SOUND RECOGNITION OF BIRD SPECIES USING MACHINE LEARNING ALGORITHMS

Nelson Lee Chin Sheng

Bachelor of Computer Science with Honours (Information Systems)

2025

UNIVERSITI MALAYSIA SARAWAK

THESIS STATUS ENDORSEMENT FORM

TITLE Sound Recognition of Bird Species Using Machine Learning Algorithms

ACADEMIC SESSION: 2024/2025 -2

NELSON LEE CHIN SHENG

(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	

Nelson

(AUTHOR'S SIGNATURE)

Validated by

(SUPERVISOR'S SIGNATURE)

Permanent Address

HUA JOO PARK, 3RD MILE
93200 KUCHING, SARAWAK

Date: 19TH 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

SOUND RECOGNITION OF BIRD SPECIES USING MACHINE LEARNING ALGORITHMS

NELSON LEE CHIN SHENG

This project is submitted in partial fulfilment of the requirements for the degree of

Bachelor of Computer Science with Honours (Information Systems)

Faculty of Computer Science and Information Technology

UNIVERSITI MALAYSIA SARAWAK
2025

PENGENALAN BUNYI SPESIES BURUNG MENGGUNAKAN ALGORITMA PEMBELAJARAN MESIN

NELSON LEE CHIN SHENG

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

Faculty of Computer Science and Information Technology

UNIVERSITI MALAYSIA SARAWAK
2025

DECLARATION

I confirm that this project represents my independent and original work. All content is my own creation, and I have properly cited and acknowledged any external sources used. No part of this work has been plagiarized from colleagues or other sources, nor has anyone else written any portion on my behalf.

Nelson

(NELSON LEE CHIN SHENG) 19 JUNE 2025

Matric No: 80357

ACKNOWLEDGEMENT

I extend my deepest appreciation to my supervisor, Dr. Mohammad Bin Hossin, for his exceptional guidance and mentorship throughout this project. His expert insights and dedicated support during our consultations were crucial to completing this work successfully and on time. I am also grateful to my examiner, Assoc. Prof. Dr. Jacey Lynn Minoi, whose thoughtful feedback and recommendations significantly enhanced the quality of this project.

I wish to thank our coordinator, Prof. Dr. Wang Yin Chai, for providing clear direction and formatting guidance. Additionally, I am thankful to my friends and course mates who generously offered their assistance and expertise during their own busy academic schedules, helping me resolve various project-related challenges.

Most importantly, I am profoundly grateful to my family for their constant emotional support and encouragement. Their unwavering faith in my abilities gave me the strength to persevere through difficulties and ultimately complete this project successfully.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Analysis of bird species using their sounds in the ecosystem is crucial for the effective conservation of natural habitats. However, this method requires significant expertise and knowledge on the part of the observer, and with increasing lengths of bird sound recordings taken every year, such expertise resources will become limited. The complexity and scale of this challenge necessitates an automated classification system, which can be effectively implemented through machine learning techniques. This paper examined the performance of three machine learning algorithms, Naïve Bayes, Support Vector Machines (SVM), and Central Neural Network (CNN), in classifying publicly accessible, crowd-sourced bird sound recordings. These algorithms were implemented using Python scripts and libraries. This work also utilizes the highest-performing machine learning model to develop a bird sound recognition application. The application was developed using React Native and Python, and evaluated following the System Usability Scale. The results of this study show that CNN was the most effective machine learning algorithm for classifying bird audio, with a classification accuracy of 0.3492, compared to Naive Bayes (0.0849) and SVM (0.2720). Through the System Usability Scale (SUS), the application obtained a SUS Score of 83.91, suggested users have positive experiences with the mobile application. This study shows the potential of CNN for bird audio classification, and further work could be done to enhance its accuracy and robustness, such as data augmentation, data expansion and examination of different data feature extraction and deep learning architecture.

ABSTRAK

Analisis spesies burung menggunakan bunyinya dalam ekosistem adalah penting untuk pemuliharaan habitat semula jadi yang berkesan. Walau bagaimanapun, kaedah ini memerlukan kepakaran dan pengetahuan yang ketara di pihak pemerhati, dan dengan peningkatan bilangan rakaman panggilan burung yang diambil setiap tahun, sumber kepakaran tersebut menjadi terhad. Kerumitan dan skala cabaran ini memerlukan sistem pengelasan automatik, yang boleh dilaksanakan dengan berkesan menggunakan teknik pembelajaran mesin. Kertas kerja ini menyiasat prestasi tiga algoritma pembelajaran mesin, Naïve Bayes, Support Vector Machine (SVM) dan Convolution Neural Network (CNN), dalam mengklasifikasikan rakaman panggilan burung yang boleh diakses secara umum dan sumber orang ramai. Algoritma ini dilaksanakan menggunakan skrip dan perpustakaan Python. Kerja ini juga memanfaatkan model pembelajaran mesin berprestasi tinggi untuk membangunkan aplikasi pengecaman panggilan burung. Aplikasi ini dibangunkan menggunakan React Native dan Python, dan dinilai menggunakan Skala Kebolehgunaan Sistem. Hasil kajian ini menunjukkan bahawa CNN adalah algoritma pembelajaran mesin yang paling berkesan untuk mengklasifikasikan panggilan burung, dengan ketepatan klasifikasi 0.3492, berbanding dengan Naïve Bayes (0.0849) dan SVM (0.2720). Melalui Skala Kebolehgunaan Sistem (SUS), aplikasi tersebut memperoleh Skor SUS yang sebanyak 83.91, mencadangkan pengguna mempunyai pengalaman positif dengan aplikasi mudah alih. Kajian ini menunjukkan potensi CNN untuk klasifikasi audio burung, dan kerja selanjutnya boleh dilakukan untuk meningkatkan ketepatan dan keteguhannya, seperti penambahan data, pengembangan data, dan pemeriksaan pengekstrakan ciri data serta seni bina pembelajaran mendalam yang berbeza.

CHAPTER 1: INTRODUCTION

Technological advancement has facilitated significant progress in the resolution of human problems. However, there are a number of issues that cannot be adequately addressed through conventional computer-based methods. In such cases, the application of machine learning can offer a viable alternative (Abdulkareem & Abdulazeez, 2021). This is especially true when such problems require solutions discovered through past relevant data rather than innate knowledge (Charbuty & Abdulazeez, 2021). One such problem is the recognition of bird species for global conservation efforts (Howard et al., 2021). The identification of these species can be achieved through the recognition of their distinctive vocalisations, which vary significantly between different species (Maclean & Triguero, 2023). A substantial body of research has been conducted with the objective of developing automated techniques for the classification of bird calls. This can be advantageous in the context of annotating a considerable quantity of unidentified recordings, a process which can be time-consuming if undertaken manually (Howard et al., 2021). Furthermore, manual verification can be unreliable due to subjective interpretations and inconsistencies (Sainburg et al., 2020). This study represents one of numerous efforts to develop a bird sound recognition application using machine learning techniques.

1.1 Problem statement

Monitoring the biodiversity in ecosystems is crucial for ensuring the conservation efforts can be carried out effectively. One method of identification of animal species can be achieved through the analysis of their sounds and calls (Ekpezu et al., 2022). This method is particularly applicable to birds, for which the recognition of sounds requires a significant degree of experience and knowledge on the part of the observer (Lehikoinen et al., 2023).

Ecological and anthropogenic factors have negatively impacted the population of the bird species in recent years. Research by Morrison et al. (2021) revealed a significant decline in both the intensity and diversity of bird soundscapes across a large number of sites (approximately 200,000) throughout North America and Europe. This reduction stems from a marked decrease in both avian species diversity and population numbers, resulting in substantial implications for ecosystem health and biodiversity maintenance.

To advance conservation efforts, researchers have developed various machine learning solutions that enable rapid and accurate bird species identification. Some of the proposed solutions favour the utilisation of visual identification (Kumar et al., 2024; Yang et al. 2024) given the inherent limitations associated with acoustic detection. These include the interference of ambient noise in the recorded data (Varghese et al., 2022). Nevertheless, the potential of acoustic detection should not be discounted, given its resilience to visual interference. In scenarios where solely acoustic identification is feasible, for instance in low-light or hazy environments, the deployment of acoustic detection solutions can prove advantageous (Kumon et al., 2022).

1.2 Research scope

The datasets used in this project will be exclusively collected from crowdsourced recordings on a website called Xeno-Canto. The recordings will be restricted to those created in and around the Sarawak region. The recordings are licensed for use and modified by the author for non-commercial purposes. A total of 433 recordings will be utilised in this study. This research will focus on implementing and comparing three specific machine learning algorithms for bird call classification: Naïve Bayes, Support Vector Machine (SVM), and Convolutional Neural Network (CNN). A distinct model will be developed for each algorithm, followed by a comprehensive

performance evaluation using established metrics. The best-performing model will then be integrated into the final deliverable: a practical application designed to identify bird species through their vocalizations.

1.3 Research objectives

The objectives of this research are as follows:

- Data collection and pre-processing of avian audio recordings.
- Develop classifier models using Python for three machine learning algorithms, Naive Bayes, Support Vector Machine (SVM) and Convolutional Neural Network (CNN).
- Evaluate the machine learning algorithms based on accuracy, precision, recall, F1 score and time efficiency.
- Utilise the best-performing machine learning algorithm to develop an Android mobile application.
- Examine the compatibility, functionality, and usability of the Android mobile application through tests such as device compatibility tests, functionality tests, and the System Usability Scale (SUS).

1.4 Research significance

The creation of an application for identifying bird species through their calls holds considerable significance across a number of domains, particularly in terms of its contribution to the accumulation of knowledge and the engagement of the wider community.

In terms of the contribution to the knowledge base, the application is a valuable tool for ornithological research, enabling the systematic collection and documentation of avian acoustic data. This improves understanding of species distribution, behavioural patterns and population dynamics. Furthermore, the accumulated data can support studies in bioacoustics.

In terms of the contribution to the community, this application makes ornithological knowledge more accessible. It also helps to educate professionals and amateur birdwatchers, encouraging them to take part in conservation. This application can also significantly enhance the accuracy and reliability of citizen-contributed data, leading to more precise and dependable scientific observations.

The use of machine learning in traditional ornithological techniques helps conserve and study bird species, advancing scientific knowledge and encouraging environmental stewardship. This makes specialised scientific knowledge more accessible while expanding our understanding of birds.

1.5 Thesis structure

The structural organization of this thesis comprises five distinct chapters, which are systematically arranged in the following manner:

Chapter 1 presents a comprehensive introduction to avian vocalization identification through machine learning methodologies. This chapter elucidates the fundamental requirements and inherent challenges associated with such identification protocols. Furthermore, it delineates the research scope, anticipated outcomes, and significance of this investigation. The chapter concludes with a systematic overview of the thesis structure.

Chapter 2 encompasses a thorough literature review of contemporary research in avian species acoustic recognition utilizing machine learning approaches. The discussion of the recent works related to this research are presented and analysed.

Chapter 3 explicates the research methodology in detail. This encompasses the systematic development of machine learning architectures, the iterative prototyping process of the bird sound recognition application, and the implementation of user feedback mechanisms.

In Chapter 4, the development, results and analysis of the models are discussed.

In Chapter 5, the development, results and analysis of the application are discussed.

In Chapter 6, the conclusion of the research is established. The results of this study are presented in the context of its initial objectives and significance. The chapter also outlines the future work that will be conducted as a result of this study.

1.6 Project Schedule

The implementation of a well-structured project schedule is paramount for ensuring timely project completion and optimal outcomes. Such a schedule functions as a critical monitoring tool for evaluating project progression against predetermined milestones and deadlines. The project's temporal framework encompasses two consecutive academic semesters, and its chronological organization has been systematically delineated through the utilization of a Gantt chart methodology, enabling comprehensive visualization and assessment of various project phases. The dates were formatted according to ISO 8061 set by the International Organization for Standardization (2019).

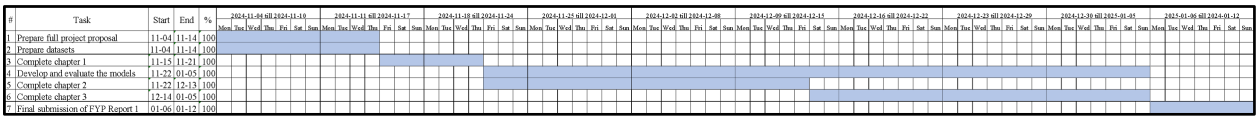


Figure 1.1: Gantt chart week 1-10

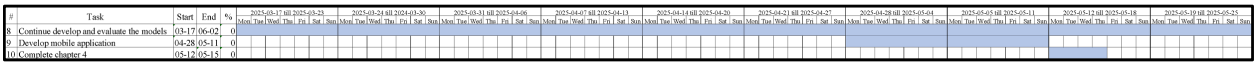


Figure 1.2: Gantt chart week 11-20

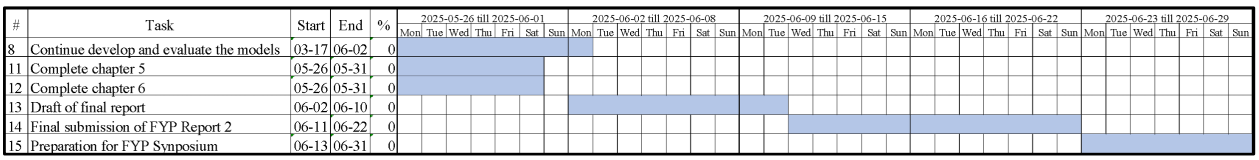


Figure 1.3: Gantt chart week 21-25

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

This chapter provides a comprehensive review of previous research, focusing on three key areas: the fundamental need for automated bird recognition systems, the various methods used to process acoustic data before its integration into ML classification models, and an analysis of different ML approaches, including Naïve Bayes, Support Vector Machines (SVM), and Convolutional Neural Networks (CNN). These elements are examined in the context of their effectiveness in bird species identification.

2.2 The need for bird sound recognition

Bird species monitoring plays a crucial role in biodiversity conservation efforts. According to the International Union for Conservation of Nature's Red List (IUCN, 2021) created in 2021, a total of 1,445 bird species, representing approximately 12.95% of the total number of species, have been classified as being at risk of extinction. It is of the utmost importance that strategies for the management and conservation of bird species are effective, and therefore, optimising bird monitoring represents a pivotal component in the evaluation of the status and condition of the species in question. One of the principal aspects of bird monitoring is the utilisation of birds' acoustics. This is due to the fact that there are relatively stable and distinct discriminating features in bird vocalisations between bird species (Green & Marler, 1979). Furthermore, distinguishing between individual birds based on their vocalisations is a viable approach, as each individual displays consistent vocalisation features over time, and the variation in vocalisations is greater between individuals than within them (Gibb et al., 2019).

Researchers increasingly favor passive acoustic monitoring (PAM) methods for bird species identification. This approach enables automated sound collection in the field and subsequent species identification from recorded vocalizations. PAM systems can effectively gather data across various geographical locations over extended time periods (Sedlacek et al., 2015; Wan, 2014; Wheeldon et al., 2019). These methods offer several advantages over traditional manual surveys: they require fewer resources, minimize disturbance to bird populations, provide broader monitoring coverage, and demonstrate greater efficiency (Xie et al., 2023).

Additionally, PAM methods overcome common limitations associated with manual observation, such as researcher bias and visibility issues caused by lighting conditions or physical obstacles. This makes PAM particularly valuable for detecting and monitoring elusive or rare bird species (Dong et al., 2013). However, the substantial volume of data generated by long-term PAM systems presents challenges for manual analysis (Kasten et al., 2012). Consequently, there is a pressing need for automated bird sound recognition systems to enhance the processing efficiency of acoustic recordings, thereby improving the overall effectiveness of bird monitoring programs (Xie et al., 2023).

Bird vocalisations can be categorised into two distinct categories: calls and songs (Payne, 1979). Bird vocalizations can be categorized into two primary types: calls and songs. Calls are characterized by their simplicity, consisting of either one (monosyllabic) or two (bisyllabic) sound elements. Songs, however, exhibit greater complexity, featuring a combination of diverse, multiple-syllable (multisyllabic) sounds produced in continuous sequences. While bird vocalizations demonstrate greater variability than human speech patterns, their fundamental structure remains relatively simple in comparison. Despite the simple nature of bird sounds, the conditions faced by field recording can be more challenging than those encountered when

recording human speech, making the recognition of bird sounds more challenging than voice-print recognition (Xie et al., 2023).

2.3 Methods employed to process the data prior to classification

Prior to classification, a variety of methodologies were employed with the objective of enhancing the overall performance of the classifying using ML techniques.

Kahl et al. (2021) transformed the audio files into spectrograms, which were treated as monochrome images. This permitted a variety of modifications to be made to the frequency and magnitude scaling, as well as the time step configurations. The researchers standardized their methodology by segmenting the audio data into three-second segments and applying a uniform sampling rate of 48 kHz. The data were then augmented in order to ensure that the model would be robust against the variations that occur in real-world scenarios. The focal data were combined in order to imitate the soundscape environment, thus ensuring that the model would be able to handle multi-label tasks.

In their research, Mehyadin et al. (2021) implemented a two-stage data preprocessing approach. Initially, they applied noise filtering techniques to remove unwanted acoustic interference, thus enhancing the quality of audio playback. Subsequently, they conducted feature extraction using Mel-frequency cepstral coefficients (MFCC), a method specifically chosen to identify and extract the distinctive characteristics of bird vocalizations.

Maclean and Triguero (2023) developed a systematic approach to bird call classification. First, they segmented audio recordings into equal-length portions and converted these segments into spectrogram images using short-time Fourier transformation. They then implemented a binary

classification system to detect the presence or absence of bird calls within these spectrograms, which enhanced the reliability of species identification in subsequent processing stages. A notable feature of their approach was the development of a "nocall" discriminator, derived from the binary classification dataset. This discriminator assessed the probability of bird call presence in each segment, which in turn informed the decision-making process of the multi-label classifier. This methodical approach significantly improved the accuracy of multi-label classification outcomes.

Zhang et al. (2023) implemented a comprehensive preprocessing framework that consisted of three main components: noise estimation and removal, silence detection, and feature extraction. Their approach to feature extraction was particularly thorough, incorporating multiple acoustic characteristics, including Mel-frequency cepstral coefficients (MFCC), Chroma features, Tonnetz features, and Log-Mel spectrograms. These diverse features were selected to capture a complete representation of bird vocalizations. The researchers enhanced their feature set through feature fusion, combining deep features extracted from multiple sources using Convolutional Neural Networks (CNNs) and Transformer encoder. To optimize computational efficiency, they applied Principal Component Analysis (PCA) to reduce the dimensionality of the combined features while minimizing redundant information.

2.4 Review on three selected machine learning algorithms

This section examines three prominent machine learning approaches utilized in classification tasks: Naïve Bayes, Support Vector Machines (SVM), Convolutional Neural Networks (CNN). Each of these methodologies offers distinct advantages and disadvantages in the context of bird sound classification, and their applications will be analysed in detail in the following sections.

2.4.1 Naïve Bayes

Naïve Bayes classification demonstrates particular effectiveness in analyzing high-dimensional data (Abdulrazzaq & Saeed, 2019). This machine learning algorithm operates on two fundamental assumptions: all attributes carry equal importance a priori, and all attributes maintain statistical independence relative to the target class (Tivarekar et al., 2018). Although these assumptions may appear oversimplified in practical applications, research by Tivarekar et al. (2018) has shown that the algorithm maintains robust performance. The classification process involves assigning items to the class with the highest posterior probability (Tivarekar et al., 2018). Adopted from paper written by Tivarekar et al. (2018), the equation of the posterior probability is calculated as below:

$$P(H|E) = \frac{P(E1|H) * P(E2|H) *\ldots* P(En|H) * P(H)}{P(E)}$$

where "H" represents the hypothesis in a given scenario, "E" represents the evidence supporting hypothesis "H", "P(H)" denotes the prior probability (initial probability of the hypothesis), "P(E)" represents the probability of the evidence occurring, "P(E|H)" indicates the probability of evidence "E" occurring, given that hypothesis "H" is true, "P(H|E)" represents the posterior probability (the probability that hypothesis "H" is true, given the occurrence of evidence "E").

Naïve Bayes is effective in dealing with attributes that the user may not have a comprehensive understanding of. It also demonstrates proficiency in classifying using attributes based on a straightforward, scheme-specific approach (Tivarekar et al., 2018). Furthermore, its runtime is relatively low, particularly in comparison to more intricate ML algorithms such as decision trees and Multilayer Perceptron (Mehyadin et al., 2021).

Tivarekar et al. (2018) identified several challenges in the implementation of Naïve Bayes classification. A primary concern arises when attributes used for classification exhibit dependencies or redundancies, contradicting the algorithm's fundamental assumption of independence. However, researchers have developed mitigation strategies, such as implementing forward selection algorithms that can detect potentially redundant attributes before their inclusion and evaluate attribute quality within training set subsets. Another significant limitation of Naïve Bayes classification emerges in multi-label scenarios. This is particularly evident in bird call classification when attempting to identify multiple species vocalizing simultaneously with similar intensity levels in a single recording. Such situations challenge the algorithm's basic classification framework, which is primarily designed for single-label classification tasks.

In their 2018 study, Tivarekar and colleagues employed the Naïve Bayes classification algorithm to identify four distinct avian species commonly observed in the Indian region. The results demonstrated that this ML algorithm was highly effective, with 100% recall and precision. The same algorithm was employed in a subsequent study conducted by Mehyadin and other authors (2021) to identify species and emotions through female bird calls using Mel-frequency cepstral coefficients (MFCCs). However, the authors observed that the algorithm exhibited the lowest accuracy compared to other ML algorithms that they had tested.

2.4.2 Support Vector Machines (SVM)

Support Vector Machines (SVM) stand out as a unique classification algorithm rooted in the fundamental principle of creating a separating hyperplane (Vapnik, 2013). This machine learning approach focuses on determining the most strategic boundary between different data categories. The core objective of SVM is to locate a hyperplane that maximizes the margin between

distinct classes, essentially finding the most robust line of separation that can effectively distinguish between data points (The MathWorks, 2020). When dealing with linearly separable data, the SVM algorithm meticulously seeks to establish the widest possible separation between classes. This approach ensures that the classification boundary provides the most significant buffer between different data groups (Vapnik, 2013). Researchers like Rai and colleagues (2016) have visually represented this concept, showcasing how SVMs can effectively segregate two distinct classes by identifying the optimal hyperplane that creates the most pronounced distinction between them, which is shown in Figure 2.1.

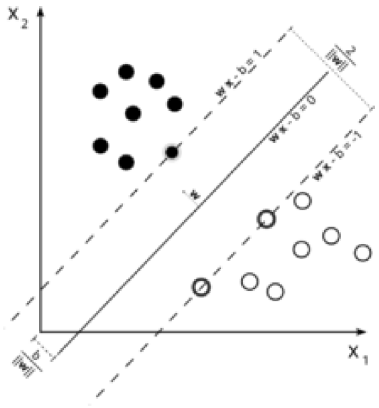


Figure 2.1: Sample representation of SVM model adopted from Rai et al. (2016)

SVM is highly effective in binary labelling situations, particularly in binary sound classification (Ekpezu et al., 2022). It also functions effectively as a baseline classifier when identifying optimal pre-classification methods, such as different methodologies for feature extraction (Ramashini et al., 2022). This is due to its proven high accuracy results, which are supported by its defined statistical learning theory and minimal structural risk properties (Fagerlund, 2007).

While Support Vector Machines (SVM) demonstrate remarkable performance in binary classification scenarios, where data can be cleanly separated into two distinct categories, their effectiveness

significantly diminishes when confronted with multi-label classification challenges. A review of the literature by Ekpezu et al. (2022) found that SVM is not always the optimal choice for multi-label classification. For example, when attempting to classify multiple bird species simultaneously, SVM can be a cumbersome approach.

The SVM was employed to identify the optimal methodology for feature extraction (Ramashini et al., 2022). Although classification was not the main objective of the study, it has been demonstrated to be highly accurate when combined with the most optimal feature extraction method. In another study, Demir et al. (2020) employed SVM for the classification of environmental sound. The findings indicated that the performance was relatively accurate when compared to other methods reviewed by the authors. The authors' proposed solution demonstrated superior performance, as evidenced by achieving the highest accuracy across two of the three datasets used in the evaluation.

2.4.3 Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNNs) represent a sophisticated machine learning architecture that mimics the intricate processing mechanisms of the human brain (Kumar et al., 2024). Structured with three fundamental layers—input, hidden, and output—CNNs are engineered to autonomously discover and extract complex patterns within datasets through interconnected neural networks. These computational models possess a remarkable capacity to learn and identify hidden relationships independently across different layers, making them a powerful tool for trend detection and pattern recognition (Dwivedi et al., 2019; Maclean & Triguero, 2023). Adapted from Kumar et al. (2024), Figure 2.2 shows a simplified representation of the CNN model.

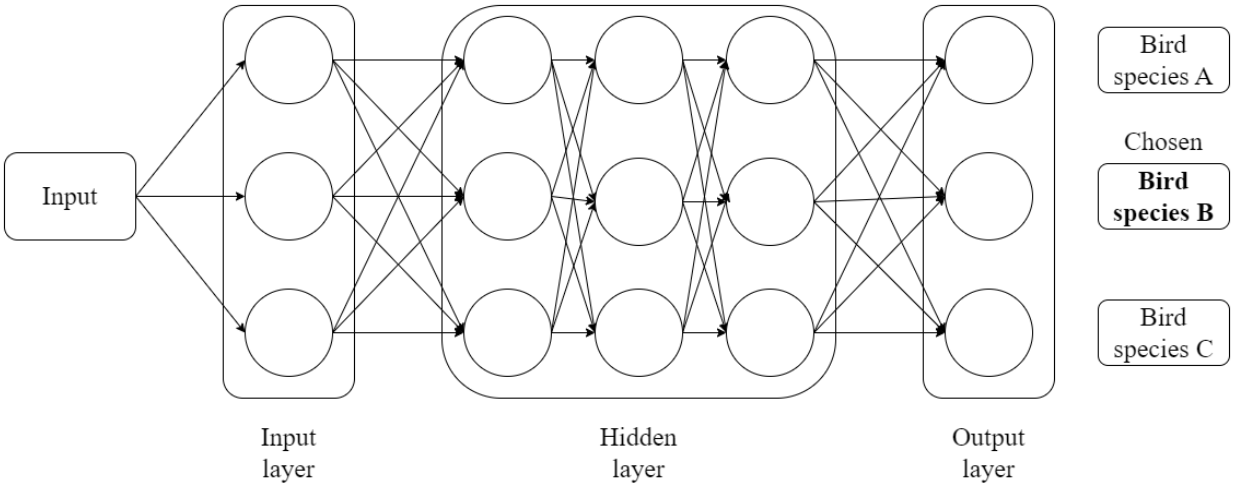


Figure 2.2: Simplified representation of CNN model adopted from Kumar et al. (2024)

The convolutional layer within a Convolutional Neural Network (CNN) offers a distinctive capability to dynamically transform and compress input data into a more compact, semantically meaningful representation that significantly enhances the network's classification performance. By strategically distilling complex information into more focused and relevant features, this layer enables the CNN to extract and prioritize the most critical characteristics essential for accurate data categorization (Maclean & Triguero, 2023). Furthermore, it is robust in handling domain mismatch, for example, when a model is trained on single-labelled datasets and evaluated on multi-labelling tasks (Maclean & Triguero, 2023). Additionally, CNNs have the capacity to self-organise, perform real-time operations and undergo adaptive learning (Dwivedi et al., 2018).

