.time() - start_time

print("Random Forest 5-Fold CV Accuracies:", rf_cv_scores)
```

```
print("Random Forest Mean CV Accuracy:", np.mean(rf_cv_scores))
y_pred_rf = rf_model.predict(X_test)
print("Random Forest Test Accuracy:", accuracy_score(y_test,
y_pred_rf))
print(f"Random Forest Training Time: {rf_time:.2f} seconds")

# Define CNN Model
cnn_model = keras.Sequential([
    layers.Input(shape=(X_train.shape[1], 1)),
    layers.Conv1D(32, kernel_size=3, activation='relu'),
    layers.MaxPooling1D(pool_size=2),
    layers.Conv1D(64, kernel_size=3, activation='relu'),
    layers.MaxPooling1D(pool_size=2),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(len(np.unique(y_encoded)), activation='softmax')
])

# Compile CNN Model
cnn_model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Reshape input for CNN
X_train_cnn = X_train[..., np.newaxis]
X_test_cnn = X_test[..., np.newaxis]

# Train CNN Model with Progress Bar
print("\nTraining CNN Model...")
start_time = time.time()
cnn_model.fit(X_train_cnn, y_train, epochs=25, batch_size=32,
validation_data=(X_test_cnn, y_test), verbose=1)
cnn_time = time.time() - start_time
print(f"CNN Training Time: {cnn_time:.2f} seconds")

# Hyperparameter Tuning for CNN Model
def build_model(hp):
    model = keras.Sequential()
    model.add(layers.Input(shape=(X_train.shape[1], 1)))
```

```
    model.add(layers.Conv1D(filters=hp.Choice('filters', [32, 64,
128]), kernel_size=hp.Choice('kernel_size', [3, 5, 7]),
activation='relu'))
    model.add(layers.MaxPooling1D(pool_size=2))
    model.add(layers.Conv1D(filters=hp.Choice('filters_2', [64, 128,
256]), kernel_size=hp.Choice('kernel_size_2', [3, 5]),
activation='relu'))
    model.add(layers.MaxPooling1D(pool_size=2))
    model.add(layers.Flatten())
    model.add(layers.Dense(units=hp.Choice('dense_units', [64, 128,
256]), activation='relu'))
    model.add(layers.Dropout(rate=hp.Choice('dropout', [0.2, 0.3,
0.5])))
    model.add(layers.Dense(len(np.unique(y_encoded)),
activation='softmax'))
    model.compile(optimizer=keras.optimizers.Adam(learning_rate=hp.Ch
oice('learning_rate', [1e-2, 1e-3, 1e-4])),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model

tuner = kt.RandomSearch(
    build_model,
    objective='val_accuracy',
    max_trials=10,
    executions_per_trial=1,
    directory='my_dir',
    project_name='cnn_tuning'
)

tuner.search(X_train_cnn, y_train, epochs=20,
validation_data=(X_test_cnn, y_test), verbose=1)
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
print("Best CNN Parameters:", best_hps.values)

# Prepare data for CNN
X_cnn = X.values[..., np.newaxis]
y_cnn = y_encoded

# 5-Fold Cross Validation for CNN
```

```
print("\nTraining CNN Model with 5-Fold Cross Validation...")
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
cnn_cv_accuracies = []

def create_cnn_model(input_shape, num_classes):
    model = keras.Sequential([
        layers.Input(shape=input_shape),
        layers.Conv1D(32, kernel_size=3, activation='relu'),
        layers.MaxPooling1D(pool_size=2),
        layers.Conv1D(64, kernel_size=3, activation='relu'),
        layers.MaxPooling1D(pool_size=2),
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(num_classes, activation='softmax')
    ])
    model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model

for fold, (train_idx, val_idx) in enumerate(skf.split(X_cnn, y_cnn),
1):
    print(f"Fold {fold}...")
    model = create_cnn_model((X_cnn.shape[1], 1),
len(np.unique(y_cnn)))
    history = model.fit(
        X_cnn[train_idx], y_cnn[train_idx],
        epochs=25,
        batch_size=32,
        validation_data=(X_cnn[val_idx], y_cnn[val_idx]),
        verbose=0
    )
    val_acc = history.history['val_accuracy'][-1]
    print(f"Validation Accuracy for fold {fold}: {val_acc:.4f}")
    cnn_cv_accuracies.append(val_acc)

print("CNN 5-Fold CV Accuracies:", cnn_cv_accuracies)
print("CNN Mean CV Accuracy:", np.mean(cnn_cv_accuracies))

# Save Models
joblib.dump(svm_model, "svm_model.pkl")
```

```
joblib.dump(rf_model, "rf_model.pkl")