One of the challenges frequently encountered by researchers utilizing CNN for classification is the necessity for a substantial amount of data to achieve optimal performance (Dwivedi et al., 2019; Gupta et al., 2021). Obtaining manually labelled datasets from scratch is a time-consuming and resource-intensive process (Gupta et al., 2021). Furthermore, CNN is more computationally expensive than other machine learning algorithms, such as SVM (Dwivedi et al., 2019).

Convolutional Neural Networks (CNNs) have become a widely adopted method for categorizing bird sound recordings in the field of avian bioacoustics (Maclean & Triguero, 2023; Kahl et al., 2021). Kahl et al. (2021) opted to employ CNNs for bird vocalization classification in their research, citing the proven effectiveness of comparable machine learning approaches in related areas of study. Although no direct comparisons were identified by the authors, the model demonstrated competitive performance relative to other attempts in a similar domain. In their 2023 paper, Maclean and Triguero report that the top 10 solutions for BirdCLEF 2021, a competition for identifying bird species in recordings of different bird sounds, employed CNN approach. Furthermore, the authors developed a CNN solution that achieved the highest score in the private category of the BirdCLEF 2021 leaderboard.

2.5 Bird sound recognition application

Xie and colleagues (2023) conducted a review and compiled a comprehensive list of applications available on the market, as shown in Table 2.1. The applications were divided into two main categories: mobile phone applications (Apps) and semi-automatic software tools.

The former category typically incorporates a pre-installed trained model, which only requires bird vocalisations as inputs, a feature that has led to its popularity among users with limited software development expertise (Xie et al., 2023). A proportion of these applications have the model installed on the user's phone, enabling offline functionality; others require a connection to the server for app functionality. While apps are convenient and user-friendly, the number of species that the apps can identify is fixed until the developer updates the apps.

Semi-automatic software tools are designed for users, particularly biologists and bioacoustic researchers, to create their own recognition models (Xie et al., 2023). This process

involves the time-consuming task of manually annotating spectrograms. To aid users, some of these tools incorporate syllable analysis functions that emphasize more ecologically significant annotations or measurements. These software solutions streamline the modeling process by automating various stages, including data annotation, model training and testing, and evaluation. This automation is typically achieved through scripts or a user-friendly graphical interface (GUI).

Table 2.1: Comparison of available bird sound recognition application in the market

Application	Manual Annotation	Input	Method	Recognition target	Interface	Access	Platform
Animal Sound Identifier (Ovaskainen et al., 2018)	Yes	1-min segment	Template matching based on cross-correlation	Species recognition	No GUI	Free	Computer
Arbimon (Connection, 2022)	Yes	Syllable	Template matching	Species recognition	Web	Free / Paid	Computer
AviaNZ (AviaNZ, 2021)	Yes	Manual setting	Inherent Wavelet recogniser, can extend for other CNNs	Species recognition	GUI	Free	Computer
Avisoft-SASLab (Avisoft Bioacoustics, 2022)	Yes	Syllable	Axis-parallel threshold; Linear discriminant analysis	Syllable recognition	GUI	Paid	Computer
BirdNET (Cornell Lab of Ornithology, 2022a)	No	3-s segment	BirdNET(157 layers CNN with Resblock)	Species recognition	GUI	Free	IOS / Android

ChirpOMatic Bird Song ID (Spiny Software, 2020)	No	12-s segment	ML	Species recognition	GUI	Paid	IOS
Kaleidoscope Pro (Wildlife Acoustics, 2022)	Yes	Syllable	Hidden Markov Model + K-means clustering algorithm	Species cluster	GUI	Paid	Computer
Luscinia (Lachlan, 2016)	Yes	Syllable	Dynamic time warping	Syllable recognition	GUI	Free	Computer
Merlin Bird ID (Cornell Lab of Ornithology, 2022b)	No	Not given	Non-specific AI model	Species recognition	GUI	Free	IOS / Android
MonitoR (Hafner et al., 2018)	Yes	Syllable	Template matching based on binary-point matchinging	Species recognition	No GUI	Free	Computer
RavenPro (K. Lisa Yang Center for Conservation Bioacoustics, 2022)	Yes	Syllable	Not given	Species recognition	GUI	Paid	Computer
Shiny_PNW_Cnet (Ruff, 2022)	Yes	12-s segment	PNW-Cnet (contains four convolutiona l layers and two fully connected layers)	Species recognition	GUI	Free	Computer

Smart Bird ID (Yellow Cardinal Inc, 2022)	No	Not given	Not given	Species recognition	GUI	Free	IOS / Android
Sound Analysis Pro (Tchernichovski et al., 2019)	Yes	Syllable	Nearest neighbour clustering algorithm	Syllable cluster	GUI	Free	Computer
SoundClass (Silva, 2022)	Yes	Not given	Inherent VGG16, can extend for other CNNs	Species recognition	GUI	Free	Computer
TadariDeep (Didier and Yves, 2022)	Yes	1-s segment	Non-specific DL model	Species recognition	GUI	Free	Computer

Note. Adopted from Xie et al., 2023

2.6 Summary

To conclude, the development of machine learning-based bird sound recognition models emerged as a response to the challenge of classifying extensive bird sound recordings with limited expert availability. This review explored various pre-processing methodologies proposed in earlier studies, noting both commonalities and variations in approaches. Additionally, the review investigated the application of diverse machine learning (ML) techniques in identifying bird vocalizations.

CHAPTER 3: METHODOLOGY

3.1 Introduction

This chapter discusses the methodology used for this research project. Literature review, data preparation, machine learning algorithms comparison and evaluation, application development, and documentation will be done in this project. The application development will be following the prototyping model, in which the prototyping will be iterated until a satisfactory product is produced, which will then be developed into the final product for user evaluation.

3.2 Research project methodology

This section will present an overview of the methodology employed in this research project. The methodology commences with a literature review in which insights are provided in Chapter 2. The subsequent phases are data collection and preparation, which is followed by an evaluation of machine learning classification. Subsequently, a system will be developed to illustrate the outcome of classification of bird sounds. The development cycle will be conducted in accordance with the prototyping model. Finally, the project will be documented. This is the stage at which a thesis report, a research paper and presentation slides are prepared. Figure 3.1 illustrated the overview of the methodologies used in the study.

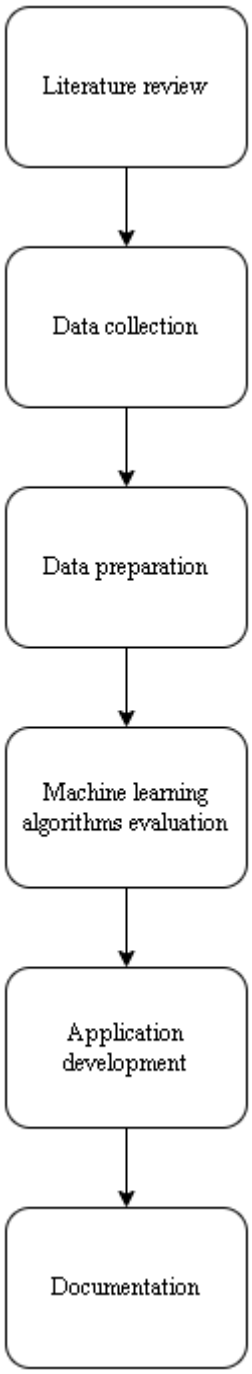


Figure 3.1: Steps of project methodology used in this study

3.3 Step 1 - Literature review

In this step, an examination of existing literature and research on bird sound classification

is undertaken. This step encompasses an investigation of the historical development of bird sound classification, an analysis of the challenges encountered in the classification of bird sounds, and a comparison of the machine learning techniques employed in classifying bird sounds. The objective of this step is to assess the current state of bird sound classification using machine learning techniques, as well as to evaluate the approaches taken to ensure the accuracy of the classification of bird sounds. The review is written in the Chapter 2 of this report.

3.4 Step 2 - Data collection

For this research, the dataset is the audio recordings obtained from Xeno-Canto website. These recordings are taken within and surrounding the Sarawak region, with the quality of the recording above the "D" rating. The parameter of the source is as follows: "box%3A0.358%2C109.183%2C5.797%2C116.149", "q>D". This amounts to 433 recordings of bird sounds with its associated metadata recorded in several JSON files. These recordings were given the licenses that permitted for non-commercial alteration and distribution, allowing the datasets to be processed as needed for this research without infringing the terms of use of these recordings. As illustrated in Table 3.1, the table provides a comprehensive overview of the sample count for each class utilised in the dataset for the present study.

Table 3.1: Sample count of each class in dataset

Class	**Sample count**
Ashy Tailorbird	7

Asian Glossy Starling	6
Black-and-yellow Broadbill	7
Blue-eared Barbet	15
Blue-throated Bee-eater	5
Bold-striped Tit-Babbler	19
Bornean Barbet	5
Bornean Black-capped Babbler	8
Bornean Black Magpie	24
Bornean Blue Flycatcher	6
Bornean Bulbul	6
Brown-throated Sunbird	9
Brown Fulvetta	10

Chestnut-rumped Babbler	7
Cinereous Bulbul	7
Cream-vented Bulbul	7
Crested Serpent Eagle	12
Crimson Sunbird	7
Dark-necked Tailorbird	14
Dulit Frogmouth	6
Eurasian Tree Sparrow	5
Fluffy-backed Tit-Babbler	12
Golden-whiskered Barbet	18
Greater Coucal	8
Greater Racket-tailed Drongo	6

Grey-hooded Babbler	15
Hooded Pitta	7
Indian Cuckoo	8
Lesser Coucal	5
Maroon-breasted Philentoma	5
Moustached Babbler	5
Philippine Cuckoo-Dove	7
Pied Triller	8
Plaintive Cuckoo	9
Red-bearded Bee-eater	10
Red-crowned Barbet	8
Red-legged Crake	5

Red-throated Barbet	9
Rufous-crowned Babbler	10
Rufous-tailed Shama	6
Rufous-tailed Tailorbird	12
Scaly-breasted Munia	6
Scaly-crowned Babbler	6
Short-tailed Babbler	6
Sooty-capped Babbler	5
Spectacled Bulbul	6
Spotted Dove	6
Streaked Bulbul	5
Sunda Scimitar Babbler	6

White-breasted Waterhen	6
White-chested Babbler	5
Yellow-vented Bulbul	6
Zebra Dove	5

3.5 Step 3 - Data preparation

The sample rate of the audio files will be normalised to 32 kHz using Librosa library following the guidelines provided by BirdCLEF 2021, and the peak loudness level of the audio will be normalized to -3db (Howard et al., 2021). Subsequently, the files will undergo denoising to enhance the accuracy of the bird sound identification in real-world environments using the spectrum gating noise reduction algorithm in the "noisereduce" Python library (Sainburg et al., 2020). For detecting silent segments within bird sound fragments, the average envelope threshold method was employed (Zhang et al., 2023). This technique analyzes the average envelope of frames within a sliding window to identify silent portions. As described by Zhang et al. (2023), the method is represented by the following equation:

$$\underline{y}\,[n] = \frac{1}{L} \sum_{k=-\frac{L-1}{2}}^{\frac{L-1}{2}} |y[n+k]|$$

$$mask[n] = \left\{True, if\ \underline{y}\,[n] > threshold;\ False, otherwise\right\}$$

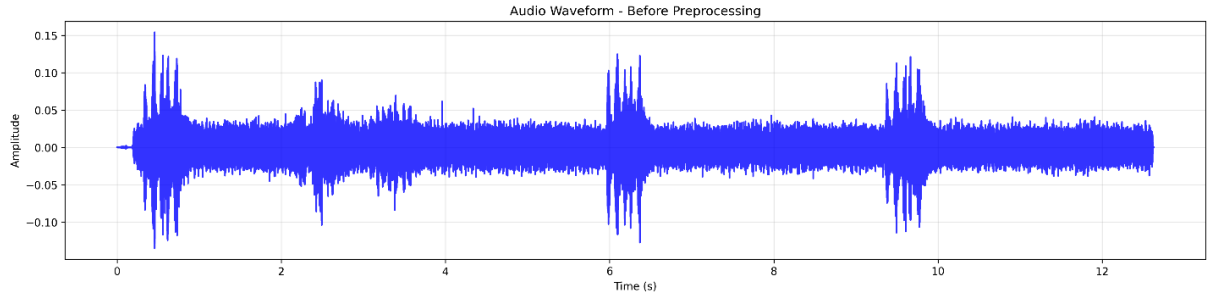
In the equation "$y[n]$" denotes the initial "n" frames of the bird sound, "L" represents the sliding window duration. "$\underline{y}\ [n]$ " represents the mean envelope within the window, "$mask[n]$" determines whether the first "n" frame is considered as a silent frame, and "threshold" represents the predefined threshold value. By removing silent frames, the analysis can focus solely on valid bird vocalizations. Subsequently, the pre-processed audio will be divided into 3-second segments.

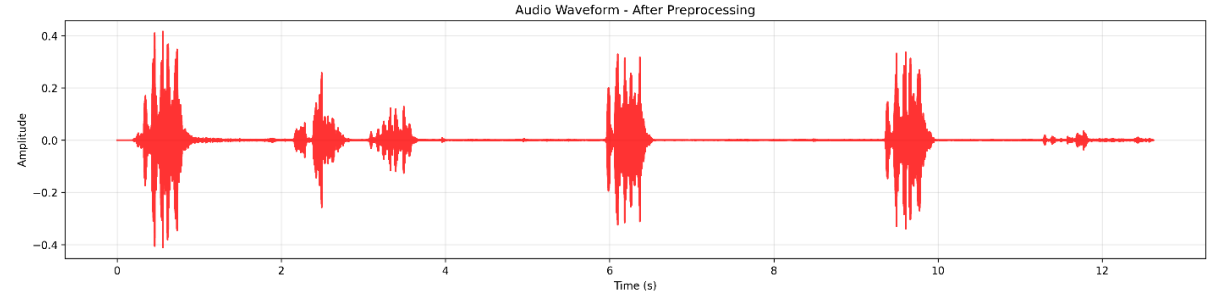
The next step involves generating log-mel spectrograms of the bird sounds, as image files serve as effective inputs for machine learning algorithms (Maclean & Triguero, 2023). This process begins with the application of short-time Fourier transformation (STFT) to convert audio segments into spectrograms (Zhang et al., 2023), followed by a logarithmic transformation to produce log-mel spectrograms. For ML classification, a combination of acoustic features will be extracted, including Log-mel spectrograms, Mel frequency cepstral coefficients (MFCCs), Chroma, and Tonnetz features.

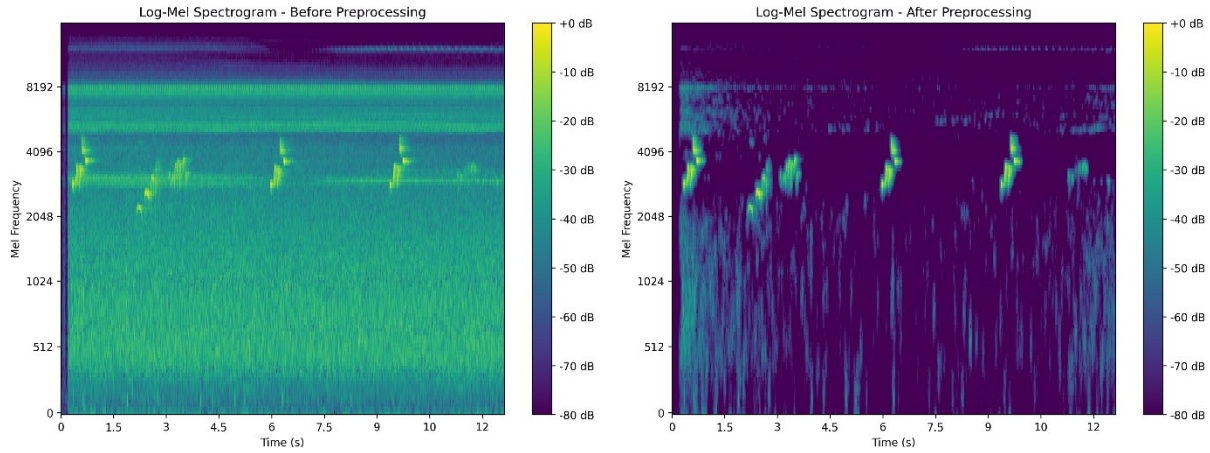
Table 3.2 illustrates the summary of the steps taken for data preparation. Figure 3.2 illustrates the features extracted before and after pre-processing steps were taken, excluding the 3-second segmentation. As illustrated in Figure 3.2, a comparison is presented of diagrams of waveform, log-mel spectrogram, MFCC, chroma and tonnetz between before and after the pre-processing steps have been applied to an audio sample, with the exception of the 3-second segmentation.

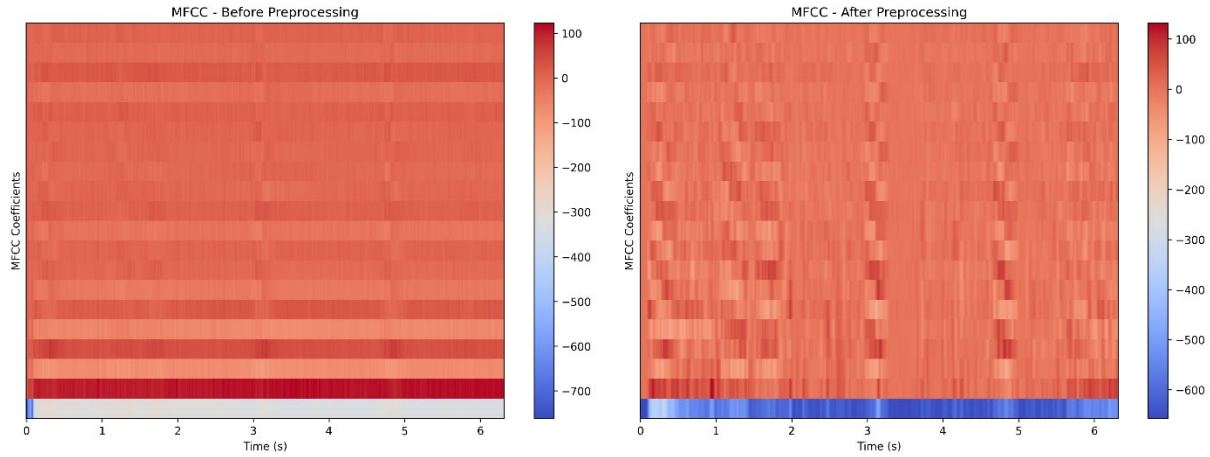
Table 3.2: Summary of steps taken for data preparation

No.	Preparation steps	Details
1.	Data sourcing	Xeno-Canto Bird Recordings in and from Sarawak region
2.	Normalisation	Resample the audio to 32kHz
3.	Denoising	Spectral Noise Gating
4.	Silence detection	Average envelope threshold method
5.	Segmentation	3-seconds audio segmentation
8.	Transformation into image	Short-time Fourier transformation Logarithmic transformation
9.	Features extraction	Extract Log-mel spectrogram, Mel frequency cepstral coefficients, Chroma and Tonnetz features


Audio Waveform - Before Preprocessing
Amplitude
Time (s)


Audio Waveform - After Preprocessing
Amplitude
Time (s)


Log-Mel Spectrogram - Before Preprocessing
Log-Mel Spectrogram - After Preprocessing
Mel Frequency
Time (s)


MFCC - Before Preprocessing
MFCC - After Preprocessing
MFCC Coefficients
Time (s)

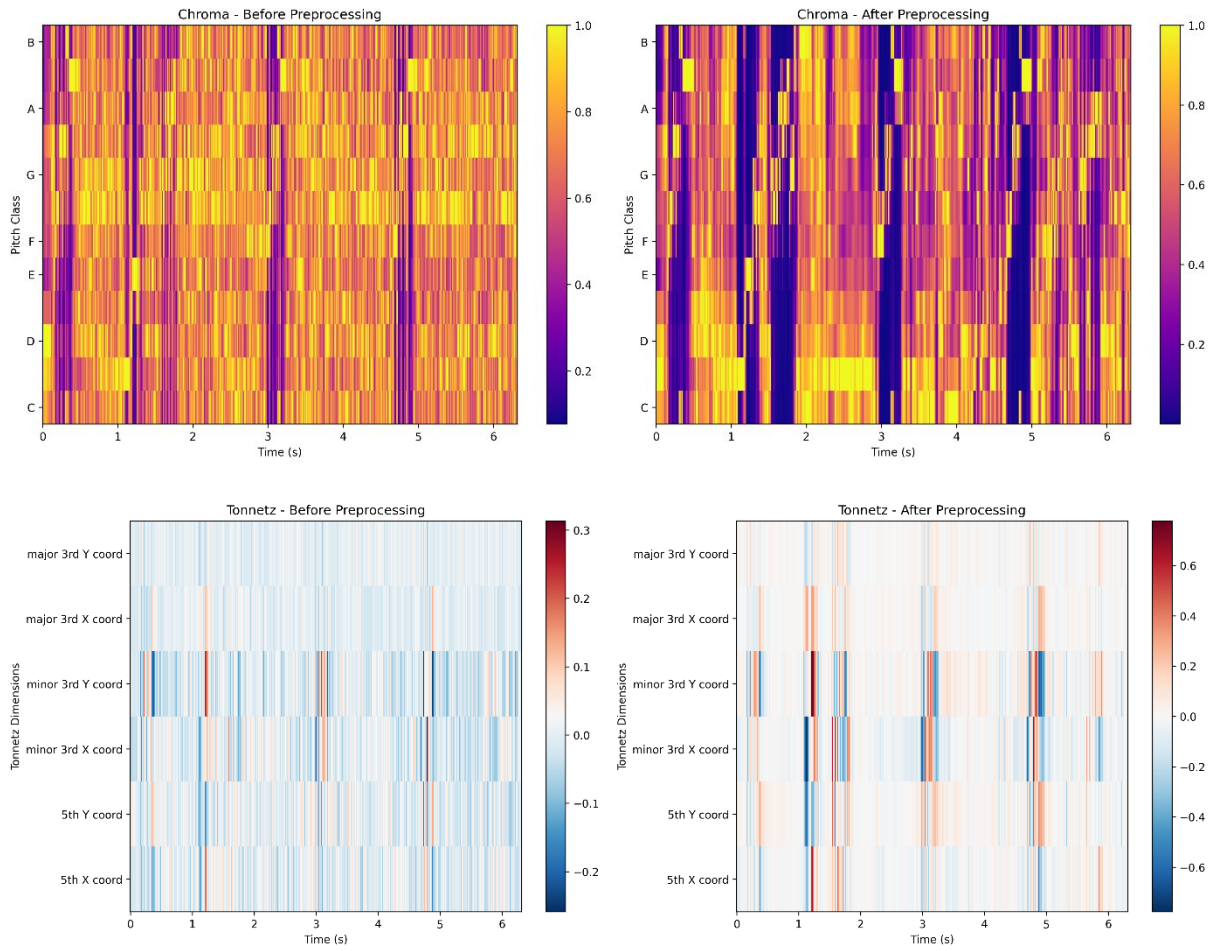


Figure 3.2: Comparison of diagrams of wave-form, log-mel spectrogram, MFCC, chroma and tonnetz between before and after the pre-processing steps

3.6 Step 4 - Machine learning algorithms evaluation

The development of three classification models utilising machine learning algorithms will be conducted using Python libraries, namely Naïve Bayes, Support Vector Machines (SVM), and Convolutional Neural Networks (CNN). A 5-fold cross-validation split of the feature vector, with equal proportions of classes, is created for the purpose of training and assessing ML algorithms. This approach is employed to prevent the models from overfitting to the testing data, thereby facilitating a more accurate assessment of the models' generalisation capabilities with respect to

unseen data.

The performance of the models will be assessed using multiple metrics, including accuracy, precision, recall, f1-score, as well as the computational resources and time required for bird sound classification post-training.

Accuracy is defined as the ratio of correctly identified samples over the total number of samples (Zhang et al., 2023). This metric is used to quantify the correctness of the model and is calculated as follows:

$$Accuracy = \frac{True\ Positive + True\ Negative}{True\ Positive + True\ Negative + False\ Positive + False\ Negative}$$

The "True Positive" category refers to cases where the model correctly identifies positive instances, while the "True Negative" category refers to cases where the model correctly identifies negative instances. "False Positive" and "False Negative" categories refer to cases where the model incorrectly identifies instances as positive and negative respectively.

Recall, as defined by Zhang et al. (2023), is the ratio of correctly predicted samples to the total number of samples predicted as positive by the models. This metric measures the proportion of true-positive cases identified by the model. It is calculated as follows:

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Precision, as defined by Zhang et al. (2023), is the ratio of correctly identified positive samples to the total number of samples labeled as positive. This metric assesses the accuracy of the models when classifying cases as positive. The formula for precision is:

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

Where "False Positive" refers to instances where the model incorrectly identifies a case as positive when it should be negative. Precision measures the model's ability to avoid labelling negative instances as positive.

The F1 score is a metric that provides a balanced measure of the model's performance by measuring both recall and precision within a single metric (Zhang et al., 2023). This enables the assessment of the model's overall performance in handling imbalanced data, thereby serving as the primary evaluation metric. It is calculated as follows:

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

The models will also be evaluated in terms of the computational resources and time required to classify the bird sounds after training. This will ensure that the pre-trained models are able to run without large amounts of computing power or time, as the application will be used in remote locations where internet access may not be readily available to connect to a server with better computer specifications. Fine-tuning the parameters of the most powerful pre-trained model using random-selection algorithms will be carried out to ensure that the model is able to run as efficiently as possible.

As demonstrated in Figure 3.3, the sequence of steps involved in the process, from data preparation to the development of a bird sound classification application using the most effective model, is illustrated.

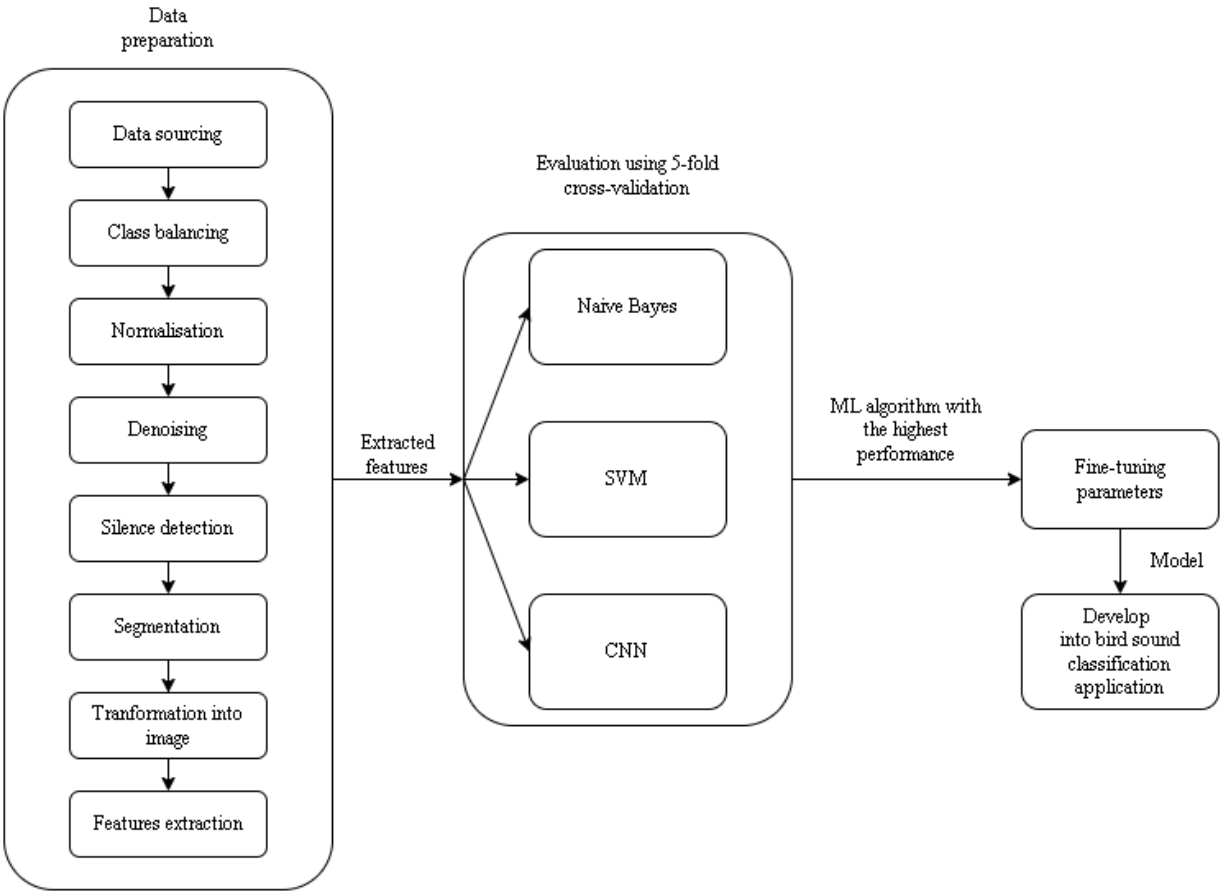


Figure 3.3: The overview architecture of data preparation and machine learning algorithms evaluation before the application development

3.7 Step 5 - Application development

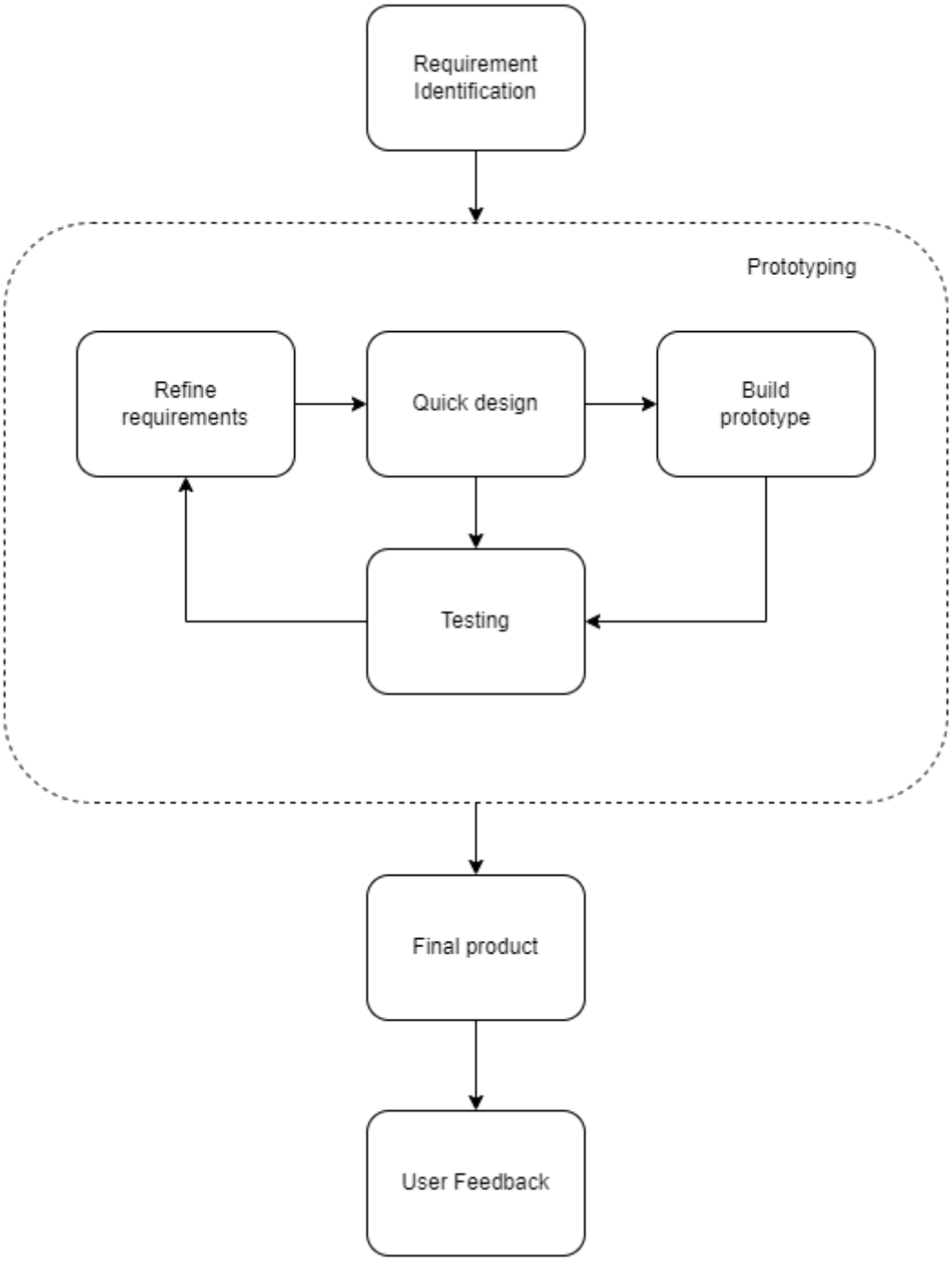


Figure 3.4: The overall flow of the prototyping of bird sound recognition application

A key outcome of this research is the development of an application designed to identify

the bird sounds. The application's development will follow the prototyping model, a systematic approach that emphasizes iterative design and user feedback. This model was selected for its capacity to facilitate the modification of the product design in response to evaluation outcomes and to enable the early detection and rectification of errors within the development process. This approach allows for a more aligned development of the application with the specified requirements and available resources that may change during development, as opposed to undertaking a complete redesign at the conclusion of the development cycle. Figure 3.4 illustrates the overall flow of the development of the bird sound recognition application.

3.7.1 Phase 1 - Requirement identification

The initiation of the application's development is contingent upon the identification of the project's requirements. This preliminary step involves the examination of the product development and feature requirements, with the objective being to ensure that subsequent development steps can be executed in a streamlined and robust manner. This process also serves to ensure that the scope of product development remains confined to the project's defined scope, thereby facilitating the effective execution of the project requirements.

3.7.1.1 Application development requirement

The identification of the application's development requirements is instrumental in ensuring its compatibility with standard devices. The hardware and software prerequisites are outlined in Tables 3.3 and 3.4, respectively, providing a foundation for the subsequent phases of the development process. These are identified through literature review done in Chapter 2.

The application is designed with common Android devices in mind, with the interfaces being interacted with using touch screens. The microphone on the devices will also be utilised to record sounds prior to classification.

The development of the application consists of two primary components: the user interface in the front-end and the sound pre-processing and classification system in the back-end. React Native is used to program the front-end, while Python is used to program the back-end of the application. These scripting languages have been utilized to reduce the complexity of developing the application. The utilisation of development tools and environment is instrumental in ensuring the seamless and effective development of the application.

3.7.2 Phase 2 - Prototyping

Once the initial requirements have been examined and documented, the prototype of the application will be promptly designed and developed. The prototyping will be primarily focused on the functionality and user-friendliness of the application, with the objective of ensuring that the application can function outside of the development environment. The requirement and design of the prototypes will be reiterated as required by the evaluations provided by the supervisor. This prototyping phase will be terminated once a substantial number of the application's requirements have been identified and resolved through the functionality’s development and bug fixes of the application.

3.7.2.1 Quick design

The quick design of the application is crucial in ensuring the efficiency of the product

development process. The process is primarily facilitated by a graph drawing software tool known as Draw.io, which is distinguished by its simplicity and ease of use, rendering it a suitable candidate for the design of Unified Modelling Language (UML) diagrams. Typically, these diagrams consist of simple shapes and lines, offering a simplified graphical representation of the application design. A key feature of UML diagrams is the presence of nodes, which are connected by edges to illustrate the relationships between various application elements. This process will be reiterated as many as needed.

3.7.2.1.1 Use case diagram

As illustrated in Figure 3.5, the use case diagram depicts the interaction between an actor, otherwise referred to as a user, and an application, along with the functionalities of the proposed application.

The user will input the recording of the sounds, which will then be pre-processed by the application in order to extract the necessary features for classification. Then, the trained model will identify the species of bird from the recorded sound. The result of the recognition will be displayed to the user, alongside with all the relevant information. Furthermore, the user may submit feedback on the audio recording, including suggestions for the classification.

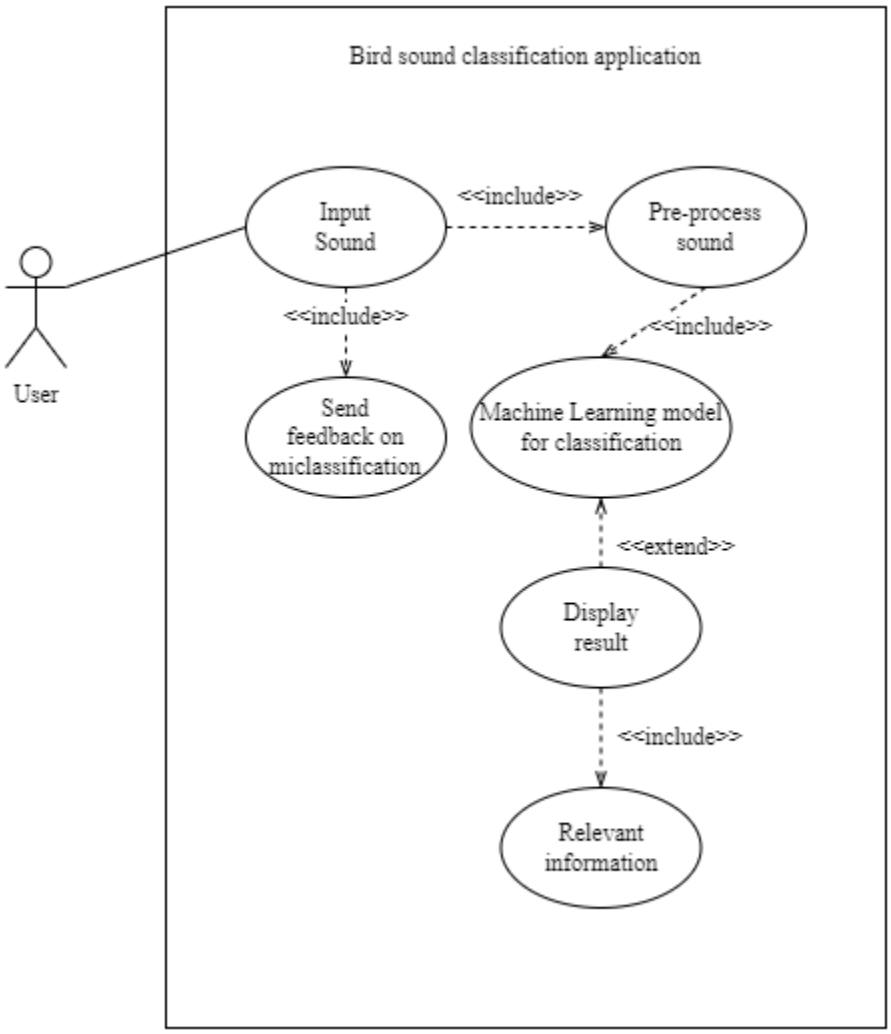


Figure 3.5: Use case diagram of bird sound recognition application

3.7.2.1.2 Class diagram

A class diagram visually depicts an application's structure by illustrating its classes, their attributes and functions, and the interconnections among these elements. The class diagram is a valuable tool for understanding the structure of an application and the relationships between its components.

As illustrated in Figure 3.6, the diagram depicts the classes utilised in the application, including

“User”, “Sound”, “MachineLearning”, “Model”, and “SendFeedback”. The diagram demonstrates the inheritance relationship between the “Model” classes and the “MachineLearning” abstract class, and the association relationship between the “User” class and the “Sound” and “MachineLearning” classes. The diagram also illustrates the association relation between the “Sound” and “SendFeedback” classes.

Each class possesses its own functions and attributes.The "User" class has public functions, namely “uploadSound()” and “displayResult()”. The “Sound” class has a private attribute, “soundFile”, and public functions, namely “getContent()” and “getFeatures()”. The “SendFeedback” has private attributes, “correctedLabel” and “comment”, and public functions, “writeComment()” and “send(file)”, in which “send(file)” is dependent on the sound file. The “MachineLearning” class possesses private attributes, “model” and “result”, while the “Model” class incorporates a public function, “classifyBirdSound(file)”, which is dependent on the sound file.

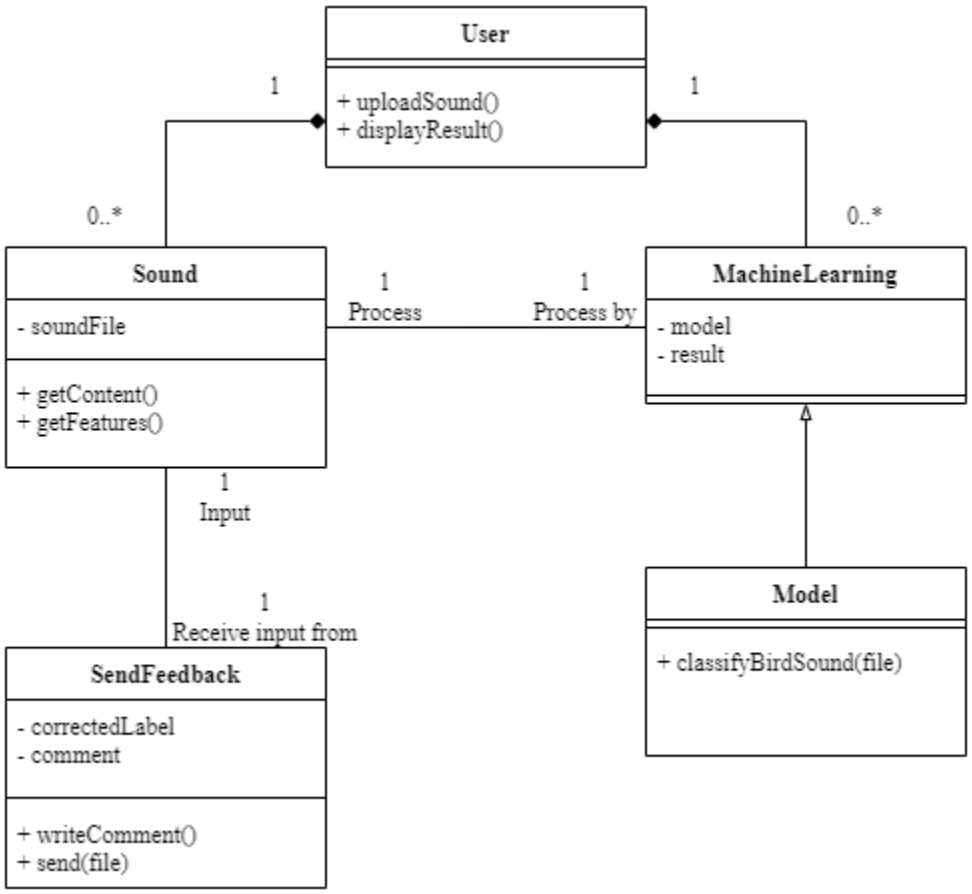


Figure 3.6: Class diagram of bird sound recognition application

3.7.2.1.3 Sequence diagram

A sequence diagram is used to illustrate the interactions between objects or components in an application, as well as the sequence of data transfer between them. It is employed to model the dynamic behaviour of an application and the interactions between objects or components in the application.

Figure 3.7 presents the sequence diagram of the bird sound recognition application comprising the following elements. The “User” object is representative of the user interacting with the application, whereas the “Bird sound recognition application” object is representative of the

application itself. The "Machine learning system" component in the diagram represents the classification model employed to identify and categorize bird vocalizations.

The “User” object initiates the process by transmitting the sound recording containing bird sounds to the “Bird sound recognition application”. The “Bird sound recognition application” object then processes the recording and transmits “Processed features” to the “Machine learning system” for the ML model to perform the classification tasks. Upon completion of the classification process by the “Machine learning system”, the results are returned to the application, which subsequently relays them alongside with relevant information to the “User” object for display.

Figure 3.8 presents the sequence diagram of sending feedback on the corrected label of the sound comprising the following elements. The “User” object is representative of the user interacting with the application, whereas the “Bird sound recognition application” object is representative of the application itself. The “Feedback server” is representative of the server receiving feedback from the user.

The “User” object initiates the process by transmitting the sound recording containing bird sounds and the comment to the “Bird sound recognition application”. The “Bird sound recognition application” object then processes the feedback and transmits “Feedback message” to the “Feedback server” for storage. Upon completion of the feedback storage, the success message is returned to the application from the “Feedback server”, which subsequently relays them to the “User” object for display.

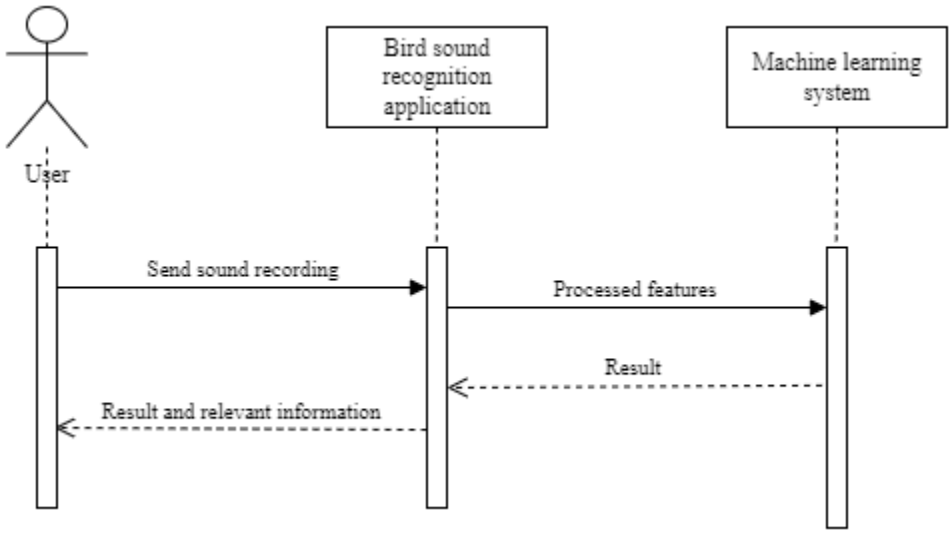


Figure 3.7: Sequence diagram of bird sound recognition in the application

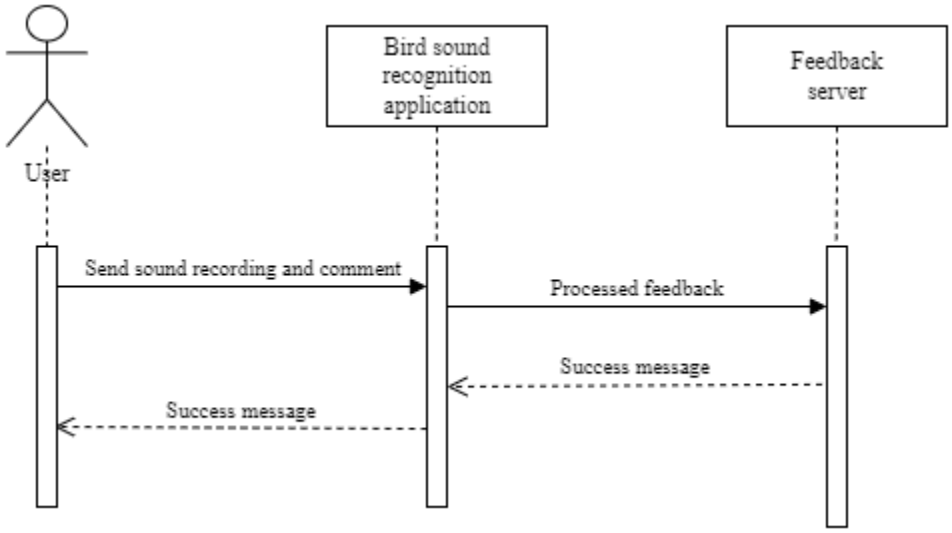


Figure 3.8: Sequence diagram of sending feedback on the corrected label of the sound in the application

3.7.2.1.4 State diagram

A state diagram visually depicts the different conditions and transitions that components or objects within an application can undergo throughout its operation. The application of state diagrams facilitates the modelling of the dynamic behaviour exhibited by an application, including

the events that initiate transitions among different states.

As demonstrated in Figure 3.9, the bird sound recognition application involves various states, each with specific triggering events. Initially, the sound file is input into the application by the user. If the user chooses to classify the sound, the sound file undergoes various transformations, transitioning through sequences of events that process the sound file and then extract features from the pre-processed sound file. These features are then transferred to a machine learning model that employs classification techniques to analyse the sounds based on the features extracted. Subsequent to the classification process, the machine learning model will export its result, which will then be combined with relevant information for display.

In the event of the user electing to transmit feedback on the corrected label of the sound, they will be prompted to input the relevant feedback. Consequently, the feedback and sound file will be transmitted to the server for storage.

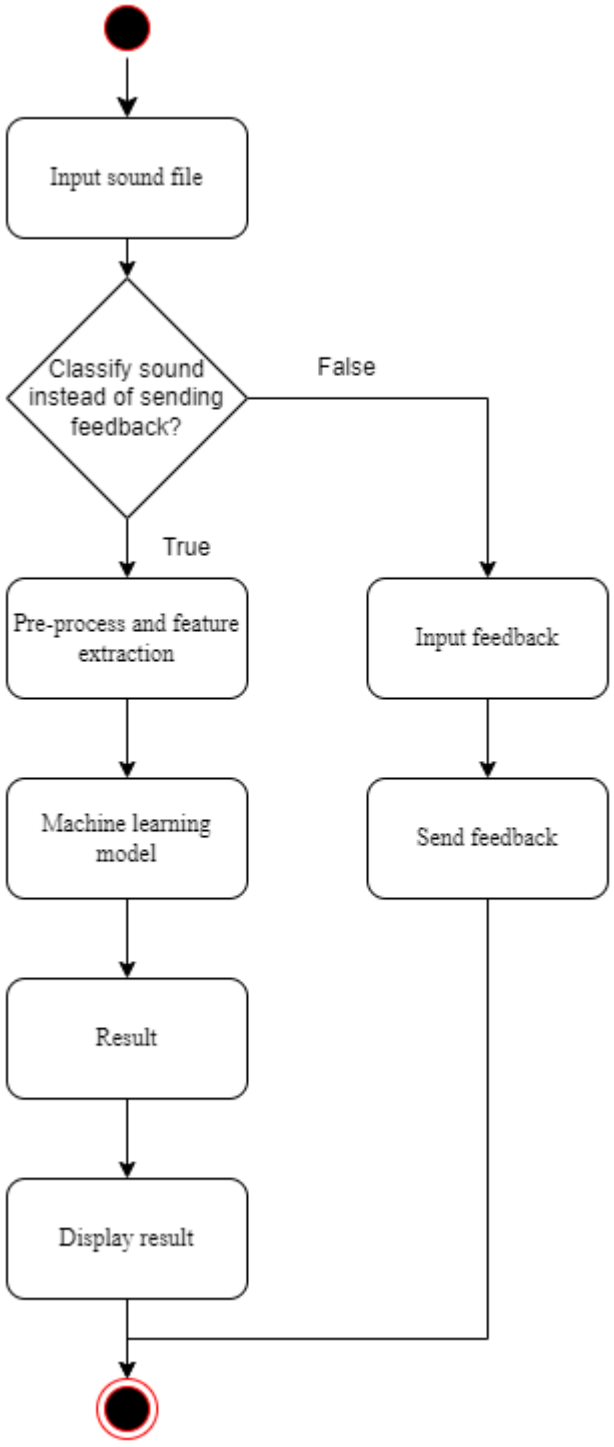


Figure 3.9: State diagram of bird sound recognition application

3.7.2.1.5 User interface

The application's user interface design is critical for ensuring ease of use and intuitive operation. A mock-up diagram was developed before implementation to provide a clear visualization of the interface's functionality and design direction. This focus on user-friendliness is essential for two reasons: firstly, it allows user feedback to concentrate on the application's core functionality and effectiveness rather than on usability issues; secondly, it significantly contributes to the application's success and widespread adoption, particularly among users who may have limited experience with computers or bird-watching. By prioritizing an accessible interface, the application aims to cater to a broad user base, including both novices and experts in the field.

As illustrated in Figure 3.10, the application's user interface will consist of three main screens: one for recording bird sounds, one for displaying the results of the classification of these sounds, and one for showing the menu of the application. The first interface will contain a button with “Record Now” text underneath that users can press to identify the bird sound. Pressing this button will start the application's recording of the target sound, which can be pressed as long as required. Once the recording is sufficient, the user may release the "Record Now" button to initiate the pre-processing and classification of the sound.

Once these processes are complete, the user will be presented with the classification results of the sound. If the sound has been successfully classified as belonging to a specific bird species, the picture of the species and its name of the bird will be shown to the user. The user may choose the “feedback” button to send feedback in regards to the classification of the recording, or the left-arrow button on the top left to return to the main interface.

The screen for recording bird sounds, displaying results, browsing past recordings, and submitting contains a menu button in the lower central area. This initiates the navigation of the user to the application menu screen. The menu screen directs the user to the sound recording screen, the past sound recordings, or the exit of the application. The lower central area of the application screens where the left-arrow button is located can be pressed to direct the user back to the screen that was visited prior to the pressing of the menu button.

The “Past Records” screen provides the user with a list of past recordings made using the application. The user may opt to reclassify the sound by pressing the “Identify again” button, or alternatively submit feedback on the sound recording by pressing the “Feedback” button.

The “Feedback” screen facilitates the submission of feedback on the classification of the recording. The screen is composed of three rows of fields, of which the last two rows are interactive. The first row contains the name of the audio file, while the second row comprises a selection field in which the user can select the correct bird species for the audio. The third row is a text field in which the user can enter their comment. Upon completion of the aforementioned fields, the user is prompted to press the “Send” button to submit the feedback.