# Save CNN Model in Keras Format
cnn_model.save("cnn_model.keras")

print("All models trained and saved successfully.")
```

Appendix C

Timestamp	Email address	I think I would lik	I think the web-base	I think the web-b	I would need help t	I think the diff	I think there is t	I imagine most pe	I find the web-base	I feel very comfo	I have to learn a	What are your ma	Give your opinion	What else do you thin
6/15/2025 20:57:08	78016@siswa.unimas.r	5	1	5	1	5	1	5	1	5	1	The app allows me	It's clean, I give it th	**I need to see the pictu**
6/15/2025 21:13:32	80357@siswa.unimas.r	5	1	4	1	4	1	4	1	5	1	I like that I can kno	Its' clean	**Need the pictures to s**
6/15/2025 21:28:33	79790@siswa.unimas.r	5	2	5	1	5	1	4	1	4	1	None	quite good	**picture for the car**
6/16/2025 10:24:12	sebastianmaximusbatel	5	1	5	1	4	1	5	1	5	1	Very good	Attentive	**Model of cars**
6/16/2025 10:27:28	79242@siswa.unimas.r	4	1	4	1	4	1	4	1	4	1	Not sure if i need it	Design no issue, jus	**Unsure but seems use**
6/16/2025 10:28:24	79155@siswa.unimas.r	5	1	5	1	5	1	5	2	4	2	I think the app is si	The design is minim	**I want image to appea**
6/16/2025 18:06:38	79084@siswa.unimas.r	5	2	5	2	5	2	5	2	5	1	No opinions	Look good	**No**
6/16/2025 18:09:50	79210@siswa.unimas.r	4	2	4	2	4	2	4	2	4	2	Pretty good, have s	Could be more color	**More samples**
6/16/2025 18:20:35	ericnaweam1503@gm	5	1	5	1	5	1	5	1	5	1	The app is very use	The design of the ap	**The app is working pr**
6/16/2025 19:08:32	79354@siswa.unimas.r	5	1	5	1	5	1	5	1	5	1	Amazing app	Amazing design	**Nothing**
6/16/2025 20:32:43	78363@siswa.unimas.r	5	1	5	1	5	1	5	1	5	1	I like the idea of th	Nice and simple	**Nothing**
6/16/2025 20:34:39	81688@siswa.unimas.r	4	2	5	2	3	3	3	1	4	1	It functions well	Modern and simple	**Need more pictures**
6/16/2025 20:36:11	79218@siswa.unimas.r	5	1	5	1	5	2	5	1	5	1	No opinions	Very easy to unders	**Nothing**
6/16/2025 20:37:06	79256@siswa.unimas.r	5	1	5	1	5	1	5	1	5	1	Very nice	Beautiful	**Need car pictures**
6/16/2025 20:38:44	79707@siswa.unimas.r	5	2	5	1	5	1	5	1	5	1	App is useful	Very direct	**nothing at all**
6/16/2025 20:39:52	80034@siswa.unimas.r	5	1	4	1	5	1	5	1	5	1	I like the app	Very user friendly	**Nothing**
6/16/2025 20:47:44	78977@siswa.unimas.r	5	1	5	1	5	1	5	1	5	1	No opinions	Simple	**Nothing**
6/16/2025 20:48:40	79642@siswa.unimas.r	5	1	5	1	3	2	5	1	5	1	App is direct	The app looks nice	**Need pictures**
6/16/2025 20:50:50	77868@siswa.unimas.r	5	1	5	1	5	1	5	1	5	1	The app is really ni	Really well designed	**Prefer to have picture**
6/16/2025 20:52:12	81435@siswa.unimas.r	3	2	4	2	3	1	3	1	4	1	Incredibly easy to u	Very easy to unders	**Not much**
6/16/2025 20:53:45	81505@siswa.unimas.r	4	1	4	1	5	1	5	1	5	1	Cool	The design is well th	**None**
6/16/2025 20:54:54	81895@siswa.unimas.r	5	1	4	1	4	1	5	1	5	1	I like it	The design is straigh	**Nothing**
6/16/2025 20:56:05	81378@siswa.unimas.r	5	1	5	2	4	1	5	1	4	1	Functions well	Uncomplicated	**I would prefer the add**
6/16/2025 20:57:48	80329@siswa.unimas.r	5	1	5	1	5	1	5	1	4	1	Quite useful	Quite interesting	**Nothing**
6/16/2025 21:00:43	81780@siswa.unimas.r	5	1	5	1	5	1	5	1	5	1	I like the functional	Simple and easy	**Nothing**