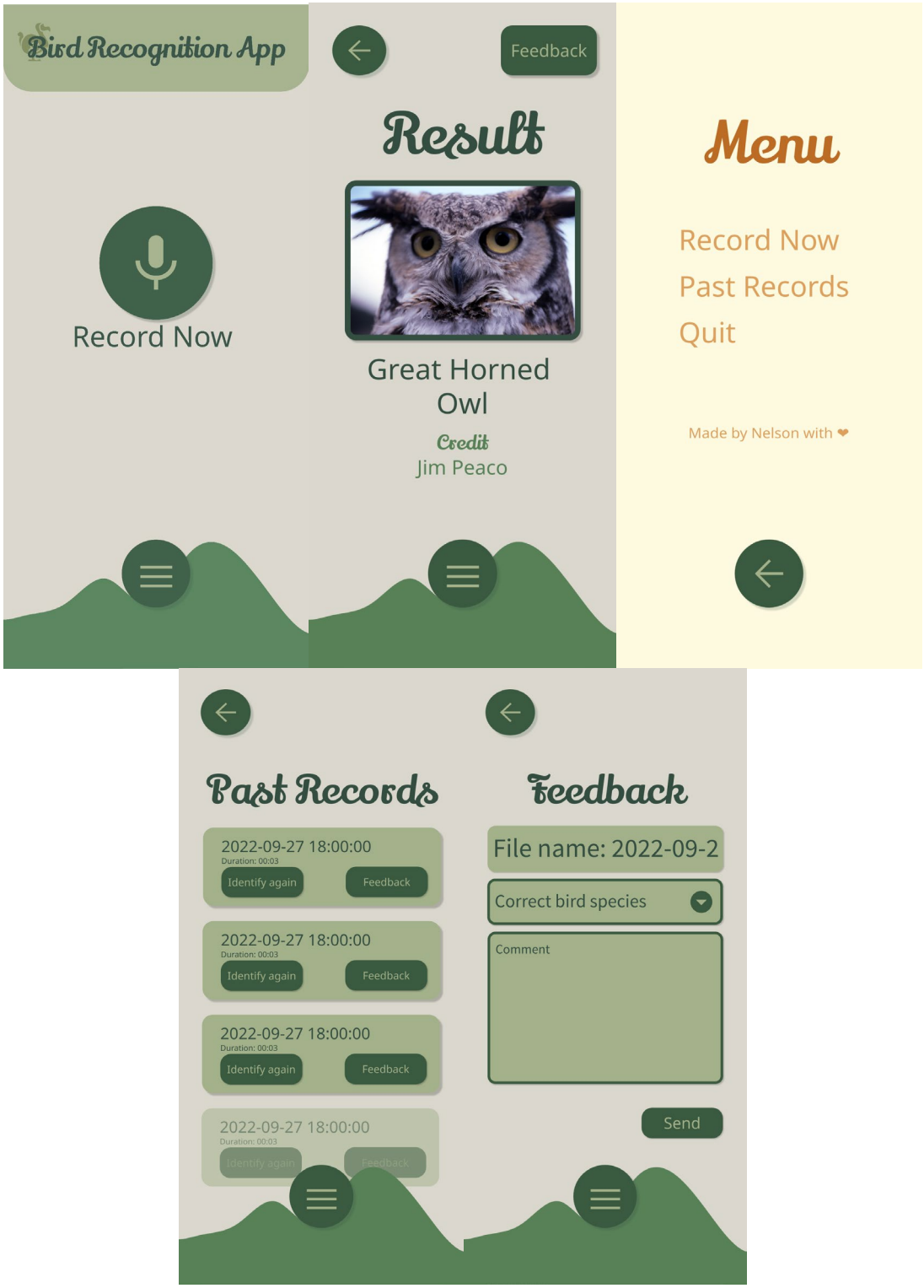


Figure 3.10: Mock-up of the user interface of bird sound recognition application

3.7.2.2 Build prototype

Following the approval of the quick design by the supervisor, a prototype of the application will be created. This prototype will be built upon the existing prototype and incorporate the latest iteration of the quick design. The prototype will then be given to the supervisor for further evaluation, after which the prototyping cycle will be repeated if deemed necessary.

3.7.2.3 Testing

The application will be subjected to a rigorous testing process, encompassing a variety of functionality test cases, with the objective of ensuring the accurate implementation of its features. Feedback from the supervisor will also be taken into consideration to ensure that the requirement was met. Should further changes or improvements be required, the requirements of the application will be subjected to further refinement. Conversely, if the latest iteration of the application is deemed to be ready to be transformed into the project's final product, the final product phase will commence.

3.7.2.4 Refine requirements

The application's requirements will be subject to refinement in accordance with the evaluation provided by the supervisor. This process is intended to ensure that the additional features required during the quick design phase are not overlooked. Following refinement, the requirements will serve as the guidelines during the subsequent quick design phase.

3.7.3 Phase 3 - Final product

The subsequent phase of the development cycle will be developing the final product of the application. The objective of this phase is to finalise the design of the user interface, as this is the aspect that users will interact with directly. This approach ensures that the evaluation of the user interface is not negatively affected by its visual appearance, but rather focuses on the reliability of the application in identifying bird sounds. To assess the application's compatibility with conventional devices, it will be tested on multiple Android devices with different specifications.

3.7.4 Phase 4 - User evaluation

The application package will be distributed to individuals using a simple random sampling method. Once the users have completed testing the application, they will be provided with a link to complete a survey via the Google Forms platform. This method was selected as it offers the convenience of consolidating user evaluations for carrying out the analysis in a single location. Inspired by the System Usability Scale framework as shown in Brooke's research (1996), the survey will consist of structured questions employing a Likert scale (1-5) to quantify user satisfaction. As illustrated in Figure 3.11, the figure presents a series of questions derived from Appendix A. The complete questionnaire can be located in Appendix A.

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

Figure 3.11: Questionnaire derived from Appendix A

3.8 Step 5 - Documentation

Once the previous methodological stages have been completed, the documentation of the project will begin. During this stage a detailed thesis report, research paper and presentation slides will be produced. The thesis report will provide a comprehensive overview of the project, including but not limited to the background, motivation, problem statements, objectives, methods used, results, conclusions and future work of the project. The research paper will summarise the contributions and findings of the project in a format suitable for publication in a research journal. The presentation slides will be a condensed version of the project report, highlighting key points that are easily digestible for a wider audience.

3.9 Hardware and software requirement

The hardware requirements for the proposed development and testing:

Table 3.3: Hardware requirement for the proposed development and testing

No.	Hardware	Specification
1.	Accelerated processing unit (APU)	i7-11370H
2.	Random access memory (RAM)	32GB
3.	Storage	400MB

4.	External Graphical Processing Unit	RTX 3600 12 GB
5.	Mobile devices	Samsung Note 9 (Exynos); Samsung S21 5G; Google Pixel 8a

The software requirements for the proposed development and testing:

Table 3.4: Software requirement for the proposed development and testing

No.	**Software**	**Specification**
1.	Development operating system	Linux Mint 21.3
2.	Testing operating system	Android 12, 14, 15
3.	Integrated development environment (IDE)	Microsoft Visual Studio Code
4.	Front-end	React Native
5.	Back-end (Pre-processing and classification engine)	Python

3.10 Summary

This chapter discussed the project methodology, which consisted of several crucial stages for developing a machine learning algorithm for bird sound classification and its associated application. The process began with a comprehensive literature review, exploring existing research in the field. Following this, the data preparation phase involved collecting, processing, and augmenting relevant data. The project then progressed to evaluating three distinct machine learning models for bird sound classification. Subsequently, the application development phase focused on prototyping the bird sound classification tool. The final documentation phase encompassed the creation of a thesis report, a research paper, and presentation materials to showcase the project's outcomes and potential future directions.

CHAPTER 4: IMPLEMENTATION

4.1 Introduction

The chapter will provide a thorough overview of the implementation and use of machine learning techniques for the classification of avian audio, as well as an examination of the Bird Sound Recognition Application in detail. This chapter provides a comprehensive overview of the steps and tools employed in the development of the machine learning algorithms using Python, and the mobile application using React Native with Expo framework.

4.2 Integrated development environment (IDE)

The integrated development environment (IDE) employed for this project to develop the machine learning models and mobile application is Microsoft Visual Studio Code (VS Code) (Microsoft, 2022). The software is characterised by its simplicity, yet it is also powerful and flexible to the user's needs. Its capacity to install extensions enhances the functionality of the editors and supports a variety of programming languages. The installation of VS Code is facilitated by the APT package manager. For the purposes of this project, the non-Flatpak version has been selected and installed on the personal laptop. Following the installation process, the requisite extensions for the Python programming language and React Native were installed, thereby enhancing the effectiveness of the development process. The IDE is configured with settings to the project and developer preference, ensuring the development process can be carried out smoothly, including the debugging preferences and code formatting options. A screenshot of the project files and extensions interface opened in the VS Code is shown in Figure 4.1 and 4.2 respectively.

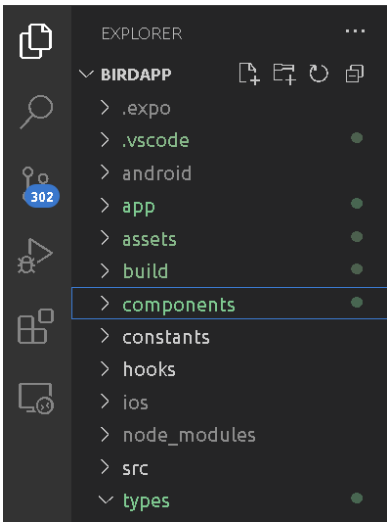


Figure 4.1: Screenshot of the project files interface in VS Code

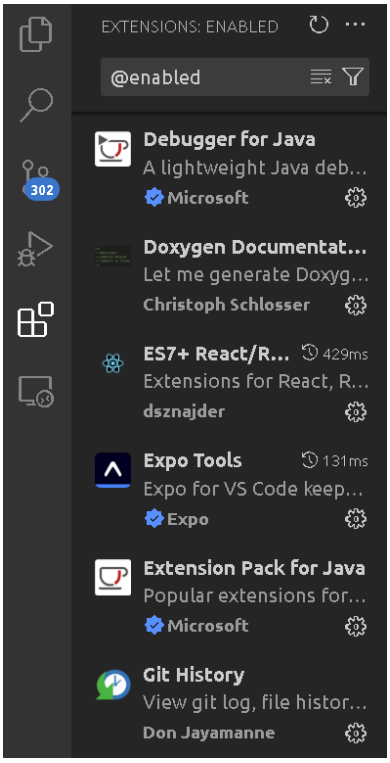


Figure 4.2: Screenshot of the extensions installed in VS Code

4.3 Implementation of machine learning algorithms

The subsequent subsections will provide a detailed discussion of the steps and tools employed in the development of machine learning algorithms.

4.3.1 Python

The programming language used for the implementation of machine learning algorithms is Python. The Linux Mint operating system comes with the 3.10 version of Python pre-installed. To ensure that the Python runtime version and packages used in the development of machine learning algorithms will not be interfered with by the system environment, a virtual environment of Python is set up through the command terminal. Figure 4.3 illustrates the setup of Python Virtual Environment.

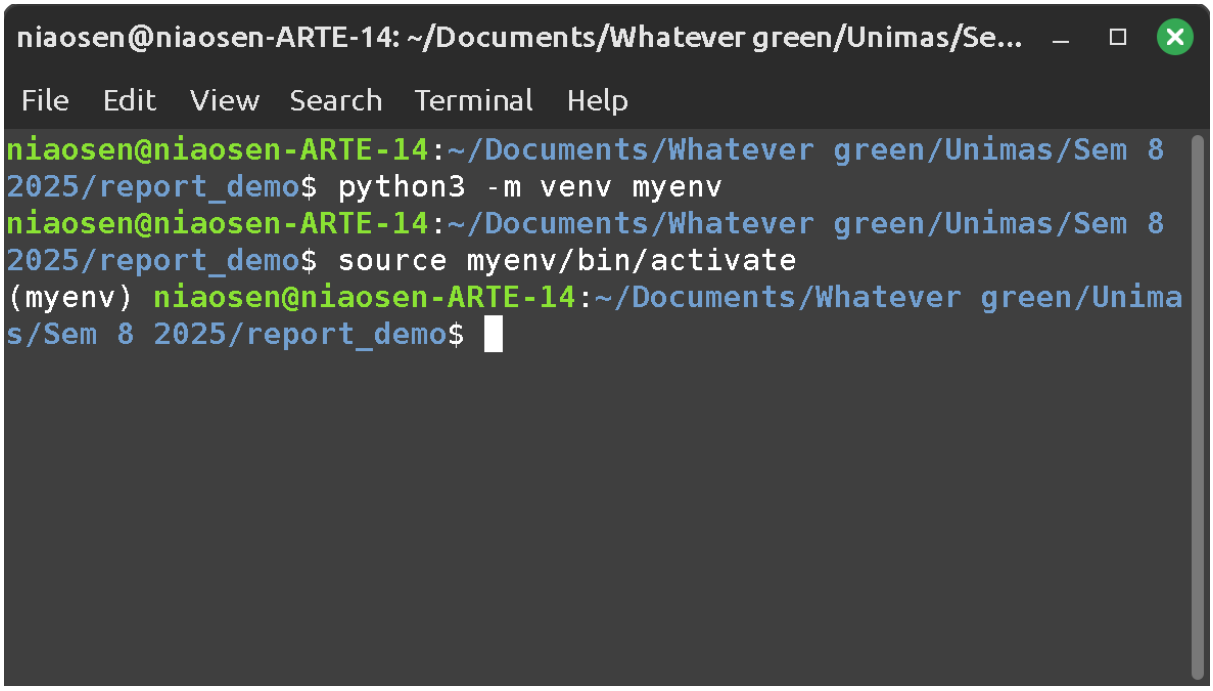
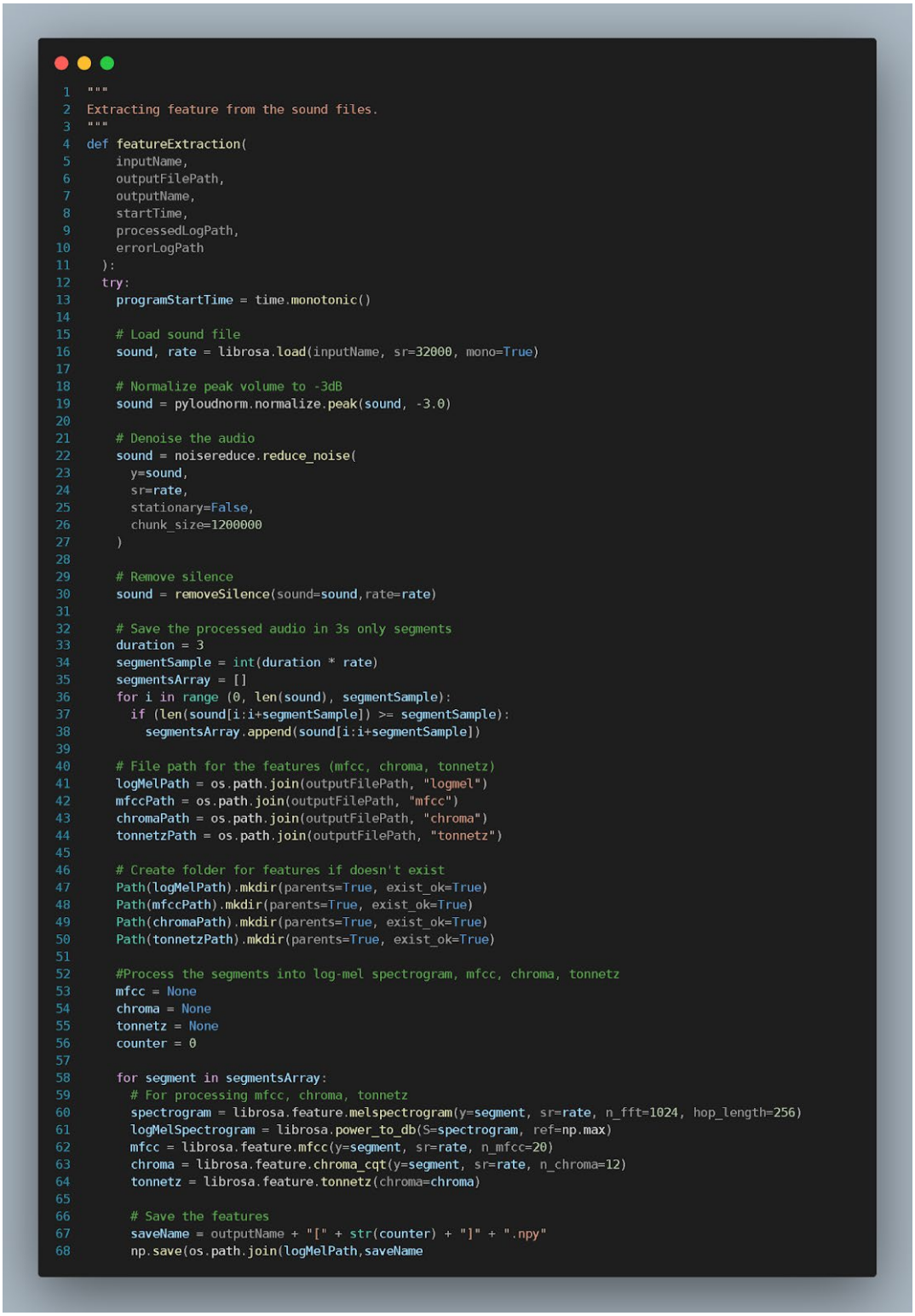


Figure 4.3: Setting up and using the Python Virtual Environment

4.3.2 Pre-processing

The pre-processing of the audio files is done in a similar way for the three machine learning techniques: Naive Bayes, Support Vector Machines and Convolution Neural Networks. Firstly, the audio files are loaded using the load function provided by the “librosa” library. The peak volume of the audio files is normalised using the “pyloudnorm” library and then denoised using

the “noisereduce” library. The silent portions of the audio are then removed and the remaining audio is segmented into 3-second clips. Log-mel spectrogram, MFCC, chroma and tonnetz features are then extracted from these segments using the “librosa” library. These features are stored on disk as NumPy array files for use in machine learning classification at a later time. As illustrated in Figure 4.4, the Python script was employed for the preparation of the data for machine learning classification.



```
"""
Extracting feature from the sound files.
"""
def featureExtraction(
    inputName,
    outputFilePath,
    outputName,
    startTime,
    processedLogPath,
    errorLogPath
  ):
  try:
    programStartTime = time.monotonic()

    # Load sound file
    sound, rate = librosa.load(inputName, sr=32000, mono=True)

    # Normalize peak volume to -3dB
    sound = pyloudnorm.normalize.peak(sound, -3.0)

    # Denoise the audio
    sound = noisereduce.reduce_noise(
      y=sound,
      sr=rate,
      stationary=False,
      chunk_size=1200000
    )

    # Remove silence
    sound = removeSilence(sound=sound,rate=rate)

    # Save the processed audio in 3s only segments
    duration = 3
    segmentSample = int(duration * rate)
    segmentsArray = []
    for i in range (0, len(sound), segmentSample):
      if (len(sound[i:i+segmentSample]) >= segmentSample):
        segmentsArray.append(sound[i:i+segmentSample])

    # File path for the features (mfcc, chroma, tonnetz)
    logMelPath = os.path.join(outputFilePath, "logmel")
    mfccPath = os.path.join(outputFilePath, "mfcc")
    chromaPath = os.path.join(outputFilePath, "chroma")
    tonnetzPath = os.path.join(outputFilePath, "tonnetz")

    # Create folder for features if doesn't exist
    Path(logMelPath).mkdir(parents=True, exist_ok=True)
    Path(mfccPath).mkdir(parents=True, exist_ok=True)
    Path(chromaPath).mkdir(parents=True, exist_ok=True)
    Path(tonnetzPath).mkdir(parents=True, exist_ok=True)

    #Process the segments into log-mel spectrogram, mfcc, chroma, tonnetz
    mfcc = None
    chroma = None
    tonnetz = None
    counter = 0

    for segment in segmentsArray:
      # For processing mfcc, chroma, tonnetz
      spectrogram = librosa.feature.melspectrogram(y=segment, sr=rate, n_fft=1024, hop_length=256)
      logMelSpectrogram = librosa.power_to_db(S=spectrogram, ref=np.max)
      mfcc = librosa.feature.mfcc(y=segment, sr=rate, n_mfcc=20)
      chroma = librosa.feature.chroma_cqt(y=segment, sr=rate, n_chroma=12)
      tonnetz = librosa.feature.tonnetz(chroma=chroma)

      # Save the features
      saveName = outputName + "[" + str(counter) + "]" + ".npy"
      np.save(os.path.join(logMelPath,saveName
```

Figure 4.4: Screenshot of the pre-processing script

4.3.3 Machine learning algorithms

The implementation of Naive Bayes, SVM and CNN is facilitated through their respective Python library classes. These classes and libraries are prepared for immediate utilisation and are accompanied by comprehensive documentation to streamline the implementation process. It is imperative to note that each of the models requires slightly different pre-processing due to their inherent requirements, such as Naive Bayes and SVM, and for the purpose of optimising model performance.

The Naive Bayes model is founded on the premise of the complement Naive Bayes classifier, a particularly appropriate choice for imbalanced datasets that is found in this study. The utilisation of the "ComplementNB" class from the "Sklearn" Python library facilitates this process. As Naive Bayes requires non-negative values for classification, the features will undergo transformation using their quantiles information through the "QuantileTransformer" from "Sklearn" before being fitted to the classifier.

For the SVM, the implementation utilises the "SVC" class from the "Sklearn" Python library. Given the model's sensitivity to the feature scales, the features will be standardized through a process of mean removal and scaling to unit variance. This is implemented using the "StandardScaler" from "Sklearn".

The CNN model is implemented through the utilisation of the classes and functions available within the Pytorch library. While the model does not require any additional pre-processing to function effectively, it has been observed that the model can achieve superior performance when the features are standardized using the "StandardScaler" function from the "Sklearn" package.

Figure 4.5, Figure 4.6 and Figure 4.7 illustrate the implementation of Naive Bayes, SVM and CNN models respectively.

```python
# Classify with Naive Bayes
def nb_classify(X_train, y_train, X_test):
  # Min max scaling to remove negative values
  scaler = QuantileTransformer(subsample=None)
  X_train = scaler.fit_transform(X_train)
  X_test = scaler.transform(X_test)

  # Train model
  nb = ComplementNB()
  nb.fit(X_train, y_train)

  # Evaluate
  return nb.predict(X_train), nb.predict(X_test)
```

Figure 4.5: Naive Bayes model implementation

```python
# Classify with SVM
def svm_classify(Xtrain, yTrain, Xtest):
  # Create model
  svm = SVC(kernel='rbf',
            max_iter=-1,
            random_state=42,
            class_weight='balanced')

  # Train model
  svm.fit(Xtrain, yTrain)

  # Evaluate
  return svm.predict(Xtrain), svm.predict(Xtest)
```

Figure 4.6: SVM model implementation

```python
class BirdSoundCNN(nn.Module):
    def __init__(self, input_dim, num_classes):...

    def forward(self, x):
        # Add channel dimension (batch_size, 1, input_dim)
        x = x.unsqueeze(1)

        # Extract features through conv layers
        features = self.conv_block(x)

        # Apply attention
        attention_weights = self.attention(features)
        features = features * attention_weights

        # Classification
        return self.fc_block(features)
```

Figure 4.7: CNN model implementation

4.4 Mobile application

The subsequent sub-sections will discuss the steps and tools employed to develop the bird sound mobile application. The recognition of bird audio was achieved through the utilisation of a hybrid mode, wherein the pre-processing of audio was executed on a server, while the classification employing machine learning models was conducted on a mobile device.

4.4.1 React Native with Expo framework

The environment setup for the React Native with Expo framework was accomplished by following the installation process provided in the official React Native documentation (Meta Open Source, 2025). The setup of the framework is initiated by the installation of the Node.js v22.15.0, which then accompanied by the installation of the open-source version of Java Software Standard Edition Development Kit (JDK) version 17, OpenSDK, through the Advanced Package Tool (APT) package manager. After that, Android Studio is downloaded, and during the installation wizard of Android Studio, Android SDK for Android 15, Android SDK Platform 35, and Android

Virtual Device are selected to be installed. Once the Android SDK is installed, the ANDROID_HOME environment variable as shown in Figure 4.8 is exported to "/etc/profile" to ensure further command terminal instances can access the newly added environment variables.

Figure 4.8: ANDROID_HOME environment variable

The Watchman tool is installed through the APT package manager. Then, to create a base application project environment, the command "npx create-expo-app@latest" is run on the terminal at the preferred directory. To connect and test the app on the android device, the setting "debugging over USB" on the mobile device is enabled, connect the device to the laptop, then run "adb devices" to check if the device is connected to the Android Debug Bridge on the laptop. To run the app, go to the root of the project, and type in "npx expo run:android" in the command terminal. The following figures illustrate the steps taken.

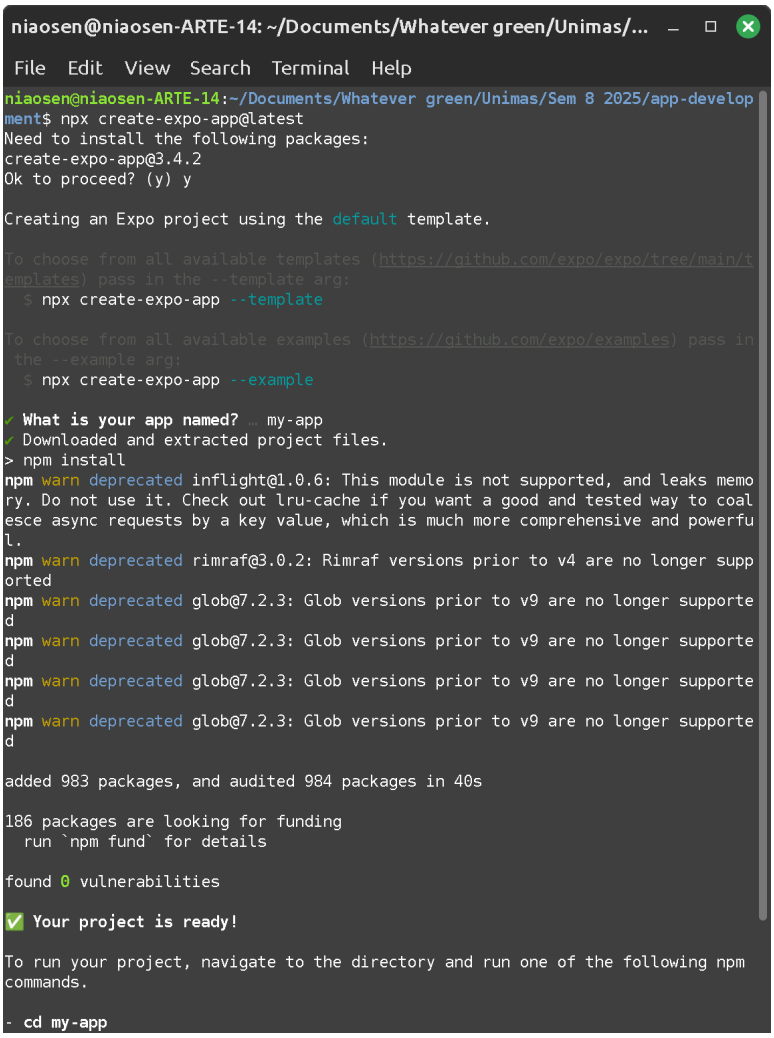


Figure 4.9: Creating project environment for mobile application

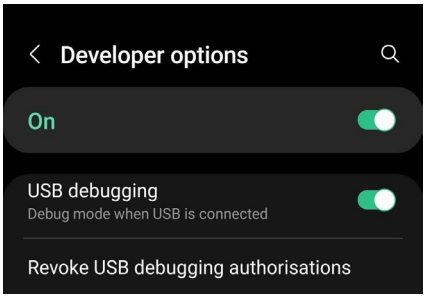


Figure 4.10: Enabling USB debugging option in the mobile device

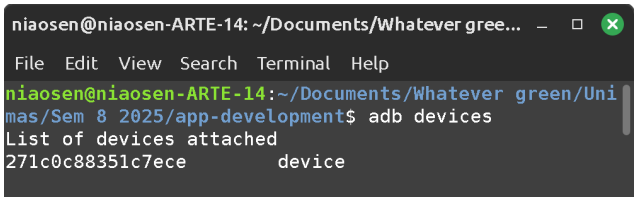


Figure 4.11: Checking if the mobile device is connected to the laptop

This configuration of the framework is crucial for the development and testing of the mobile application. With React Native and Expo framework, the mobile application that integrates the process of recording bird sound, feature extraction from the recordings, classify the bird sound using machine learning technique, and show the result to the user can be made possible.

4.4.2 Graphical User Interface (GUI) system

The mobile application incorporates a bird sound recognition system that employs machine learning techniques to classify audio recordings captured via the recording sound feature. The application under discussion was created using the React Native with Expo framework. This enables the user to record bird sounds and view the result of the bird sound recognition system. In order to initiate the recognition feature, the user may opt to record the sound on the primary interface or select a previously recorded audio file from the “past records” screen. The entirety of the processes necessary for the classification of audio are executed on the device itself, thereby enabling users to categorise avian sounds irrespective of the status of their internet connection. Figure 4.12, Figure 4.13, and Figure 4.14 showcases the interface of the application.

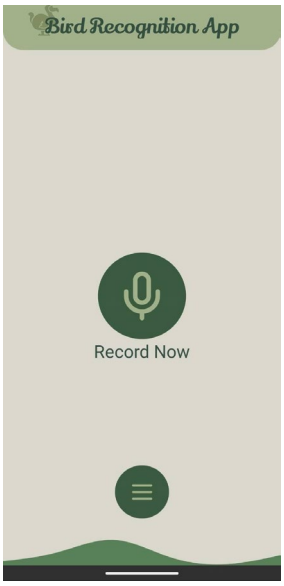


Figure 4.12: Screenshot of the main interface of the application

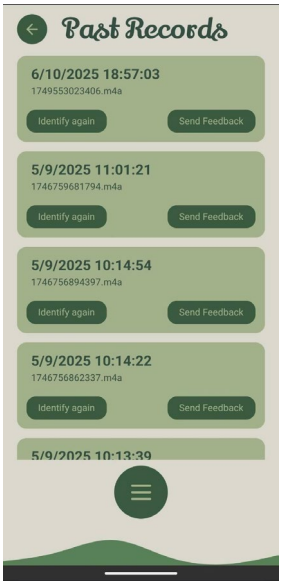


Figure 4.13: Screenshot of the list of past audio recordings

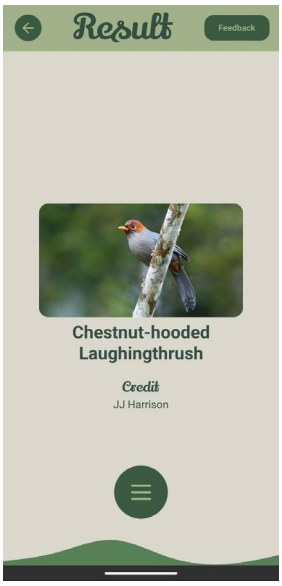


Figure 4.14: Screenshot of the result interface

4.4.3 Home page component of the application

The home page component constitutes the primary component of the application, thereby serving as the conduit between the graphical user interface and the underlying machine learning classification algorithm. In the event that the user elects to record new audio, the home page component will initially invoke the method of recording audio from the "expo-audio" library. Upon completion of the recording, the methods contained within the "expo-file-system" library are invoked to facilitate the transfer of the audio file from the cache directory to the internal directory. This process guarantees that the file is stored in a more permanent location, where users can subsequently perform additional actions with the file. Upon completion of the file transfer, the "classifyAudio" method is invoked to execute the requisite pre-processing and classification operations. Subsequently, the result page is presented, displaying the classification outcome of the bird species determined by the designated machine learning algorithm.

```
// Audio recording
const [isRecording, setRecording] = useState(false);
const audioRecorder = useAudioRecorder(recordingOptions);
const toggleRecording = async() => {
  if (!isRecording) {
    // Ask for permission
    var audioPermission = await AudioModule.requestRecordingPermissionsAsync();
    if (audioPermission) {
      await ensureDirExists();
      await audioRecorder.prepareToRecordAsync();
      setRecording(true);
      audioRecorder.record();
    } else {
      Alert.alert(
        'Permission denied',
        'Permission to access microphone was denied. Please allow the app to access the microphone',
        [{text:'OK'}]
      );
    }
  } else {
    await audioRecorder.stop();
    setRecording(false);

    // Get the URI after stopping
    const audioURI = audioRecorder.uri;
    if (audioURI) {
      const uri = FileSystem.cacheDirectory + "Audio/" + GetFilenameFromURI(audioURI)

      const fileName = `${Date.now()}.wav`;
      const destination = audioDir + fileName;

      await FileSystem.moveAsync({
        from: uri,
        to: destination
      })

      // Send the audio recording file for classification
      const encodedLabel = await ClassifyAudio.classifyAudio(destination);
      // Send the user to result screen
      navigation.navigate('resultPage', {encodedLabel: encodedLabel, fileName: fileName});

    } else {
      Alert.alert(
        'Error saving audio file',
        'There\'s an error in saving audio file, please try again. If the problem presists, contact the developer for help.'
      )
    }

  }
}
```

Figure 4.15: Snippet of the home page component

4.4.4 Pre-processing audio

Prior to the implementation of a machine learning technique for classification, it is imperative to ensure that the audio recording is pre-processed in a manner consistent with the training datasets. The implementation is done by having the server that handles the pre-processing, before sending it back to the device for classification. This approach is instrumental in preserving the efficacy of the machine learning algorithm. The server is deployed using the “Fastapi” and “Uvicorn” python library on a local machine, and connected through a proxy using Localtunnel (Shtylman, 2022). The audio would undergo a series of pre-processing steps, including

normalisation, denoising, silence removal, and segmentation. Figure 4.16 illustrates a server configured with several application programming interfaces (APIs).

```python
from fastapi import FastAPI, UploadFile, File, HTTPException
from fastapi.middleware.cors import CORSMiddleware
import tempfile
import os
from feature_extractor import extract_features

app = FastAPI(title="Bird Recognition Feature Extraction API")

# Enable CORS for mobile app
app.add_middleware(
  CORSMiddleware,
  allow_origins=["*"],  # Configure this properly for production
  allow_credentials=True,
  allow_methods=["*"],
  allow_headers=["*"],
)

@app.post("/extract-features")
async def extract_audio_features(audio: UploadFile = File(...)):…

@app.get("/health")
async def health_check():…

if __name__ == "__main__":
  import uvicorn
  uvicorn.run(app, host="0.0.0.0", port=os.getenv("PORT", default=8000))
```

Figure 4.16: Pre-processing server

4.4.5 Classification of the audio using machine learning technique

The fine-tuned model of the chosen machine learning algorithm will be exported as Open Neural Network Exchange (ONNX) files using the respective ONNX library (Open Neural Network Exchange, 2024). This facilitates the consolidation of models into the mobile application during the build process. The classification of the audio will be conducted within the designated "AudioClassifier" class. This class incorporates the requisite methods to transform the pre-processed audio file into the appropriate array shape prior to its submission to the models for

classification. The result of the classification process will be the encoded label of the bird species class. This facilitates the execution of subsequent actions on the encoded label. For instance, it enables a cross-reference with a JSON file, which in turn provides further details concerning the bird species.

Figure 4.17: The collapsed functions of the AudioClassifier class

4.4.6 Deployment of mobile application and server

The following sub-sections will discuss the deployment process of the mobile application and the server.

4.4.6.1 Mobile application

In order to create an Android Package (APK) that can be installed independently on the device, it is first necessary to install the Package Expo Application Services Command Line Interface (EAS CLI) through the Node Package Manager (NPM) via the command terminal.

Subsequently, an Expo account is created via the EAS CLI, which then can be accessed for the functionalities of EAS CLI. In order to facilitate the local construction of the Android application in APK format, it is necessary to adjust the configurations in the "eas.json" file. This modification will streamline the installation process on mobile devices. Upon completion, the following command should be entered in the command terminal: “npx eas build --local”. The APK file will be created in the project directory. Figure 4.18, Figure 4.19, and Figure 4.20 illustrated the steps taken to create the mobile application for ease of installation.

Figure 4.18: Command for the installation of the EAS CLI

Figure 4.19: Command for logging into EAS CLI

Figure 4.20: The highlighted configuration in the "eas.json" file that is required for local construction of the APK file

4.4.6.2 Server

A Python virtual environment was configured for the server. This procedure is designed to prevent any disruption to the server components that may occur as a result of system activity. A command terminal on the local machine will be responsible for executing the server script, while an additional terminal will facilitate communication with the devices via a proxy, thereby establishing a connection between the devices and the local machine through the domain. The selection of this deployment method is driven by its capacity to facilitate the expeditious development and demonstration of application capabilities.

The proxy service utilised for this project was entitled "Zrok". The application can be downloaded via the official download page: https://docs.zrok.io/docs/guides/install. In order to utilise the proxy server offered by https://myzrok.io, it is necessary to create an account on the aforementioned website. Following the successful download of the Zrok binary, the command terminal must be opened in the same directory as the binary. The commands for enabling the Zrok

for the environment and reserving the public share token will then need to be entered. Following the configuration of the requisite settings, the execution of the proxy is to be initiated concurrently with the server script. Figure 4.21, Figure 4.22 and Figure 4.23 illustrated the steps taken to develop the server.

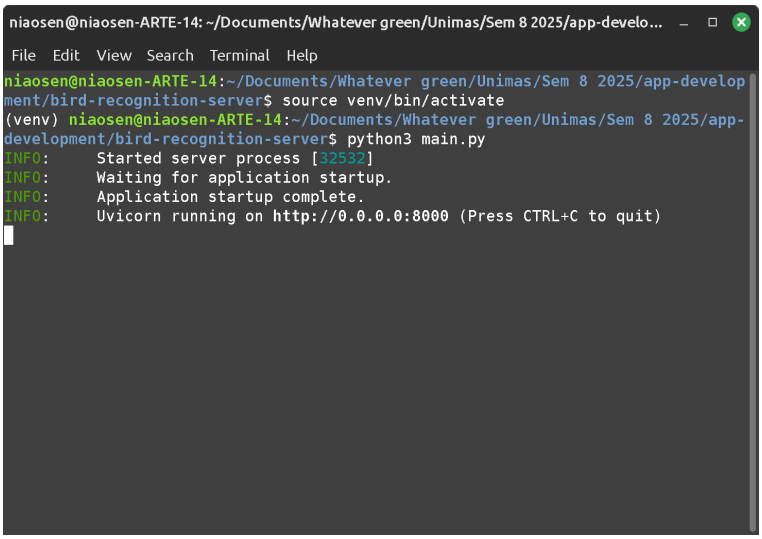


Figure 4.21: Starting server script

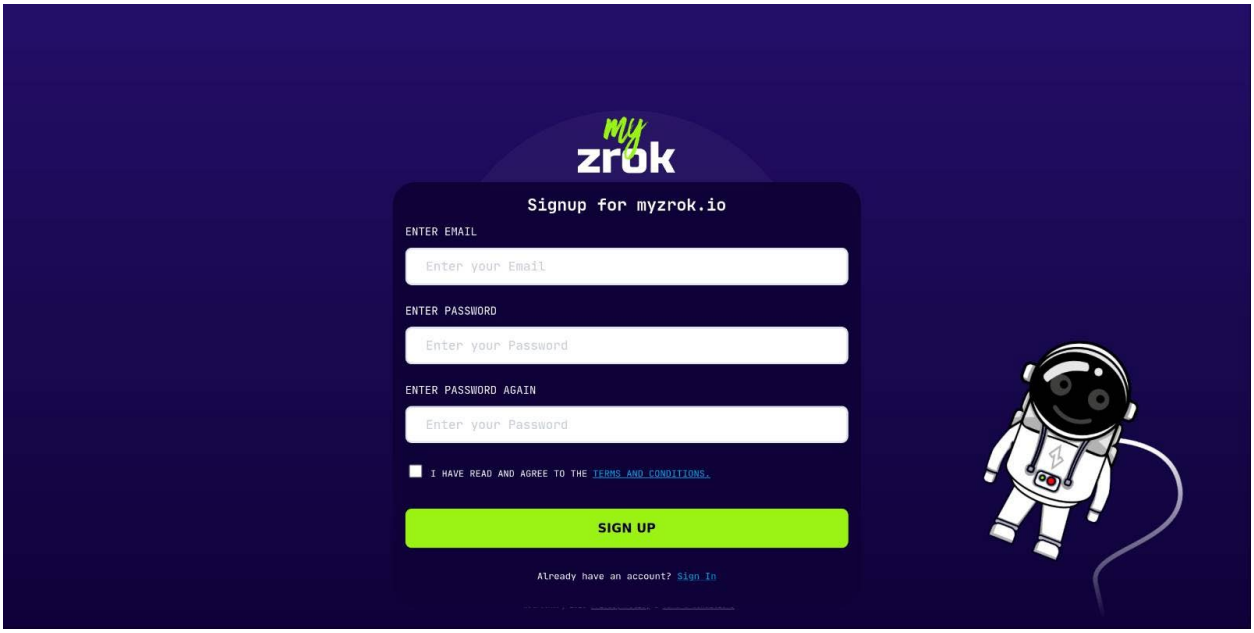


Figure 4.22: Creating account on https://myzrok.io

```
zrok enable <your_account_token>

Reserve with the Command Line

zrok reserve public 8000

zrok share reserved <reserved token>
```

Figure 4.23: Commands for setting up and running the proxy service

4.5 Summary

This chapter discussed the steps involved in implementing the bird sound classification system into a mobile application. The application was developed using React Native with Expo framework, and the machine learning model using ONNX. The graphical interface of the application would facilitate the recording and classification of avian sounds according to the user's preferences. The implementation was conducted in a moderate environment, utilising a personal laptop. The application was successfully exported as an APK file, allowing for straightforward installation on mobile devices. It is evident that the implementation of the bird sound classification was successful in its objective. This provides the capability to demonstrate the use of machine learning techniques in the domain of avian audio through the user's devices, as opposed to the development environment.

CHAPTER 5: RESULTS

5.1 Introduction

This chapter will provide an evaluation of testing and results of the machine learning techniques used for bird sound recognition and the developed application. The chapter commences with an introduction to the significance of evaluating and testing machine learning techniques in identifying bird species through their acoustic signatures.

5.2 Evaluation

A thorough evaluation was conducted to assess the effectiveness and efficiency of the proposed machine learning techniques for the classification of bird sounds. The evaluation process concentrated on four primary metrics: classification accuracy, precision, recall, and the F1-score. Additionally, the time efficiency of the system was taken into consideration. The evaluation phase employed standard machine learning performance metrics to assess the effectiveness and performance of the bird sound recognition models. Moreover, the application's compatibility, functionality and usability were evaluated through device tests, unit tests and the System Usability Scale (SUS). This section will discuss the metrics of the methodologies employed.

5.3 Machine learning techniques evaluation

The evaluation of classification performance aimed to assess the effectiveness of the proposed machine learning techniques in accurately identifying bird species from their acoustic signatures.

5.3.1 Machine learning parameters tuning

Table 5.1 illustrate the best performing hyper parameter used for configuring the machine learning algorithms. These hyper parameters of Naive Bayes were chosen through random search, with the predefined space of values shown in Table 5.2. The hyper parameter of the CNN layers was done by changing the architecture or the number of layers of the neural network**.**

Table 5.1: Best-performing hyper parameters for machine learning algorithms

Technique	**Hyper parameters**
Naïve Bayes	"alpha" = 1
	"norm" = False
Support Vector Machine	"C" = 394.07472067394843
	"class_weight" = "balanced"
	"coef0" = 0.10770856880264157
	"decision_function_shape" = "ovo"
	"degree" = 2
	"gamma" = "auto"
	"kernel" = "poly"

	"max_iter" = -1
	"tol" = 0.0007730292273498736
Convolution Neural Network	Conv1d (Input=1, Output=64, Kernel=7, Padding=3)
	BatchNorm1d (Channels=64)
	ReLU
	Conv1d (Input=64, Output=64, Kernel=7, Padding=3)
	BatchNorm1d (Channels=64)
	ReLU
	MaxPool1d (Size=2)
	Dropout1d (Probability=0.2)
	Conv1d (Input=64, Output=128, Kernel=5, Padding=2)
	BatchNorm1d (Channels=128)
	ReLU
	Conv1d (Input=128, Output=128, Kernel=5, Padding=2)
	BatchNorm1d (Channels=128)
	ReLU

MaxPool1d (Size=2)

Dropout1d (Probability=0.2)

Conv1d (Input=128, Output=256, Kernel=3, Padding=1)

BatchNorm1d (Channels=256)

ReLU

Conv1d (Input=256, Output=256, Kernel=3, Padding=1)

BatchNorm1d (Channels=256)

ReLU

MaxPool1d (Size=2)

Dropout1d (Probability=0.2)

Conv1d (Input=256, Output=512, Kernel=3, Padding=1)

BatchNorm1d (Channels=512)

ReLU

Conv1d (Input=512, Output=512, Kernel=3, Padding=1)

BatchNorm1d (Channels=512)

ReLU

AdaptiveAvgPool1d (Size=1)

Flatten

Linear (Input=512, Output=128)

ReLU

Linear (Input=128, Output=512)

Sigmoid

Linear (Input=512, Output=512)

BatchNorm1d (Channels=512)

ReLU

Dropout (probability =0.5)

Linear (Input=512, Output=256)

BatchNorm1d (Channels=256)

ReLU

Dropout (Probability=0.5)

Linear (Input=256, Output=128)

BatchNorm1d (Channels=128)

	ReLU
	Dropout (Probability=0.5)
	Linear (Input=128, Output="Number of classes")

Table 5.2: Predefined values of hyper parameters to search using random search

Techniques	**Predefined values**
Naïve Bayes	"alpha": loguniform(1e-8, 1e1) "norm": [True, False]
Support Vector Machine	"C": loguniform(0.01, 1000) "kernel": ["rbf", "linear", "poly"] "gamma": ["scale", "auto] + list(loguniform(1e-6, 1e-1).rvs(10)) "degree": [2, 3, 4] "coef0": uniform(-1, 2) "class_weight": ["balanced", "none] "tol": loguniform(1e-5, 1e-2) "decision_function_shape": ["ovr", "ovo"]

5.3.2 Machine learning model performance

The performance evaluation was conducted using a 5-fold cross-validation approach to ensure robust assessment of the models' generalisation capabilities.

Table 5.3: Classification performance of testing and training datasets

Technique	Datasets	Metrics	Cross-validation fold				
			1	2	3	4	5
Naïve Bayes	Training	Accuracy	0.3004	0.2734	0.2951	0.2656	0.3174
		Recall	0.2028	0.1720	0.1989	0.1728	0.2043
		Precision	0.3542	0.4039	0.3769	0.3085	0.3675
		F1-score	0.1913	0.1598	0.1726	0.1437	0.1703
	Testing	Accuracy	0.0858	0.0703	0.0965	0.0684	0.1033
		Recall	0.0554	0.0635	0.0689	0.0718	0.0698
		Precision	0.0945	0.0675	0.0483	0.0696	0.0460
		F1-score	0.0412	0.0376	0.0374	0.0454	0.0427
Support Vector Machine	Training	Accuracy	1.0000	1.0000	1.0000	1.0000	1.0000
		Recall	1.0000	1.0000	1.0000	1.0000	1.0000
		Precision	1.0000	1.0000	1.0000	1.0000	1.0000
		F1-score	1.0000	1.0000	1.0000	1.0000	1.0000
	Testing	Accuracy	0.2923	0.2780	0.2694	0.2618	0.2720
		Recall	0.1884	0.2163	0.2357	0.2290	0.2107

		Precision	0.2051	0.2351	0.2749	0.2782	0.2399
		F1-score	0.1683	0.1979	0.2254	0.2069	0.1926
Convolution Neural Network	Training	Accuracy	0.9010	0.8295	0.8530	0.8773	0.8050
		Recall	0.9384	0.8706	0.8792	0.9205	0.8558
		Precision	0.8975	0.8422	0.8719	0.8687	0.7802
		F1-score	0.9106	0.8414	0.8669	0.8857	0.7962
	Testing	Accuracy	0.3550	0.3429	0.3969	0.3679	0.2831
		Recall	0.2618	0.3351	0.3723	0.3812	0.2775
		Precision	0.2527	0.3545	0.3481	0.3570	0.2446
		F1-score	0.2312	0.3012	0.3220	0.3215	0.2322

As illustrated in Table 5.3, the performance of machine learning algorithms is demonstrated on training and testing datasets for each cross-validation fold. In the context of a 5-fold cross-validation, the Naive Bayes technique exhibited suboptimal performance metrics across all five datasets, both during training and testing. In the context of 5-fold cross-validation, Support Vector Machines (SVMs) have demonstrated superior performance metrics in terms of classification on training datasets, while Convolutional Neural Networks (CNNs) have exhibited the highest performance metrics in classification on testing datasets.

Table 5.4: Overall mean of classification performance results using testing datasets

Technique	Accuracy	Recall	Precision	F1-Score
Naïve Bayes	0.0849	0.0652	0.0659	0.0409
Support Vector Machine	0.2720	0.2107	0.2399	0.1926
Convolutional Neural Network	0.3492	0.3256	0.3114	0.2816

Table 5.4 illustrates the mean of classification performance results of the machine learning algorithms.

Naive Bayes obtained the lowest classification metrics among the three. The model demonstrated an accuracy of 0.0849, which was higher than the baseline accuracy (0.0189) if chosen randomly from 53 bird species. The recall value of the Naive Bayes model was approximately 0.0652, indicating that approximately 6.52% of the testing samples identified by the model were deemed to be correct. The precision of Naive Bayes was approximately 0.0659. This indicated that 6.59% of testing samples from a given bird species were correctly identified by the Naive Bayes model. The F1-score was a balanced metric of recall and precision, with a value of 0.0409 for the Naive Bayes model.

SVM showed an improvement in performance metrics over Naive Bayes. Its accuracy was at 0.2720, showing that about 27.20% of testing samples were correctly identified by the model. Its recall and precision were sitting at 0.2107 and 0.2399 respectively, showing that its

identification of bird sounds was more reliable compared to Naive Bayes. The F1-score of the model was at 0.1926.

CNN exhibited the highest level of performance among the three machine learning techniques in all four metrics. The model demonstrated an accuracy of 0.3492, indicating that approximately 34.92% of the testing samples were correctly identified. The recall and precision values of 0.3256 and 0.3114, respectively, indicate that the identification made by the algorithm is more reliable than that produced by the other two machine learning techniques. The balance metric of recall and precision, F1 score, of the CNN model was at 0.2816.

5.3.3 Time efficiency evaluation

The objective of the time efficiency evaluation was to assess the speed and efficiency of the machine learning techniques employed for the classification of bird sounds, with a particular focus on the Naïve Bayes, SVM, and CNN models. The evaluation process was conducted by measuring the time required to train the model using the provided dataset. This will be useful for evaluating the time taken to retrain the model if new samples were to be received. The evaluation will also ensure that accurate classifications could be generated within timeframes that were both realistic and suitable for use in real-time mobile applications.

The evaluation was conducted by recording the epoch of the various timeframes, such as from the commencement to the conclusion of training time, and the start to the end of classifying test datasets. This facilitates precise analysis of the time required for the classification to determine whether further optimisation of the model, such as quantisation, is necessary for operation on mobile devices. The following table illustrates the mean training time and classification time of one-fold for the three machine learning techniques during the 5-fold cross validation.

Table 5.5: Average processing time for each machine learning technique

Technique	Training Time	Classification Time
Naïve Bayes	10 minutes 32 seconds	01 seconds
Support Vector Machine	84 minutes 00 seconds	01 seconds
Convolutional Neural Network	567 minutes 36 seconds	01 seconds

Table 5.5 illustrate the average processing time for each of the machine learning techniques. The training time of the Naive Bayes model was the shortest of the three, with an average of 10 minutes and 32 seconds per cross-validation fold. This finding indicates that the computational demands of Naive Bayes are comparatively modest in comparison to the other two machine learning techniques. The Support Vector Machine required an average of 84 minutes to complete a cross-validation fold. It was determined that the SVM model necessitated a greater computational capacity in order to successfully undertake its training. The training time for CNN is an average of 567 minutes and 36 seconds per cross-validation fold, which is the longest of the three techniques. It is evident that a substantial computational power will be necessary to facilitate the effective training of a CNN model.

The classification time for all three machine learning techniques was recorded at an average of one second per cross-validation fold. The discrepancy between the respective training and classification times indicates that a significantly lower computational power is required to complete the audio classification for these two models. The implementation of classification features with pre-trained models could be a viable solution, as the time required for classification was within a relatively acceptable timeframe.

It is imperative to acknowledge that these times were recorded by executing the training and testing of the models on the development environment referenced in Chapters 3 and 4. The temporal parameters of this process may be influenced by a number of factors, including the computational capabilities of the system employed to run the models and different datasets used to conduct training and testing. Nonetheless, the proportional difference in time between different machine learning techniques could serve as a useful reference point for future endeavours.

5.4 Mobile application evaluation

The subsequent subsections will address the evaluation of the mobile application developed using the machine learning solution. The evaluation of the mobile application encompassed a comprehensive testing process, which included assessing its compatibility with a range of devices, evaluating its functionalities, and conducting a usability assessment. It was imperative that each component of the mobile application be thoroughly examined, allowing for discussion in the subsequent sections.

5.4.1 Compatibility Testing

The mobile application was subjected to comprehensive testing on a range of Android devices, each with distinct specifications, to ascertain its compatibility and ensure consistent performance. A range of functionalities were evaluated, encompassing audio recording, real-time audio classification, user interface responsiveness, and the storage and retrieval of previous recordings. The following table illustrates the various mobile devices utilised for the purpose of device compatibility testing.

Table 5.6: Device compatibility testing

Device Model	Android Version	RAM (GB)	Performance Status
Samsung Note 9	12	6	Able to operate without issues
Samsung S21 5G	14	6	Able to operate without issues
Google Pixel 8a	15	8	Able to operate without issues

The devices that were the focus of the study varied in terms of their technical specifications, including the version of the Android operating system and the amount of random access memory (RAM). The device with the lowest specification that was tested was the Samsung Note 9, which had 6GB of RAM and was installed with Android Version 12. The Google Pixel 8a, which is installed with Android version 15 and 8GB of RAM, was the top-performing model. As demonstrated in the above table, the application functioned correctly on testing devices without encountering any unanticipated issues.

5.4.2 Functionality testing

In order to ascertain the reliability and functionality of the system, a comprehensive testing phase was conducted. The testing process entailed the isolation of each module, function, or method of the system, with the objective of ensuring that each component operated in accordance with its intended function and yielded the anticipated outcomes. This approach facilitated the identification and resolution of defects or bugs within specific components, thereby ensuring the stability and reliability of the application as a whole.

Table 5.7: Results of the tests conducted

Test Description	Expected Result	Actual Result	Pass/Fail
Audio recording functionality	Successful audio capture	Audio is able to capture on the app	Pass
Audio preprocessing	Able to upload and preprocess the audio	Able to upload and preprocess	Pass
Model classification	Able to create species prediction	Result page show appropriate species without error	Pass
File storage and retrieval	Successful show past recordings	Able to show all of the past recordings from storage	Pass
User Feedback	Able to send the email along with audio attachments	Able to send the email attached with audio recordings	
User interface navigation	Navigate through the screens without issues	Able to navigate the application with ease	Pass

As demonstrated in the Table 5.7, each feature of the application functioned as expected. It was imperative that the usability evaluation, which was to be conducted in the forthcoming subsection, not be adversely affected by the evident bugs that persisted in the application.

5.4.3 Usability evaluation

The usability evaluation is concerned with the evaluation of the application's usability and the user's preference for it. The application was designed to allow any user, regardless of their level of expertise in ornithology. A questionnaire was created using the System Usability Scale (SUS) framework (Brooke, 1996) and administered to a simple random sampling of users to represent the intended user group. The ratings collected from the survey allow for the assessment of the usability and preference of the application. The lowest rating is at 1, while the highest is at 5. A total of 25 respondents were collected for the study. However, only 23 were deemed valid following the removal of those who had answered ratings of 5 to all questions. The comprehensive responses provided by the respondents are documented in Appendix B.

5.4.3.1 System Usability Scale (SUS) responses

The survey was distributed via Google Form using simple random sampling. The following subsection will provide a detailed examination of the responses to each of the questions.

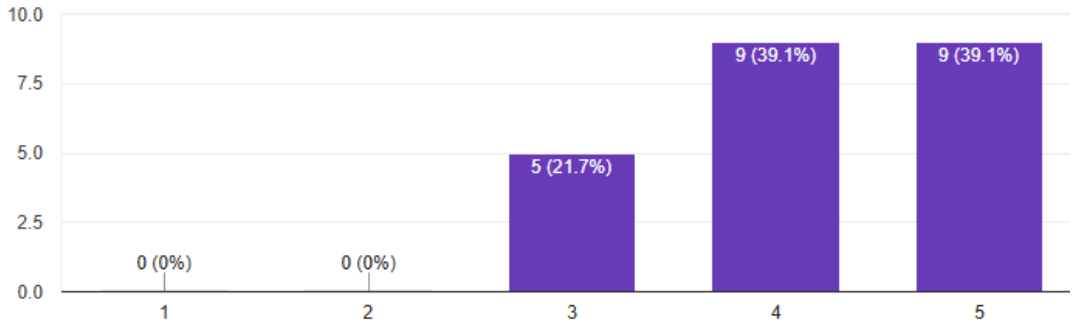


Figure 5.1: SUS question 1 results

Figure 5.1 illustrates the result for the question “I think I would like to use this application more frequently”. 5(21.7%) of the 21 respondents assigned a rating of 3 to the question, while the

remaining respondents allocated equal ratings of 4 (39.1%) and 5 (39.1%). The mean rating for this inquiry was 4.17.

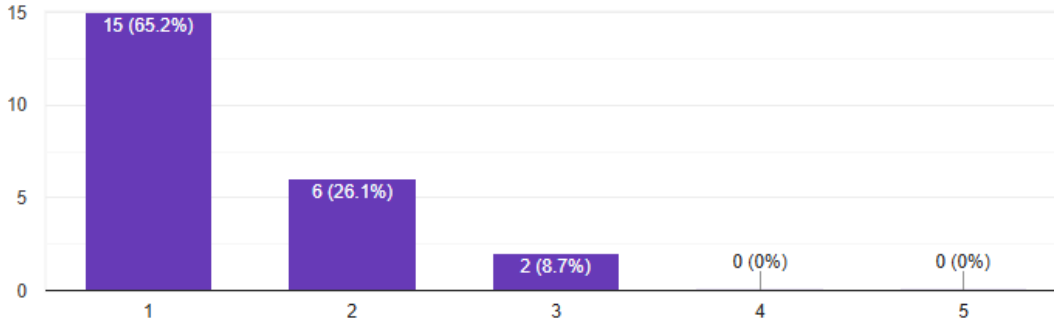


Figure 5.2: SUS question 2 results

Figure 5.2 depicts the result to the inquiry "I think the application is too complicated". Fifteen respondents (65.2%) assigned a rating of 1 to the question. Six respondents (26.1%) assigned a rating of 2, while three respondents (8.7%) assigned a rating of 3. The mean rating obtained from this inquiry was 1.43.

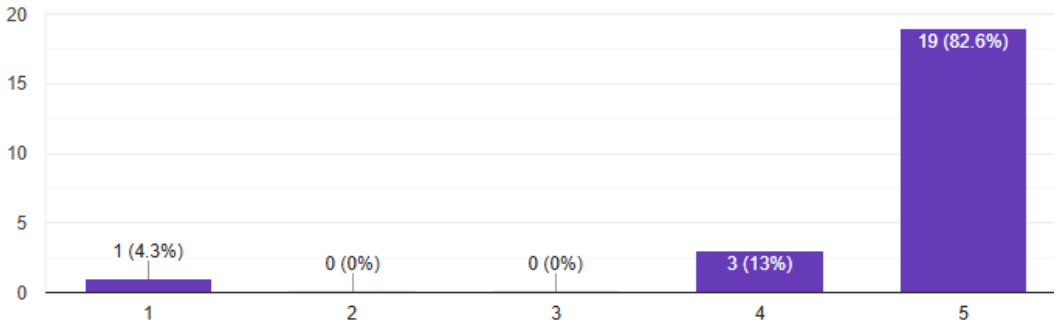


Figure 5.3: SUS question 3 results

As illustrated in Figure 5.3, the results of the question "I think the application is simple to use" are presented. One person (4.3%) has given the rating 1 for this inquiry. 4 people (13%) allocated a

rating of 4, while 19 respondents (82.6%) have responded the question with a rating of 5. The mean rating for this question was 4.70.

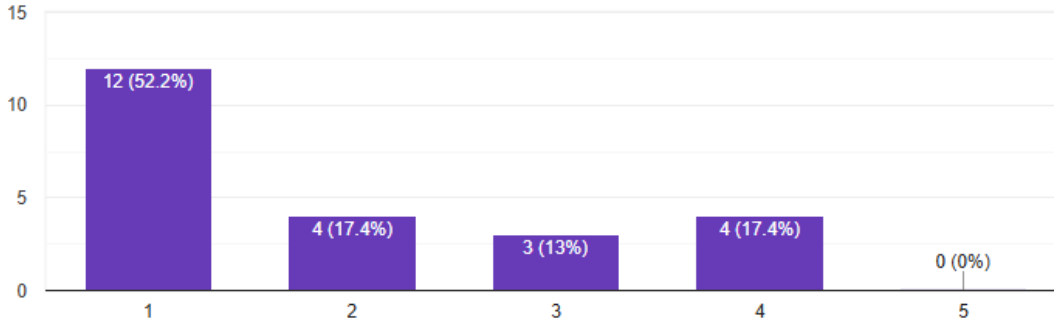


Figure 5.4: SUS question 4 results

Figure 5.4 depicts the results to the question "I would need help from a technical person to use this application". 52.2% of the total respondents have answered this question with a rating of 1. Four individuals (17.4%) assigned a rating of 2, 3 respondents (13%) allocated a rating of 3, and 4 respondents (17.4%) assigned a rating of 4. The mean rating for this aspect was 1.96.

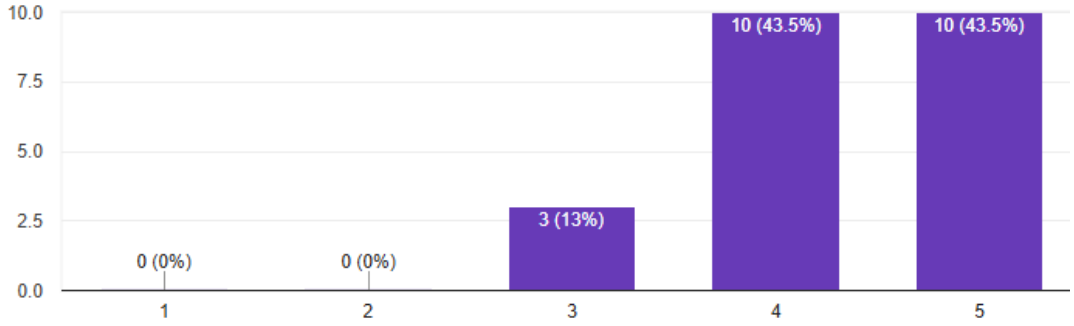


Figure 5.5: SUS question 5 results

Figure 5.5 presents the result to the inquiry "I think the different function/features of this application work well together". 3 individuals (13%) have given this question a rating of 3. The

remaining respondents have distributed their ratings equally between 4 (43.5%) and 5 (43.5%). The mean rating allocated for this inquiry was 4.30.

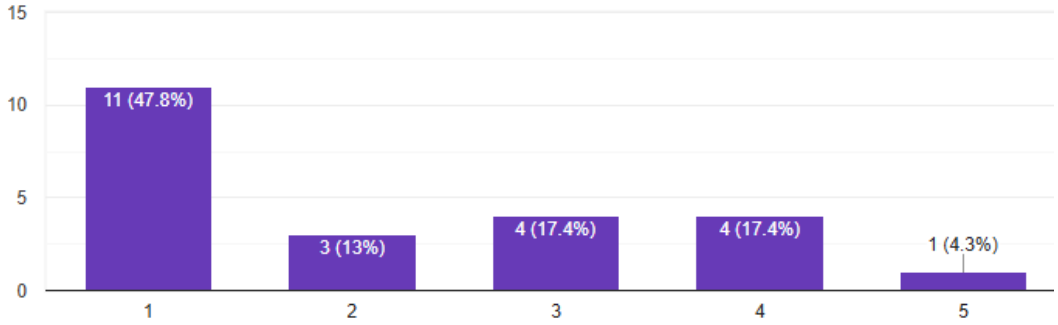


Figure 5.6: SUS question 6 results

Figure 5.6 provides the visual representation of the responses to the question "I think there is too much inconsistency in the application". 11 respondents (47.8%) provided a rating of 1, 3 respondents (13%) provided a rating of 2, and 4 respondents (17.4%) provided a rating of 4. The proportion of respondents who allocated a rating of 4 was equivalent to the proportion of respondents who allocated a rating of 3. A total of 4.3% of respondents have assigned a rating of 5 to the inquiry. The mean rating of the responses was 2.17.

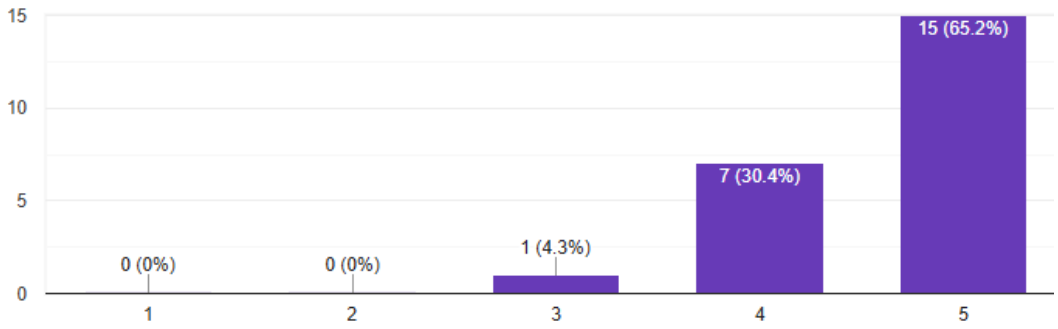


Figure 5.7: SUS question 7 results

As illustrated in Figure 5.7, one participant has assigned a rating of 3 to the question, “I imagine most people will learn to use the application very quickly.” 7 respondents (30.4%) assigned a rating of 4, while the remaining respondents (65.2%) assigned a rating of 5. The mean rating for this question was 4.61.

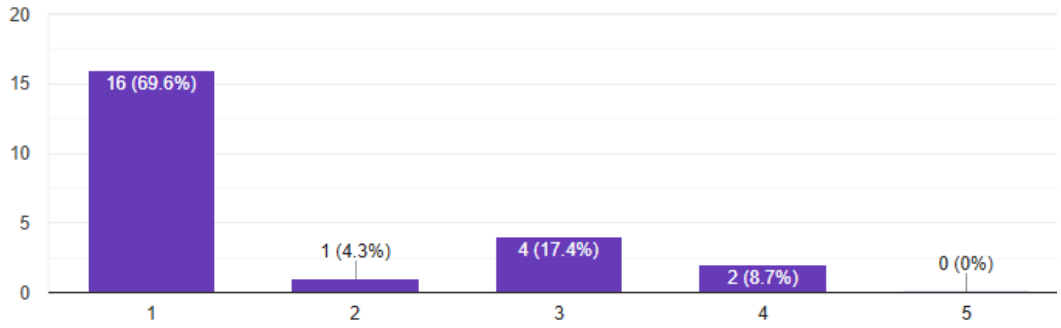


Figure 5.8: SUS question 8 results

As illustrated in Figure 5.8, 16 respondents (69.6%) have assigned a rating of 1 to the inquiry, "I find the application very tricky to use." One individual (4.3%) has also allocated a rating of 1. Meanwhile, 4 respondents (17.4%) have allocated a rating of 4, while the remaining 8.6% have not provided a response to the inquiry. The mean rating of the responses was 1.65.

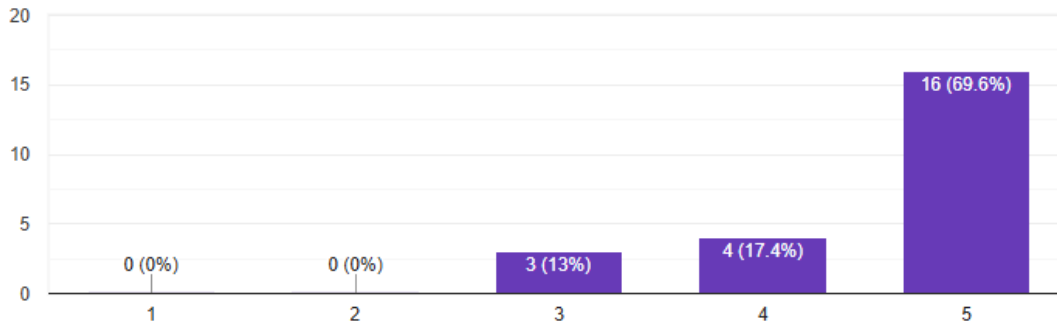


Figure 5.9: SUS question 9 results

As displayed in Figure 5.9, 3 respondents (13%) have assigned a rating of 3 to the inquiry "I feel very comfortable using this application." A total of 4 respondents (17.4%) assigned a rating of 4 to the inquiry, while the remaining 69.6% of respondents allocated a rating of 5 to the inquiry. The mean rating obtained was 4.57.

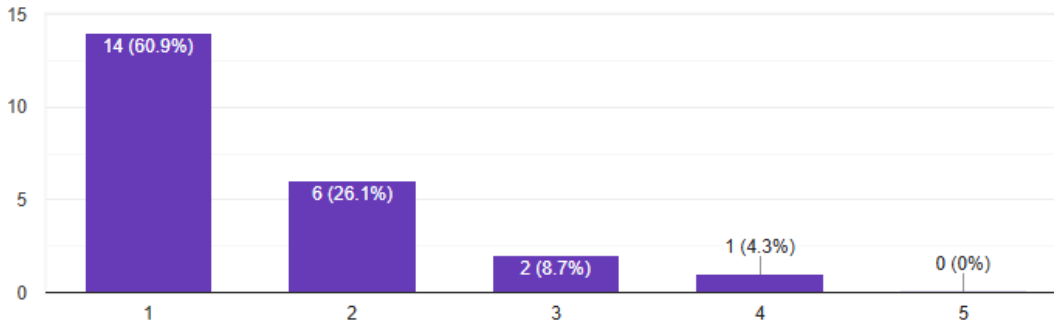


Figure 5.10: SUS question 10 results

As illustrated in Figure 5.10, the ratings assigned to the inquiry “I have to learn a lot of new skills before I can start using the application”. A total of 14 respondents (60.9%) provided a rating of 1, 2 respondents (26.1%) provided a rating of 2, 2 respondents (8.7%) provided a rating of 3, and 1 respondent (4.3%) provided a rating of 5. The mean rating allocated for this question was 1.57.

5.4.3.2 SUS score

Table 5.8: Ratings of the questions and SUS score

Respondent/ Question	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	SUS Score
R1	5	1	5	1	5	1	5	1	5	1	100.00
R2	4	1	1	1	4	4	5	4	5	1	70.00
R3	4	2	4	3	5	5	5	2	4	1	72.50
R4	4	2	5	3	4	3	5	1	5	1	82.50
R5	3	1	5	1	3	1	4	1	4	2	82.50
R6	5	1	5	1	5	1	5	1	5	1	100.00
R7	4	1	5	1	4	4	5	1	5	1	87.50
R8	5	1	5	1	5	1	4	1	5	1	97.50
R9	4	1	5	1	4	1	5	1	5	1	95.00

R10	5	1	5	1	5	2	5	1	5	1	97.50
R11	3	1	5	1	3	1	5	1	3	1	85.00
R12	5	2	5	2	4	1	5	1	5	1	92.50
R13	3	1	5	1	4	2	4	3	3	2	75.00
R14	4	1	5	1	4	1	5	1	5	1	95.00
R15	3	2	4	4	3	3	4	3	4	3	57.50
R16	5	1	5	2	5	4	5	1	5	2	87.50
R17	3	2	5	4	4	3	4	3	3	2	62.50
R18	4	1	5	1	5	1	5	1	5	1	97.50
R19	4	1	5	2	5	1	5	1	5	2	92.50
R20	5	1	5	3	4	2	5	1	5	2	87.50
R21	4	3	5	4	4	3	3	3	4	3	60.00
R22	5	3	4	4	5	4	4	4	5	4	60.00
R23	5	2	5	2	5	1	4	1	5	1	92.50

As illustrated in Table 5.8, the participants' ratings are presented alongside the calculated SUS score of the application. The mean SUS score collected from the valid respondents was 83.91. This finding suggests that the application provided a positive user experience to the respondents. It also suggests that the application was considered satisfactory, intuitive and effective by the respondents. The application was found to be capable of meeting the user requirements, requiring minimal prior experience or expertise to use, and functioning without significant assistance or inconvenience. This finding emphasises the potential for widespread acceptance and utilisation by the general public, which was the target demographic of this application.

5.5 Dicussion

The evaluation phase aimed to assess the effectiveness, accuracy, and time efficiency of the implemented bird sound recognition application. The evaluation process consisted of multiple components, including the evaluation of classification performance, mobile application functionality, and system usability testing.

The evaluation of the bird audio classification was conducted using multiple classification metrics, including accuracy, precision, recall and f1-score, as well as the time required to train and classify the sample dataset. It was demonstrated that, while the model required the most time to train from scratch, the Convolution Neural Network demonstrated the highest performance metrics among the three machine learning techniques. The accuracy recorded by CNN also indicated that approximately one third of the testing samples were correctly identified by the machine learning algorithm.

However, it was imperative to acknowledge that the dataset utilised in this study was derived from crowd-sourced recordings collected in a specific region. It was determined that the capacity of machine learning models to identify bird species was constrained by limitations. The quality and comprehensiveness of the data could affect the accuracy of the machine learning models with unseen data. In the event that the recordings were taken in a highly noisy environment, or if certain bird species made calls or songs that were not present in the training data, the models may fail to correctly identify the bird species. Nevertheless, the utilisation of recordings from crowd-sourced projects has the potential to provide a valuable insight into the identification of audio through various recording equipment, particularly through mobile devices.

The evaluation of the mobile application encompassed a range of criteria, including compatibility, functionality and usability. The compatibility of the application was conducted on various Android devices with different hardware and software configurations. It was determined that all three testing devices were capable of operating the application without encountering any unanticipated issues. This finding suggested that the framework employed for developing the application was robust, and that the application was relatively well-optimised, at least when operated on a device with Android version 12 and 6GB of RAM. Further development of the

mobile application could be undertaken for a more extensive range of mobile specifications, with a particular focus on iOS devices.

The testing of functionalities was conducted by accessing the various features and functions of the application. This encompassed the audio recording functionality, audio preprocessing, model classification, file storage and retrieval, user feedback, and user interface navigation. The functionality of each component was consistent with the expected outcomes, thereby facilitating the demonstration of the utilisation of machine learning solutions in practical, real-life scenarios.

The evaluation of the application's usability was conducted through the administration of a questionnaire utilising the System Usability Scale (SUS) framework (Brooke, 1996), and the subsequent calculation of scores for each aspect. The SUS score provided an overview of the application's usability from the perspective of the participants. The overall SUS score of 83.91 indicated favourable experiences with the application. Consequently, the necessity for substantial enhancements to the user experience design of the application may be reduced. The current design could be maintained in order to ensure a positive user experience.

To summarise, the analysis concluded that CNN was the most optimal machine learning technique for the classification of bird audio, despite its lengthy training time. In order to enhance the robustness of the CNN model, further optimisations towards the architecture of the model and an expansion of the training datasets with recordings from additional regions would be required. The application was found to be compatible with Android devices; however, further development may be required for iOS devices. The application functioned without any issues, and participants reported positive experiences with it.

5.6 Summary

Chapter 5 discussed a comprehensive evaluation, testing, and result analysis of the implemented bird sound recognition system. Convolution Neural Network (CNN) showed superior performance in terms of classification accuracy, while the mobile application demonstrated reliable functionality across different Android devices. The testing phase confirmed the reliability and functionality of the system across various operational scenarios. The evaluation results provided valuable insights into the system's strengths and highlighted areas where the machine learning approach successfully addresses the challenges of automated bird species identification through acoustic analysis.

CHAPTER 6: CONCLUSION AND FUTURE WORK

6.1 Achievements

The objectives of the project were successfully achieved, including comparative analysis of the machine learning techniques, and the development and implementation of a mobile application that integrates machine learning algorithms for bird sound recognition. This study examines three different machine learning techniques (Naïve Bayes, Support Vector Machine, and Convolutional Neural Network) to classify bird species from acoustic recordings. The effectiveness and efficiency of the proposed machine learning techniques were analysed and compared based on classification accuracy, precision, recall, F1-score, and time efficiency.

6.1.1 Technical achievements

This study has successfully integrated and trained three distinct machine learning models for the classification of bird sounds, thereby providing a comprehensive comparison of traditional machine learning approaches (Naïve Bayes, SVM) with deep learning methods (CNN). The implementation of a robust audio pre-processing pipeline has been achieved. This encompasses audio resampling and volume normalisation, as well as noise reduction through the utilisation of spectral gating algorithms. Furthermore, it involves the extraction of features and the segmentation of audio for the purpose of optimal classification.

Furthermore, a fully functional mobile application using React Native with Expo framework for recognising bird sounds was successfully created. The application was able to record sounds in real-time, classify recorded sounds using trained models, store and retrieve past

recordings, view detailed information about the identified bird species, and operate on a hybrid mode which only requires a server to pre-process the audio. The application has been optimised to a satisfactory degree, with the capacity to function on different mobile devices. It is also capable of executing classification tasks by utilising machine learning models that have been converted into ONNX format.

6.1.2 Research contributions

The study has conducted a comparative analysis of the machine learning approaches for bird sound classification, offering a comprehensive evaluation that highlights effective techniques for acoustic-based species identification. Furthermore, a practical and user-friendly tool was designed and developed to assist ornithologists, researchers, and bird enthusiasts in the field by automating species identification. The application's incorporation of accessibility features enabled users lacking specialized expertise to benefit from accurate species identification, thereby broadening the reach of ornithological knowledge and fostering greater engagement with wildlife monitoring and conservation.

6.2 Limitations

While the study achieved a number of achievements and contributed to the existing body of knowledge, it was not without limitations. The following sections are dedicated to the discussion of these topics.

6.2.1 Dataset limitations

At present, the study is limited to the avian species and their respective recordings obtained in and around the Sarawak region. This limitation renders the system's applicability exclusive to specific geographical areas characterised by distinct avian populations. Moreover, the efficacy of the system is contingent upon the quality of the training data. It is acknowledged that the nature of the data used in this study, which is crowd-sourced, may result in a reduced classification accuracy in cases where samples of certain bird species are of poor quality or limited in number. This, in turn, rendered it less robust to unseen data. The model failed to consider the seasonal variations in bird vocalisations, such as breeding calls versus regular calls, which may have an effect on the classification accuracy during different times of the year.

6.2.2 Technical limitations

Although noise reduction techniques are implemented during the pre-processing of audio classification, the system may still encounter difficulties with recordings that contain significant background noise or multiple overlapping bird calls. This also means that the system requires relatively clear audio recordings to achieve optimal performance, which is impractical to achieve if used in certain field conditions. Moreover, its application is confined to the specific avian species and their distinct vocalisations incorporated within the training dataset. In the event that the system encounters species or sounds that are not represented in the training data, misclassification or assignment to the closest matching species will be shown.

6.2.3 Application limitations

The application was exclusively designed and tested for Android devices; it has not been adequately developed and tested for iOS devices. Moreover, the incorporation of uncompressed bird images on the result page to facilitate the identification of species considerably increases the application size. This may present a challenge in terms of installation on devices with constrained storage capacity. Furthermore, the model was executed on the device itself, a factor which may have resulted in slower classification times in devices with limited processing power.

6.3 Future work

Subsequent research and development endeavours can address the identified limitations and extend the system's capabilities in a number of directions.

6.3.1 Dataset expansion and improvement

The dataset can be extended to include avian species and recordings from other regions, thus creating a more comprehensive and globally applicable bird sound recognition system. Furthermore, data augmentation techniques have the capacity to artificially expand the training dataset, with a particular application in the context of underrepresented species. This process has the dual benefits of reducing overfitting of the models and improving classification accuracy. The utilisation of newly collected recordings and user feedback from the feedback feature could also be employed for the continuous enhancement of the models and application.

6.3.2 Technical enhancements

The exploration of more sophisticated deep learning architectures could also be a fruitful avenue for enhancing the machine learning techniques. One such architectural solution would involve the utilisation of a transfer learning technique derived from pre-trained models. The utilisation of pre-trained models, such as those trained on large-scale audio datasets like BirdNet (Kahl et al., 2021), offers a potential avenue for enhancing the efficiency and accuracy of classification systems. This approach not only reduces the time required for training but also ensures the capture of relevant information from the features extracted, thereby improving the overall classification accuracy. The employment of ensemble or hybrid model approaches has the potential to facilitate enhanced classification. The integration of CNN with Recurrent Neural Networks (RNNs) or Transformers for temporal modelling is a key component of the proposed approach. The implementation of species-specific expert models with a gating mechanism is also a key component of the proposed approach, as is the use of knowledge distillation from multiple teacher models. Another approach that has the potential to contribute to improvements in this area is the implementation of mechanisms that are able to utilise multi-scale temporal analysis. Given the variability in duration exhibited by avian auditory signals, architectures designed to process fixed-duration audio may encounter challenges in capturing the temporal diversity present. This encompasses dilated convolutions or wavelet-based approaches that are capable of capturing both brief calls and more protracted songs, in addition to developing hierarchical models that process varying temporal scales in parallel.

In addition, there are several feature engineering improvements that have the potential to enhance classification accuracy. The implementation of additional acoustic characteristics beyond

those currently employed, including gammatone features, learnable feature extraction layers that adapt to particular bird characteristics, and attention mechanisms that dynamically weight different feature types, will prove advantageous in enhancing the classification of machine learning techniques. The enhancement of the context of features has the potential to contribute to the optimisation of classification outcomes. In order to incorporate temporal context, it is necessary to analyse sequences of calls rather than isolated segments. Furthermore, it may be beneficial to utilise auxiliary inputs, such as time of day, season and geographic location, in order to determine the relevant class for a given recording.

6.3.3 Application development

It is also acknowledged that several enhancements can be made to the mobile application to improve its usability and performance. The pre-processing of audio functions can be implemented to allow for the application to be used fully offline, thereby eliminating the need for an internet connection. The implementation of edge computing techniques is another consideration that can support and improve the classification on devices that have low computing power. The development of more efficient model architectures has the potential to reduce both classification time and storage requirements. Moreover, the application has the potential to leverage the advanced hardware acceleration that is becoming increasingly prevalent in the mobile market.

The application's compatibility with iOS platforms facilitates the extension of its user base and ensures the consistency of its functionality across diverse mobile operating systems. The integration of cloud-based functionalities, including data synchronisation across multiple devices, access to continuously updated models, and community-driven data collection and validation, can

be implemented into the application to enhance its functionality. Furthermore, integration with citizen science projects and birding communities such as Xeno Canto, along with GPS and time data for improved location-based and time-based species suggestions, has the potential to enhance the user experience of the application.

The system can also be extended to include the identification of species, as well as the analysis of behavioural contexts, ecological monitoring, and integration with Internet of Things (IoT) devices. The variation in bird calls, including territorial calls versus alarm calls, mating calls and seasonal variations, and stress indicators in vocalisations, can aid in improving the understanding of bird species that are examined. The enhancement of acoustic analysis capabilities to evaluate population trends and ecosystem health can be advantageous in the context of long-term biodiversity monitoring. The integration of IoT devices, such as passive automated monitoring stations, with environmental monitoring networks in remote areas is a potential avenue for enhancement.

REFERENCES

Abdulkareem, N. M., & Abdulazeez, A. M. (2021). Machine Learning Classification Based on Radom Forest Algorithm: A Review. https://doi.org/10.5281/ZENODO.4471118

Adavanne, S., Drossos, K., Cakir, E., & Virtanen, T. (2017). Stacked convolutional and recurrent neural networks for bird audio detection. 2017 25th European Signal Processing Conference (EUSIPCO), 1729–1733. https://doi.org/10.23919/EUSIPCO.2017.8081505

AviaNZ. (2021). http://www.avianz.net/

Avisoft Bioacoustics. (2022). Avisoft-SASLab Pro. https://avisoft.com/sound-analysis/

Brooke, J. (1996). SUS-A quick and dirty usability scale. Usability evaluation in industry, 189(194), 4-7.

Charbuty, B., & Abdulazeez, A. (2021). Classification Based on Decision Tree Algorithm for Machine Learning. Journal of Applied Science and Technology Trends, 2(01), 20–28. https://doi.org/10.38094/jastt20165

Connection Rainforest. (2022). Arbimon. https://arbimon.org/

Cornell Lab of Ornithology. (2022a). BirdNET Sound ID. https://birdnet.cornell.edu/

Cornell Lab of Ornithology. (2022b). Merlin Bird ID. https://merlin.allaboutbirds.org/sound-id/

Demir, F., Turkoglu, M., Aslan, M., & Sengur, A. (2020). A new pyramidal concatenated CNN approach for environmental sound classification. Applied Acoustics, 170, 107520.

https://doi.org/10.1016/j.apacoust.2020.107520

Didier, B., & Yves, B. (2022). TadariDeep. https://github.com/YvesBas/TadariDeep

Dong, X., Towsey, M., Zhang, J., Banks, J., Roe, P. (2013). A novel representation of bioacoustic events for content-based search in field audio data. 2013 International Conference on Digital Image Computing: Techniques and Applications (DICTA), 1–6. IEEE.

Dwivedi, A. K., Imtiaz, S. A., & Rodriguez-Villegas, E. (2019). Algorithms for Automatic Analysis and Classification of Heart Sounds–A Systematic Review. IEEE Access, 7, 8316–8345. https://doi.org/10.1109/ACCESS.2018.2889437

Ekpezu, A. O., Katsriku, F., Yaokumah, W., & Wiafe, I. (2022). The Use of Machine Learning Algorithms in the Classification of Sound: A Systematic Review. International Journal of Service Science, Management, Engineering, and Technology, 13(1), 1–28. https://doi.org/10.4018/IJSSMET.298667

Fagerlund, S. (2007). Bird species recognition using support vector machines. EURASIP Journal on Advances in Signal Processing, 2007(1), 038637. https://doi.org/10.1155/2007/38637

Gibb, R., Browning, E., Glover-Kapfer, P., Jones, K.E. (2019). Emerging opportunities and challenges for passive acoustics in ecological assessment and monitoring. Methods Ecol. Evol, 10(2), 169–185.

Green, S., Marler, P. (1979). The analysis of animal communication. Social behavior and communication, 73–158. Springer.

Gupta, G., Kshirsagar, M., Zhong, M., Gholami, S., & Ferres, J. L. (2021). Comparing recurrent convolutional neural networks for large scale bird species classification. Scientific Reports, 11(1), 17085. https://doi.org/10.1038/s41598-021-96446-w

Hafner, S. D., Katz, J., & Donovan, T. (2018). Package monitoR. CRAN. https://cran.r-project.org/web/packages/monitoR/index.html

Howard, A., Joly, A., Klinck, H., Dane, S., Kahl, S., Denton, T., & Denton, T. (2021). BirdCLEF 2021 - Birdcall Identification. Retrieved from https://kaggle.com/competitions/birdclef-2021

International Organization for Standardization. (2019). Introduction to the new ISO 8601-1 and ISO 8601-2. ISO. https://www.isotc154.org/posts/2019-08-27-introduction-to-the-new-8601/

IUCN (2021). The IUCN red list of threatened species. https://www.iucnredlist.org/.

K. Lisa Yang Center for Conservation Bioacoustics. (2022). Raven Pro. Cornell Lab of Ornithology Cornell University. https://ravensoundsoftware.com/software/raven-pro/

Kahl, S., Wood, C. M., Eibl, M., & Klinck, H. (2021). BirdNET: A deep learning solution for avian diversity monitoring. Ecological Informatics, 61, 101236. https://doi.org/10.1016/j.ecoinf.2021.101236

Kasten, E.P., Gage, S.H., Fox, J., Joo, W. (2012). The remote environmental assessment laboratory's acoustic library: an archive for studying soundscape ecology. Ecol. Inform.,

12, 50–67.

Kumar, D., Mounika, S., Barnwal, K., Azhari, M., & Kaur, U. (2024). Birds Species Identification. SSRN Electronic Journal. https://doi.org/10.2139/ssrn.4495994

Kumon, M., Fukunaga, R., Manabe, T., & Nakatsuma, K. (2022). Object Surface Recognition Based on Standing Waves in Acoustic Signals. Frontiers in Robotics and AI, 9, 872964. https://doi.org/10.3389/frobt.2022.872964

Kwan, C., Mei, G., Zhao, X., Ren, Z., Xu, R., Stanford, V., Rochet, C., Aube, J., & Ho, K. C. (2004). Bird classification algorithms: Theory and experimental results. 2004 IEEE International Conference on Acoustics, Speech, and Signal Processing, 5, V-289–292. https://doi.org/10.1109/ICASSP.2004.1327104

Lachlan, R. (2016). Luscinia. https://rflachlan.github.io/Luscinia/

Lehikoinen, P., Rannisto, M., Camargo, U., Aintila, A., Lauha, P., Piirainen, E., Somervuo, P., & Ovaskainen, O. (2023). A Successful Crowdsourcing Approach for Bird Sound Classification. Citizen Science: Theory and Practice, 8(1), 16. https://doi.org/10.5334/cstp.556

Maclean, K., & Triguero, I. (2023). Identifying bird species by their calls in Soundscapes. Applied Intelligence, 53(19), 21485–21499. https://doi.org/10.1007/s10489-023-04486-8

Mehyadin, A. E., Abdulazeez, A. M., Hasan, D. A., & Saeed, J. N. (2021). Birds Sound Classification Based on Machine Learning Algorithms. Asian Journal of Research in

Computer Science, 1–11. https://doi.org/10.9734/ajrcos/2021/v9i430227

Meta Open Source. (2025, April 14). Get started with react native · REACT native. React Native. https://reactnative.dev/docs/environment-setup

Microsoft. (2022). Visual Studio Code [Computer Software]. Retrieved from https://code.visualstudio.com

Morrison, C. A., Auniņš, A., Benkő, Z., Brotons, L., Chodkiewicz, T., Chylarecki, P., Escandell, V., Eskildsen, D. P., Gamero, A., Herrando, S., Jiguet, F., Kålås, J. A., Kamp, J., Klvaňová, A., Kmecl, P., Lehikoinen, A., Lindström, Å., Moshøj, C., Noble, D. G., … Butler, S. J. (2021). Bird population declines and species turnover are changing the acoustic properties of spring soundscapes. Nature Communications, 12(1), 6217. https://doi.org/10.1038/s41467-021-26488-1

Open Neural Network Exchange. (2024). Converters¶. Converters - ONNX 1.19.0 documentation. https://onnx.ai/onnx/intro/converters.html

Ovaskainen, O., Moliterno de Camargo, U., & Somervuo, P. (2018). Animal sound identifier (ASI): Software for automated identification of Vocal Animals. Ecology Letters, 21(8), 1244–1254. https://doi.org/10.1111/ele.13092

Payne, R. B. (1979). Song structure, behaviour, and sequence of song types in a population of village indigobirds, Vidua chalybeata. Animal Behaviour, 27, 997–1013. https://doi.org/10.1016/0003-3472(79)90047-2

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

Ramashini, M., Abas, P. E., Mohanchandra, K., & De Silva, L. C. (2022). Robust cepstral feature for bird sound classification. International Journal of Electrical and Computer Engineering (IJECE), 12(2), 1477. https://doi.org/10.11591/ijece.v12i2.pp1477-1487

Rassak, S., Nachamai, M., & Krishna, M. A. (2016). Survey study on the methods of bird vocalization classification. 2016 IEEE International Conference on Current Trends in Advanced Computing (ICCTAC), 1–8. https://doi.org/10.1109/ICCTAC.2016.7567337

Ruff, Z. (2022). Shiny_PNW-Cnet. https://github.com/zjruff/Shiny_PNW-Cnet

Sabour, S., Frosst, N., & Hinton, G. (2017). Dynamic Routing Between Capsules. Advances in Neural Information Processing Systems. Curran Associates, Inc.

Sainburg, T., Thielk, M., & Gentner, T. Q. (2020). Finding, visualizing, and quantifying latent structure across diverse animal vocal repertoires. PLOS Computational Biology, 16(10), e1008228. https://doi.org/10.1371/journal.pcbi.1008228

Sainburg, T., Thielk, M., & Gentner, T. Q. (2020). Finding, visualizing, and quantifying latent structure across diverse animal vocal repertoires. PLOS Computational Biology, 16(10), e1008228. https://doi.org/10.1371/journal.pcbi.1008228

Sedlacek, O., Vokurkova, J., Ferenc, M., Djomo, E.N., Albrecht, T., Horak, D. (2015). A comparison of point counts with a new acoustic sampling method: a case study of a bird community from the montane forests of Mount Cameroon. Ostrich, 86(3), 213–220.

Selin, A., Turunen, J., & Tanttu, J. T. (2006). Wavelets in recognition of bird sounds. EURASIP Journal on Advances in Signal Processing, 2007(1), 1–9. https://doi.org/10.1155/2007/51806

Shtylman, R. (2022). Localtunnel ~ expose yourself to the world. https://theboroer.github.io/localtunnel-www/

Silva, B. (2022). Package soundClass. CRAN. https://cran.r-project.org/web/packages/soundClass/index.html

Spiny Software. (2020). ChirpOMatic. http://www.chirpomatic.com/

Sun, L., Lyu, G., Feng, S., & Huang, X. (2021). Beyond missing: Weakly-supervised multi-label learning with incomplete and noisy labels. Applied Intelligence, 51(3), 1552–1564. https://doi.org/10.1007/s10489-020-01878-y

Tchernichovski, O., Nottebohm, F., Ho, C. E., Pesaran, B., & Mitra, P. P. (2000). A procedure for an automated measurement of Song similarity. Animal Behaviour, 59(6), 1167–1176. https://doi.org/10.1006/anbe.1999.1416

The MathWorks (2020). Predictive maintenance toolboxTM.

Tivarekar R. P., Chavan V. D., Shete S. A., & Vartak A. B. Audio based bird species recognition

using naïve bayes algorithm. (2018). International Journal of Modern Trends in Engineering & Research, 5(1), 117–124. https://doi.org/10.21884/IJMTER.2018.5020.SHOJV

Truong, T. H., Nguyen, H. D., Mai, T. Q. A., Nguyen, H. L., Dang, T. N. M., & Phan, T.-T.-H. (2023). A deep learning-based approach for bee sound identification. Ecological Informatics, 78, 102274. https://doi.org/10.1016/j.ecoinf.2023.102274

Vapnik, V. N. (2013). The nature of statistical learning theory. Springer International Publishing.

Varghese, A., Shyamkrishna, K., & Rajeswari, M. (2022). Utilization of deep learning technology in recognizing bird species. AIP Conference Proceedings. https://doi.org/10.1063/5.0080446

Wan, P. (2014). The mobile bird identification system research based on bird song. Ph.D. thesis. China Jiliang University.

Wheeldon, A., Mossman, H.L., Sullivan, M.J., Mathenge, J., de Kort, S.R. (2019). Comparison of acoustic and traditional point count methods to assess bird diversity and composition in the Aberdare National Park, Kenya. Afr. J. Ecol, 57(2), 168–176.

Wildlife Acoustics. (2022). Kaleidoscope Pro Analysis Software. https://www.wildlifeacoustics.com/products/kaleidoscope-pro

Xiao, H., Liu, D., Chen, K., & Zhu, M. (2022). AMResNet: An automatic recognition model of bird sounds in real environment. Applied Acoustics, 201, 109121.

https://doi.org/10.1016/j.apacoust.2022.109121

Xie, J., Zhong, Y., Zhang, J., Liu, S., Ding, C., & Triantafyllopoulos, A. (2023). A review of automatic recognition technology for bird vocalizations in the deep learning era. Ecological Informatics, 73. https://doi.org/10.1016/j.ecoinf.2022.101927

Yellow Cardinal Inc. (2022). Smart Bird ID. https://smartbirdid.com/

Zhang, S., Gao, Y., Cai, J., Yang, H., Zhao, Q., & Pan, F. (2023). A novel bird sound recognition method based on multifeature fusion and a transformer encoder. Sensors, 23(19), 8099. https://doi.org/10.3390/s23198099

Zhou, Z.-H. (2018). A brief introduction to weakly supervised learning. National Science Review, 5(1), 44–53. https://doi.org/10.1093/nsr/nwx106

APPENDICES

Appendix A: Questionaire

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

Appendix B: Questionaire responses

Timestamp	Email address	I think I would like to use this application mod	I think the application is too complicated	I think the application is simple to use	I would need help from a technical person to	I think the different function/features of this ap	I think there is too much inconsistency in the	I imagine most people will learn to use the ap	I find the application very tricky to use	I feel very comfortable using the application	I have to learn a lot of new skills before I can start using the application
6/15/2025 13:16:43	78061@siswa.unimas	5	1	5	1	5	1	5	1	5	1
6/15/2025 13:36:06	79790@siswa.unimas	4	1	1	1	4	4	5	4	5	1
6/16/2025 2:17:33	78363@siswa.unimas	4	2	4	3	5	5	5	2	4	1
6/16/2025 2:17:48	81114@siswa.unimas	4	2	5	3	4	3	5	1	5	1
6/16/2025 2:26:34	79242@siswa.unimas	3	1	5	1	3	1	4	1	4	2
6/16/2025 2:27:36	79354@siswa.unimas	5	1	5	1	5	1	5	1	5	1
6/16/2025 2:35:18	jee.pc0102@gmail.cor	4	1	5	1	4	4	5	1	5	1
6/16/2025 3:07:30	79802@siswa.unimas	5	1	5	1	5	1	4	1	5	1
6/16/2025 3:07:42	79861@siswa.unimas	4	1	5	1	4	1	5	1	5	1
6/16/2025 3:09:51	thasheela1226@gmail	5	1	5	1	5	2	5	1	5	1
6/16/2025 3:11:07	khaiyangchin@gmail.c	3	1	5	1	3	1	5	1	3	1
6/16/2025 3:31:30	melvina020528@gmail	5	5	5	5	5	5	5	5	5	5
6/16/2025 3:31:32	79256@siswa.unimas	5	2	5	2	4	1	5	1	5	1
6/16/2025 3:31:57	79115@siswa.unimas	3	1	5	1	4	2	4	3	3	2
6/16/2025 8:03:26	81021@siswa.unimas	4	1	5	1	4	1	5	1	5	1
6/16/2025 8:21:57	104778@siswa.unima	1	5	5	5	5	5	5	5	5	5
6/16/2025 8:23:09	kwsiewung@gmail.cor	3	2	4	4	3	3	4	3	4	3
6/16/2025 9:27:34	82102@siswa.unimas	5	1	5	2	5	4	5	1	5	2
6/16/2025 9:28:03	79980@siswa.unimas	3	2	5	4	4	3	4	3	3	2
6/16/2025 9:37:24	81895@siswa.unimas	4	1	5	1	5	1	5	1	5	1
6/16/2025 9:38:46	81800@siswa.unimas	4	1	5	2	5	1	5	1	5	2
6/16/2025 9:46:43	jeffeesy@gmail.com	5	1	5	3	4	2	5	1	5	2
6/16/2025 9:48:26	josh050507@gmail.co	4	3	5	4	4	3	3	3	4	3
6/16/2025 10:04:18	79084@siswa.unimas	5	3	4	4	5	4	4	4	5	4
6/17/2025 3:01:22	meatan69@gmail.com	5	2	5	2	5	1	4	1	5	1

Appendix C: Source code of the project

featureExtraction.py

```
import librosa
import pyloudnorm
import numpy as np
from scipy.signal import hilbert
import time
import noisereduce
import matplotlib.pyplot as plot
import os
from pathlib import Path

def removeSilence (
    sound,
    rate,
    smoothWindow = 0.05,
    silencePercentile = 5,
    minSilenceDuration = 0.2,
    bufferDuration = 0.1
):

  """
  To remove silence in audio
  """

  # Compute amplitude envolope using Hilbert transform
  analyticSignal = hilbert(sound)
  amplitudeEnvelope = np.abs(analyticSignal)

  # Smooth the envelope with a moving average
  smoothWindowSize = int(rate * smoothWindow)
  smoothFilter = np.ones(smoothWindowSize) / smoothWindowSize
  smoothedEnvelope = np.convolve(
    amplitudeEnvelope,
    smoothFilter,
    mode='same'
    )

  # Calculate threshold based on average of smoothed envelope
  threshold = np.percentile(smoothedEnvelope, silencePercentile)

  # Identify silent segments where envelope is below the threshold
  mask = smoothedEnvelope < threshold

  # Find transitions between silent and non-silent segments
  changes = np.diff(mask.astype(int))
  starts = np.where(changes == 1)[0] + 1
  ends = np.where(changes == -1)[0] + 1
```

```
  # Handle silence at the begging or end
  if mask[0]:
    starts = np.insert(starts, 0, 0)
  if mask[-1]:
    ends = np.append(ends, len(mask))

  silentRegions = list(zip(starts, ends))

  # Apply minimum silence durationn
  minSilenceSample = int(minSilenceDuration * rate)
  validSilence = []
  for start, end in silentRegions:
    if (end - start >= minSilenceSample): {
      validSilence.append((start, end))
    }

  # Determine non-silent segments by inverting silent segments
  nonSilent = []
  prevEnd = 0
  for start, end in validSilence:
    nonSilent.append((prevEnd, start))
    prevEnd = end
  nonSilent.append((prevEnd, len(sound)))

  # Add buffer around non-silent segments
  bufferSamples = int(bufferDuration * rate)
  segments = []
  for start, end in nonSilent:
    segmentStart = max(start - bufferSamples, 0)
    segmentEnd = min(end + bufferSamples, len(sound))
    segments.append(sound[segmentStart:segmentEnd])

  # Concatinate the segments and handle cases where there aren't any silence
  if not segments:
    processedAudio = sound
  else:
    processedAudio = np.concatenate(segments)

  return processedAudio


"""
Extracting feature from the sound files.
"""
def featureExtraction(
    inputName,
    outputFilePath,
    outputName,
    startTime,
    processedLogPath,
    errorLogPath
  ):
  try:
    programStartTime = time.monotonic()
```

```
    # Load sound file
    sound, rate = librosa.load(inputName, sr=32000, mono=True)

    # Normalize peak volume to -3dB
    sound = pyloudnorm.normalize.peak(sound, -3.0)

    # Denoise the audio
    sound = noisereduce.reduce_noise(
      y=sound,
      sr=rate,
      stationary=False,
      chunk_size=1200000
    )

    # Remove silence
    sound = removeSilence(sound=sound,rate=rate)

    # Save the processed audio in 3s only segments
    duration = 3
    segmentSample = int(duration * rate)
    segmentsArray = []
    for i in range (0, len(sound), segmentSample):
      if (len(sound[i:i+segmentSample]) >= segmentSample):
        segmentsArray.append(sound[i:i+segmentSample])

    # File path for the features (mfcc, chroma, tonnetz)
    logMelPath = os.path.join(outputFilePath, "logmel")
    mfccPath = os.path.join(outputFilePath, "mfcc")
    chromaPath = os.path.join(outputFilePath, "chroma")
    tonnetzPath = os.path.join(outputFilePath, "tonnetz")

    # Create folder for features if doesn't exist
    Path(logMelPath).mkdir(parents=True, exist_ok=True)
    Path(mfccPath).mkdir(parents=True, exist_ok=True)
    Path(chromaPath).mkdir(parents=True, exist_ok=True)
    Path(tonnetzPath).mkdir(parents=True, exist_ok=True)

    #Process the segments into log-mel spectrogram, mfcc, chroma, tonnetz
    mfcc = None
    chroma = None
    tonnetz = None
    counter = 0

    for segment in segmentsArray:
      # For processing mfcc, chroma, tonnetz
      spectrogram = librosa.feature.melspectrogram(y=segment, sr=rate,
n_fft=1024, hop_length=256)
      logMelSpectrogram = librosa.power_to_db(S=spectrogram, ref=np.max)
      mfcc = librosa.feature.mfcc(y=segment, sr=rate, n_mfcc=20)
      chroma = librosa.feature.chroma_cqt(y=segment, sr=rate, n_chroma=12)
      tonnetz = librosa.feature.tonnetz(chroma=chroma)

      # Save the features
```

```
        saveName = outputName + "[" + str(counter) + "]" + ".npy"
        np.save(os.path.join(logMelPath,saveName), logMelSpectrogram)
        np.save(os.path.join(mfccPath, saveName), mfcc)
        np.save(os.path.join(chromaPath, saveName), chroma)
        np.save(os.path.join(tonnetzPath, saveName), tonnetz)
        counter += 1

      # Append the processed audio file to the processed log.
      with open(processedLogPath, 'a') as logFile:
        logFile.write(f"{inputName}\n")

      # To print the process completion.
      print(f"Finish file {inputName} in \n{time.monotonic() -
programStartTime}s.")
      print(f"Time elapsed: {(time.monotonic() - startTime)/60}min")

    # Print the error
    except Exception as e:
      with open(errorLogPath, 'a') as errorLogFile:
        errorLogFile.write(f"{inputName}:")
        errorLogFile.write(f"{repr(e)}")
      print(f"Critical error found in {inputName}!")

      return None
```

processAudio.py

```
import multiprocessing
import os
from pathlib import Path
from featureExtraction import featureExtraction
import time

#To measure seconds passed
start = time.monotonic()

if __name__ == "__main__":
  inputFilePath = "/path/to/Audio"
  outputfilePath = "/path/to/Features"
  logPath = "/path/to/Log"
  processedLogName = "processedLog.txt"
  errorLogName = "errorLog.txt"
  fileList = []

  #Create the log folder if it doesn't exists.
  Path(logPath).mkdir(parents=True, exist_ok=True)
  processedLogPath = os.path.join(logPath, processedLogName)
  errorLogPath = os.path.join(logPath, errorLogName)
  try:
    with open(processedLogPath, "r") as file:
      processedFile = file.read().splitlines()
  except:
    processedFile = None
```

```
 #Get the file names of the audio
  for dirName in os.listdir(inputFilePath):
    directoryPath = os.path.join(inputFilePath, dirName)
    if os.path.isdir(directoryPath):
      outputDir = os.path.join(outputfilePath, dirName)
      for file in os.listdir(directoryPath):
        if file.lower().endswith('.ogg') or file.lower().endswith('.wav') or
file.lower().endswith('.mp3'):
          #Create the output directory if it doesn't exist
          Path(os.path.join(directoryPath, outputDir)).mkdir(parents=True,
exist_ok=True)

          #Check if the recording has been processed before, skip if so
          if (processedFile is None) or (os.path.join(directoryPath, file)
not in processedFile):
            print(file)
            fileList.append((directoryPath, outputDir, file))
      print()

  if fileList is not None:
    args = [
      (
        os.path.join(inputDir, fileName),
        os.path.join(outputDir),
        fileName[:-4],
        start,
        processedLogPath,
        errorLogPath
      ) for (inputDir, outputDir, fileName) in fileList
    ]

    #Extract the features in multi-core process
    with multiprocessing.Pool(5) as pool:
      pool.starmap(featureExtraction, args)

  print("Finish process.")
```

classify(NB-SVM).py

```
import time
import numpy as np
import os
from sklearn.model_selection import StratifiedKFold
from sklearn.naive_bayes import ComplementNB
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
from sklearn.preprocessing import LabelEncoder, StandardScaler,
QuantileTransformer
from concurrent.futures import ThreadPoolExecutor

from sklearn.svm import SVC

def loadDataParallel(mainDir, maxWorkers=None):
    xLogMel, xMfcc, xChroma, xTonnetz, y, baseNames = [], [], [], [], [], []
```

```
    # First collect all valid file paths
    tasks = []
    for classLabel in os.listdir(mainDir):
        classDirectory = os.path.join(mainDir, classLabel)
        if not os.path.isdir(classDirectory):
            continue

        logMelDir = os.path.join(classDirectory, 'logmel')
        mfccDir = os.path.join(classDirectory, 'mfcc')
        chromaDir = os.path.join(classDirectory, 'chroma')
        tonnetzDir = os.path.join(classDirectory, 'tonnetz')

        if not all(os.path.isdir(d) for d in [logMelDir, mfccDir, chromaDir,
tonnetzDir]):
            continue

        sampleFiles = [f for f in os.listdir(mfccDir) if f.endswith('.npy')]

        # Group files by base name (without segment index)
        file_groups = {}
        for fileName in sampleFiles:
            # Extract base name (remove segment index)
            if '[' in fileName and fileName.endswith('].npy'):
                base_name = fileName.split('[')[0]
            else:
                base_name = os.path.splitext(fileName)[0]

            if base_name not in file_groups:
                file_groups[base_name] = []
            file_groups[base_name].append(fileName)

        # Create tasks per base name
        for base_name, files in file_groups.items():
            for fileName in files:
                paths = (
                    os.path.join(logMelDir, fileName),
                    os.path.join(mfccDir, fileName),
                    os.path.join(chromaDir, fileName),
                    os.path.join(tonnetzDir, fileName)
                )
                if all(os.path.isfile(p) for p in paths):
                    tasks.append((paths, classLabel, base_name))

    # Parallel loading function
    def loadFiles(task):
        paths, classLabel, base_name = task
        try:
            return (
                np.load(paths[0]).flatten(),
                np.load(paths[1]).flatten(),
                np.load(paths[2]).flatten(),
                np.load(paths[3]).flatten(),
                classLabel,
```

```
                base_name
            )
        except Exception as e:
            print(f"Error loading {paths[0]}: {str(e)}")
            return None

    # Process tasks in parallel
    with ThreadPoolExecutor(max_workers=maxWorkers) as executor:
        results = executor.map(loadFiles, tasks)

        for result in results:
            if result is not None:
                logMel, mfcc, chroma, tonnetz, label, base_name = result
                xLogMel.append(logMel)
                xMfcc.append(mfcc)
                xChroma.append(chroma)
                xTonnetz.append(tonnetz)
                y.append(label)
                baseNames.append(base_name)

    return np.array(xLogMel), np.array(xMfcc), np.array(xChroma),
np.array(xTonnetz), np.array(y), np.array(baseNames)

# Record time to log file
def logging(logText, logFile):
  with open(logFile, 'a') as file:
    file.write(f"{time.time()}:\n")
    file.write(f"{logText}\n\n")

# Classify with SVM
def svm_classify(Xtrain, yTrain, Xtest, logFile):
  # Create model
  svm = SVC(kernel='poly',
            max_iter=-1,
            C=394.07472067394843,
            coef0=0.10770856880264157,
            decision_function_shape='ovo',
            degree=2,
            gamma='auto',
            tol=0.0007730292273498736,
            random_state=42,
            class_weight='balanced')

  # Train model
  svm.fit(Xtrain, yTrain)

  # Evaluate
  logging(f"SVM model trained with {Xtrain.shape[0]} samples and
{Xtrain.shape[1]} features.",
          logFile)
  resultTrain, resultTest = svm.predict(Xtrain), svm.predict(Xtest)
  logging(f"SVM model evaluated on {Xtest.shape[0]} test samples.",
          logFile)
  return resultTrain, resultTest
```

```
# Classify with Naive Bayes
def nb_classify(Xtrain, yTrain, Xtest, logFile):
  # Min max scaling to remove negative values
  scaler = QuantileTransformer(subsample=None)
  Xtrain = scaler.fit_transform(Xtrain)
  Xtest = scaler.transform(Xtest)

  # Train model
  nb = ComplementNB(alpha=0.10675620295084332, norm=True)
  nb.fit(Xtrain, yTrain)

  # Evaluate
  logging(f"Naive Bayes model trained with {Xtrain.shape[0]} samples and
{Xtrain.shape[1]} features.",
          logFile)
  resultTrain, resultTest = nb.predict(Xtrain), nb.predict(Xtest)
  logging(f"Naive Bayes model evaluated on {Xtest.shape[0]} test samples.",
          logFile)
  return resultTrain, resultTest


# Main extract features file
def classify(xLogMel,
             xMfcc,
             xChroma,
             xTonnetz,
             yLabel,
             baseNames,
             logFile,
             classifyType,
             standardized=False):
  # Get unique base names and their labels
  uniqueBaseNames = np.unique(baseNames)
  baseNameLabels = np.array([yLabel[np.where(baseNames == base)[0][0]] for
base in uniqueBaseNames])


  # Initialize cross-validation
  skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
  trainAccuracy = []
  testAccuracy = []
  trainRecall = []
  testRecall = []
  trainPrecission = []
  testPrecission = []
  trainFScore = []
  testFScore = []

  # Logging classification type
  if (classifyType == 0):
    logging(f"Naive bayes classification (with min max scaler)",
            logFile)
```

```
  elif (classifyType == 1):
    logging(f"SVM classification",
            logFile)
  elif (classifyType == 2):
    logging(f"Naive bayes classification (with quantile transformer)",
            logFile)

  # Cross-validation loop
  for foldId, (trainId, testId) in enumerate(skf.split(uniqueBaseNames,
baseNameLabels)):
    print(f"\nFold {foldId + 1}/5")

    # Log start time of training fold
    logging(f"Starting training fold {foldId+1}/5",
            logFile)

    trainBaseNames = uniqueBaseNames[trainId]
    testBaseNames = uniqueBaseNames[testId]
    # Get indices for train and test sets based on base names
    trainIndices = np.isin(baseNames, trainBaseNames)
    testIndices = np.isin(baseNames, testBaseNames)

    # Split data
    xLogMelTrain, xLogMelTest   = xLogMel[trainIndices], xLogMel[testIndices]
    xMfccTrain, xMfccTest       = xMfcc[trainIndices], xMfcc[testIndices]
    xChromaTrain, xChromaTest   = xChroma[trainIndices], xChroma[testIndices]
    xTonnetzTrain, xTonnetzTest = xTonnetz[trainIndices],
xTonnetz[testIndices]
    yTrain, yTest               = yLabel[trainIndices], yLabel[testIndices]

    # Standardize the features if chosen.
    if standardized:
      scaler = StandardScaler()
      xLogMelTrain = scaler.fit_transform(xLogMelTrain)
      xLogMelTest = scaler.transform(xLogMelTest)
      xMfccTrain = scaler.fit_transform(xMfccTrain)
      xMfccTest = scaler.transform(xMfccTest)
      xChromaTrain = scaler.fit_transform(xChromaTrain)
      xChromaTest = scaler.transform(xChromaTest)
      xTonnetzTrain = scaler.fit_transform(xTonnetzTrain)
      xTonnetzTest = scaler.transform(xTonnetzTest)


    # Combine features
    xTrain = np.hstack([xLogMelTrain, xMfccTrain, xChromaTrain,
xTonnetzTrain])
    xTest = np.hstack([xLogMelTest, xMfccTest, xChromaTest, xTonnetzTest])

    # Classify
    if (classifyType == 0):
      yTraining, yPrediction = nb_classify(xTrain, yTrain, xTest, logFile)
    elif (classifyType == 1):
      yTraining, yPrediction = svm_classify(xTrain, yTrain, xTest, logFile)
```

```
    # Evaluate the score
    print(f"Fold {foldId + 1} Results:")
    accuracyTrainFold = accuracy_score(yTrain, yTraining)
    recallTrainFold = recall_score(yTrain, yTraining, average="macro")
    precisionTrainFold = precision_score(yTrain, yTraining, average="macro",
zero_division=0)
    f1ScoreTrainFold = f1_score(yTrain,yTraining, average="macro")

    trainAccuracy.append(accuracyTrainFold)
    trainRecall.append(recallTrainFold)
    trainPrecission.append(precisionTrainFold)
    trainFScore.append(f1ScoreTrainFold)
    print(f"Train accuracy: {accuracyTrainFold:.6f}\nTrain Recall:
{recallTrainFold:.6f}\nTrain Precission: {precisionTrainFold:.6f}\nTrain F1
score: {f1ScoreTrainFold:.6f}")
    print("\n------------\n")

    accuracyTestFold = accuracy_score(yTest, yPrediction)
    recallTestFold = recall_score(yTest, yPrediction, average="macro")
    precisionTestFold = precision_score(yTest, yPrediction, average="macro",
zero_division=0)
    f1ScoreTestFold = f1_score(yTest,yPrediction, average="macro")

    testAccuracy.append(accuracyTestFold)
    testRecall.append(recallTestFold)
    testPrecission.append(precisionTestFold)
    testFScore.append(f1ScoreTestFold)
    print(f"Test accuracy: {accuracyTestFold:.6f}\nTest Recall:
{recallTestFold:.6f}\nTest Precission: {precisionTestFold:.6f}\nTest F1
score: {f1ScoreTestFold:.6f}")

    # Log end time of training fold
    logging(
f"""Ending training fold {foldId+1}/5
---Train Results---
Accuracy: {accuracyTrainFold:.6f}
Recall: {recallTrainFold:.6f}
Precission: {precisionTrainFold:.6f}
F1 score: {f1ScoreTrainFold:.6f}
---Test Results---
Accuracy: {accuracyTestFold:.6f}
Recall: {recallTestFold:.6f}
Precission: {precisionTestFold:.6f}
F1 score: {f1ScoreTestFold:.6f}""",
            logFile)

  # Final results
  print(f"Finished Training\n")

  print(f"Average Train Metrics:\n")
  print(f"Accuracy: {np.mean(trainAccuracy):.6f} ±
{np.std(trainAccuracy):.6f}\n")
```

```
  print(f"Recall: {np.mean(trainRecall):.6f} ± {np.std(trainRecall):.6f}\n")
  print(f"Precision: {np.mean(trainPrecission):.6f} ±
{np.std(trainPrecission):.6f}\n")
  print(f"F1 score: {np.mean(trainFScore):.6f} ±
{np.std(trainFScore):.6f}\n")

  print(f"Average Test Metrics:\n")
  print(f"Accuracy: {np.mean(testAccuracy):.6f} ±
{np.std(testAccuracy):.6f}\n")
  print(f"Recall: {np.mean(testRecall):.6f} ± {np.std(testRecall):.6f}\n")
  print(f"Precision: {np.mean(testPrecission):.6f} ±
{np.std(testPrecission):.6f}\n")
  print(f"F1 score: {np.mean(testFScore):.6f} ± {np.std(testFScore):.6f}\n")

   # Log end time of overall training
  logging(
f"""Finished Training
---Train Results---
Accuracy: {np.mean(trainAccuracy):.6f} ± {np.std(trainAccuracy):.6f}
Recall: {np.mean(trainRecall):.6f} ± {np.std(trainRecall):.6f}
Precission: {np.mean(trainPrecission):.6f} ± {np.std(trainPrecission):.6f}
F1 score: {np.mean(trainFScore):.6f} ± {np.std(trainFScore):.6f}
---Test Results---
Accuracy: {np.mean(testAccuracy):.6f} ± {np.std(testAccuracy):.6f}
Recall: {np.mean(testRecall):.6f} ± {np.std(testRecall):.6f}
Precission: {np.mean(testPrecission):.6f} ± {np.std(testPrecission):.6f}
F1 score: {np.mean(testFScore):.6f} ± {np.std(testFScore):.6f}""",
          logFile)



# Main Stuffs
print(f"Starting...")
featureDir = '/path/to/feature'
logDir = '/path/to/log'
logFile = os.path.join(logDir, 'NBSVM-ClassifyLog.txt')
xLogMel, xMfcc, xChroma, xTonnetz, yLabel, baseNames =
loadDataParallel(featureDir, maxWorkers=8)

# Encode labels
le = LabelEncoder()
yEncoded = le.fit_transform(yLabel)



"""NB without standardization"""
# Classify audio
classify(xLogMel,xMfcc,xChroma,xTonnetz,yEncoded,baseNames,logFile,classifyTy
pe=0,standardized=False)


"""SVM with standardization"""
# Classify audio
```

```
classify(xLogMel,xMfcc,xChroma,xTonnetz,yEncoded,baseNames,logFile,classifyTy
pe=1,standardized=True)
```

classify(CNN).py

```
import numpy as np
import os
import time
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.model_selection import StratifiedKFold, train_test_split
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score, balanced_accuracy_score,
precision_score, recall_score, f1_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.utils.class_weight import compute_class_weight
from concurrent.futures import ThreadPoolExecutor
import pkbar


def loadDataParallel(mainDir, maxWorkers=None):
  xLogMel, xMfcc, xChroma, xTonnetz, y, baseNames = [], [], [], [], [], []

  # First collect all valid file paths
  tasks = []
  for classLabel in os.listdir(mainDir):
      classDirectory = os.path.join(mainDir, classLabel)
      if not os.path.isdir(classDirectory):
          continue

      logMelDir = os.path.join(classDirectory, 'logmel')
      mfccDir = os.path.join(classDirectory, 'mfcc')
      chromaDir = os.path.join(classDirectory, 'chroma')
      tonnetzDir = os.path.join(classDirectory, 'tonnetz')

      if not all(os.path.isdir(d) for d in [logMelDir, mfccDir, chromaDir,
tonnetzDir]):
          continue

      sampleFiles = [f for f in os.listdir(mfccDir) if f.endswith('.npy')]

      # Group files by base name (without segment index)
      file_groups = {}
      for fileName in sampleFiles:
          # Extract base name (remove segment index)
          if '[' in fileName and fileName.endswith('].npy'):
              base_name = fileName.split('[')[0]
          else:
              base_name = os.path.splitext(fileName)[0]

          if base_name not in file_groups:
```

```
                file_groups[base_name] = []
            file_groups[base_name].append(fileName)

        # Create tasks per base name
        for base_name, files in file_groups.items():
            for fileName in files:
                paths = (
                    os.path.join(logMelDir, fileName),
                    os.path.join(mfccDir, fileName),
                    os.path.join(chromaDir, fileName),
                    os.path.join(tonnetzDir, fileName)
                )
                if all(os.path.isfile(p) for p in paths):
                    tasks.append((paths, classLabel, base_name))

    # Parallel loading function
    def loadFiles(task):
        paths, classLabel, base_name = task
        try:
            return (
                np.load(paths[0]).flatten(),
                np.load(paths[1]).flatten(),
                np.load(paths[2]).flatten(),
                np.load(paths[3]).flatten(),
                classLabel,
                base_name
            )
        except Exception as e:
            print(f"Error loading {paths[0]}: {str(e)}")
            return None

    # Process tasks in parallel
    with ThreadPoolExecutor(max_workers=maxWorkers) as executor:
        results = executor.map(loadFiles, tasks)

        for result in results:
            if result is not None:
                logMel, mfcc, chroma, tonnetz, label, base_name = result
                xLogMel.append(logMel)
                xMfcc.append(mfcc)
                xChroma.append(chroma)
                xTonnetz.append(tonnetz)
                y.append(label)
                baseNames.append(base_name)

    return np.array(xLogMel), np.array(xMfcc), np.array(xChroma),
np.array(xTonnetz), np.array(y), np.array(baseNames)


# Record time to log file
def logging(logText, logFile):
    with open(logFile, 'a') as file:
        file.write(f"{time.time()}:\n")
        file.write(f"{logText}\n\n")
```

```
class BirdSoundCNN(nn.Module):
  def __init__(self, input_dim, num_classes):
    super(BirdSoundCNN, self).__init__()

    # 1D Convolutional layers
    self.conv_block = nn.Sequential(
      # First conv block
      nn.Conv1d(1, 64, kernel_size=7, padding=3),
      nn.BatchNorm1d(64),
      nn.ReLU(),
      nn.Conv1d(64, 64, kernel_size=7, padding=3),
      nn.BatchNorm1d(64),
      nn.ReLU(),
      nn.MaxPool1d(2),
      nn.Dropout1d(0.2),

      # Second conv block
      nn.Conv1d(64, 128, kernel_size=5, padding=2),
      nn.BatchNorm1d(128),
      nn.ReLU(),
      nn.Conv1d(128, 128, kernel_size=5, padding=2),
      nn.BatchNorm1d(128),
      nn.ReLU(),
      nn.MaxPool1d(2),
      nn.Dropout1d(0.2),

      # Third conv block
      nn.Conv1d(128, 256, kernel_size=3, padding=1),
      nn.BatchNorm1d(256),
      nn.ReLU(),
      nn.Conv1d(256, 256, kernel_size=3, padding=1),
      nn.BatchNorm1d(256),
      nn.ReLU(),
      nn.MaxPool1d(2),
      nn.Dropout1d(0.2),

      # Fourth conv block
      nn.Conv1d(256, 512, kernel_size=3, padding=1),
      nn.BatchNorm1d(512),
      nn.ReLU(),
      nn.Conv1d(512, 512, kernel_size=3, padding=1),
      nn.BatchNorm1d(512),
      nn.ReLU(),

      # Global pooling
      nn.AdaptiveAvgPool1d(1),
      nn.Flatten()
    )

    # Attention mechanism for feature importance
    self.attention = nn.Sequential(
      nn.Linear(512, 128),
```

```
      nn.ReLU(),
      nn.Linear(128, 512),
      nn.Sigmoid()
    )

    # Fully connected layers
    self.fc_block = nn.Sequential(
      nn.Linear(512, 512),
      nn.BatchNorm1d(512),
      nn.ReLU(),
      nn.Dropout(0.5),

      nn.Linear(512, 256),
      nn.BatchNorm1d(256),
      nn.ReLU(),
      nn.Dropout(0.5),

      nn.Linear(256, 128),
      nn.BatchNorm1d(128),
      nn.ReLU(),
      nn.Dropout(0.5),

      nn.Linear(128, num_classes)
    )

  def forward(self, x):
    # Add channel dimension (batch_size, 1, input_dim)
    x = x.unsqueeze(1)

    # Extract features through conv layers
    features = self.conv_block(x)

    # Apply attention
    attention_weights = self.attention(features)
    features = features * attention_weights

    # Classification
    return self.fc_block(features)


# Classify based on features extracted.
def classify(featureDir, logFile, modelDir):
  # Log start time of collecting files
  logging("Starting collecting files",
          logFile)

  # Retrieve features from disk
  xLogMel, xMfcc, xChroma, xTonnetz, yLabel, baseNames =
loadDataParallel(featureDir, maxWorkers=8)
  # Encode labels
  le = LabelEncoder()
  yLabel = le.fit_transform(yLabel)

  # Log end time of preparing data
```

```
  logging("Starting preparing data",
          logFile)

  # Get unique base names and their labels
  uniqueBaseNames = np.unique(baseNames)
  baseNameLabels = np.array([yLabel[np.where(baseNames == base)[0][0]] for
base in uniqueBaseNames])

  # Initialize cross-validation
  skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
  trainAccuracy = []
  testAccuracy = []
  trainRecall = []
  testRecall = []
  trainPrecission = []
  testPrecission = []
  trainFScore = []
  testFScore = []

  # Cross-validation loop
  for foldId, (trainId, testId) in enumerate(skf.split(uniqueBaseNames,
baseNameLabels)):
    print(f"Fold {foldId + 1}/5")

    trainBaseNames = uniqueBaseNames[trainId]
    testBaseNames = uniqueBaseNames[testId]
    # Get indices for train and test sets based on base names
    trainIndices = np.isin(baseNames, trainBaseNames)
    testIndices = np.isin(baseNames, testBaseNames)

    # Split data
    xLogMelTrain, xLogMelTest   =   xLogMel[trainIndices],
xLogMel[testIndices]
    xMfccTrain, xMfccTest       =   xMfcc[trainIndices], xMfcc[testIndices]
    xChromaTrain, xChromaTest   =   xChroma[trainIndices],
xChroma[testIndices]
    xTonnetzTrain, xTonnetzTest =   xTonnetz[trainIndices],
xTonnetz[testIndices]
    yTrain, yTest               =   yLabel[trainIndices], yLabel[testIndices]

    # Standardized each of the features
    scaler = StandardScaler()
    xLogMelTrain = scaler.fit_transform(xLogMelTrain)
    xLogMelTest = scaler.transform(xLogMelTest)
    xMfccTrain = scaler.fit_transform(xMfccTrain)
    xMfccTest = scaler.transform(xMfccTest)
    xChromaTrain = scaler.fit_transform(xChromaTrain)
    xChromaTest = scaler.transform(xChromaTest)
    xTonnetzTrain = scaler.fit_transform(xTonnetzTrain)
    xTonnetzTest = scaler.transform(xTonnetzTest)

    # Combine features
    xTrain = np.hstack([xLogMelTrain, xMfccTrain, xChromaTrain,
xTonnetzTrain])
```

```
    xTest = np.hstack([xLogMelTest, xMfccTest, xChromaTest, xTonnetzTest])

    # Delete unnecessary variables for freeing up memory
    del xLogMelTrain, xLogMelTest
    del xMfccTrain, xMfccTest
    del xChromaTrain, xChromaTest
    del xTonnetzTrain, xTonnetzTest

    # Create separate datasets for training, validation and testing
    XTrain = [torch.tensor(t, dtype=torch.float32) for t in xTrain]
    yTrain = torch.tensor(yTrain, dtype=torch.long)
    trainDataset = TensorDataset(torch.stack(XTrain), yTrain)

    XTest = [torch.tensor(t, dtype=torch.float32) for t in xTest]
    yTest = torch.tensor(yTest, dtype=torch.long)
    testDataset = TensorDataset(torch.stack(XTest), yTest)

    # Get the dimension of the feature input
    dimension = xTrain.shape[1]

    # Calculate class weights
    computeClassWeight = compute_class_weight(class_weight = "balanced",
                                              classes =
np.unique(yLabel[trainIndices]),
                                              y = yTrain.numpy())
    classWeights = torch.tensor(computeClassWeight, dtype=torch.float32)

    # Delete unnecessary variables
    del xTrain, XTrain, yTrain, xTest, XTest, yTest

    # Create data loaders
    trainLoader = DataLoader(trainDataset, batch_size=10, shuffle=True,
num_workers=8, pin_memory=True)
    testLoader = DataLoader(testDataset, batch_size=10, shuffle=False)

    # Initialize model with input channels equal to the number of PCA
components
    numClasses = len(np.unique(yLabel))
    device = 'cuda'
    numEpoch = 500
    model = BirdSoundCNN(input_dim=dimension,
num_classes=numClasses).to(torch.device(device))
    classWeights = classWeights.to(device)

    # Training setup
    optimizer = optim.RAdam(params=model.parameters(), lr=0.001,
weight_decay=0.01, decoupled_weight_decay=True)
    criterion = nn.CrossEntropyLoss(weight=classWeights)

    # Log start time of training epoch
    logging(f"Starting training epoch {foldId+1}/5",
            logFile)

    # Training loop
```

```
    bestValAccuracy = 0
    patience = 20
    patience_counter = 0
    for epoch in range(numEpoch):
      kbar = pkbar.Kbar(target=len(trainLoader), epoch=epoch,
num_epochs=numEpoch, width=25)
      trainingPred = []
      trainingTarget = []
      model.train()

      for batchId, (inputs, targets) in enumerate(trainLoader):
        inputs, targets = inputs.to(device), targets.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, targets)
        loss.backward()

        # Gradient clipping to prevent exploding gradients
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)

        optimizer.step()

        # Get the loss and predictions
        kbar.update(batchId, values=[("loss", loss.item())])
        preds = torch.argmax(outputs, dim=1)
        trainingPred.extend(preds.cpu().numpy())
        trainingTarget.extend(targets.cpu().numpy())

      # Calculate the epoch metrics
      trainF1 = accuracy_score(trainingTarget, trainingPred)
      print(f" F1 score: {trainF1:.5f}")

    # Log end time of training epoch
    logging(f"Ending training epoch {foldId+1}/5",
            logFile)

    # Final training metrics
    model.eval()
    trainPreds = []
    trainTargets = []
    with torch.inference_mode():
      for inputs, targets in trainLoader:
        inputs = inputs.to(device)
        outputs = model(inputs)
        preds = torch.argmax(outputs, dim=1)
        trainPreds.extend(preds.cpu().numpy())
        trainTargets.extend(targets.cpu().numpy())

    # Evaluate the score
    print(f"Fold {foldId + 1} Results:")
    accuracyTrainFold = accuracy_score(trainTargets, trainPreds)
    recallTrainFold = recall_score(trainTargets, trainPreds, average="macro")
    precisionTrainFold = precision_score(trainTargets, trainPreds,
average="macro", zero_division=0)
```

```
    f1ScoreTrainFold = f1_score(trainTargets,trainPreds, average="macro")

    trainAccuracy.append(accuracyTrainFold)
    trainRecall.append(recallTrainFold)
    trainPrecission.append(precisionTrainFold)
    trainFScore.append(f1ScoreTrainFold)
    print(f"Train accuracy: {accuracyTrainFold:.6f}\nTrain Recall:
{recallTrainFold:.6f}\nTrain Precission: {precisionTrainFold:.6f}\nTrain F1
score: {f1ScoreTrainFold:.6f}")
    print("\n------------\n")



    # Evaluation on testing set
    model.eval()
    testPreds = []
    testTargets = []
    with torch.inference_mode():
      for inputs, targets in testLoader:
        inputs = inputs.to(device)
        outputs = model(inputs)
        preds = torch.argmax(outputs, dim=1)
        testPreds.extend(preds.cpu().numpy())
        testTargets.extend(targets.cpu().numpy())

    # Evaluate the score
    print(f"Fold {foldId + 1} Results:")
    accuracyTestFold = accuracy_score(testTargets, testPreds)
    recallTestFold = recall_score(testTargets, testPreds, average="macro")
    precisionTestFold = precision_score(testTargets, testPreds,
average="macro", zero_division=0)
    f1ScoreTestFold = f1_score(testTargets,testPreds, average="macro")

    testAccuracy.append(accuracyTestFold)
    testRecall.append(recallTestFold)
    testPrecission.append(precisionTestFold)
    testFScore.append(f1ScoreTestFold)
    print(f"Test accuracy: {accuracyTestFold:.6f}\nTest Recall:
{recallTestFold:.6f}\nTest Precission: {precisionTestFold:.6f}\nTest F1
score: {f1ScoreTestFold:.6f}")
    print("\n------------\n")

    # Save the model weights
    modelFoldPth = f"model_fold_{foldId}.pth"
    torch.save(model.state_dict(), os.path.join(modelDir, modelFoldPth))
    print(f"Saved model fold {foldId + 1}.")

     # Log end time of training fold
    logging(
f"""Ending training fold {foldId+1}/5 and saving model
---Train Results---
Accuracy: {accuracyTrainFold:.6f}
Recall: {recallTrainFold:.6f}
Precission: {precisionTrainFold:.6f}
```

```
F1 score: {f1ScoreTrainFold:.6f}
---Test Results---
Accuracy: {accuracyTestFold:.6f}
Recall: {recallTestFold:.6f}
Precission: {precisionTestFold:.6f}
F1 score: {f1ScoreTestFold:.6f}""",
            logFile)


  # Final results
  print(f"Finished Training\n")

  print(f"Average Train Metrics:\n")
  print(f"Accuracy: {np.mean(trainAccuracy):.6f} ±
{np.std(trainAccuracy):.6f}\n")
  print(f"Recall: {np.mean(trainRecall):.6f} ± {np.std(trainRecall):.6f}\n")
  print(f"Precision: {np.mean(trainPrecission):.6f} ±
{np.std(trainPrecission):.6f}\n")
  print(f"F1 score: {np.mean(trainFScore):.6f} ±
{np.std(trainFScore):.6f}\n")

  print(f"Average Test Metrics:\n")
  print(f"Accuracy: {np.mean(testAccuracy):.6f} ±
{np.std(testAccuracy):.6f}\n")
  print(f"Recall: {np.mean(testRecall):.6f} ± {np.std(testRecall):.6f}\n")
  print(f"Precision: {np.mean(testPrecission):.6f} ±
{np.std(testPrecission):.6f}\n")
  print(f"F1 score: {np.mean(testFScore):.6f} ± {np.std(testFScore):.6f}\n")

   # Log end time of overall training
  logging(
f"""Finished Training
---Train Results---
Accuracy: {np.mean(trainAccuracy):.6f} ± {np.std(trainAccuracy):.6f}
Recall: {np.mean(trainRecall):.6f} ± {np.std(trainRecall):.6f}
Precission: {np.mean(trainPrecission):.6f} ± {np.std(trainPrecission):.6f}
F1 score: {np.mean(trainFScore):.6f} ± {np.std(trainFScore):.6f}
---Test Results---
Accuracy: {np.mean(testAccuracy):.6f} ± {np.std(testAccuracy):.6f}
Recall: {np.mean(testRecall):.6f} ± {np.std(testRecall):.6f}
Precission: {np.mean(testPrecission):.6f} ± {np.std(testPrecission):.6f}
F1 score: {np.mean(testFScore):.6f} ± {np.std(testFScore):.6f}""",
          logFile)


# Main function
print(f"Starting...")

featureDir = '/path/to/feature'
logDir = '/path/to/log'
modelDir = '/path/to/model'
logFile = os.path.join(logDir, 'CNNClassifyLog.txt')
""" mfccComponents = 0.95
chromaComponents = 0.95
```

```
tonnetzComponents = 0.95 """

# Classify audio

classify(featureDir, logFile, modelDir)
```

server/featureExtractor.py

```
import librosa
import numpy as np
import noisereduce
import pyloudnorm
from scipy.signal import hilbert
import tempfile
import os

# Consistent parameters with React Native
SAMPLE_RATE = 32000
N_FFT = 1024
HOP_LENGTH = 256
N_MFCC = 20
N_CHROMA = 12
WINDOW = 'hamming'

def remove_silence (
    sound,
    rate,
    smoothWindow = 0.05,
    silencePercentile = 5,
    minSilenceDuration = 0.2,
    bufferDuration = 0.1
):
  # Compute amplitude envolope using Hilbert transform
  analyticSignal = hilbert(sound)
  amplitudeEnvelope = np.abs(analyticSignal)

  # Smooth the envelope with a moving average
  smoothWindowSize = int(rate * smoothWindow)
  smoothFilter = np.ones(smoothWindowSize) / smoothWindowSize
  smoothedEnvelope = np.convolve(
    amplitudeEnvelope,
    smoothFilter,
    mode='same'
    )

  # Calculate threshold based on average of smoothed envelope
  threshold = np.percentile(smoothedEnvelope, silencePercentile)

  # Identify silent segments where envelope is below the threshold
  mask = smoothedEnvelope < threshold

  # Find transitions between silent and non-silent segments
  changes = np.diff(mask.astype(int))
  starts = np.where(changes == 1)[0] + 1
```

```
    ends = np.where(changes == -1)[0] + 1

    # Handle silence at the begging or end
    if mask[0]:
      starts = np.insert(starts, 0, 0)
    if mask[-1]:
      ends = np.append(ends, len(mask))

    silentRegions = list(zip(starts, ends))

    # Apply minimum silence durationn
    minSilenceSample = int(minSilenceDuration * rate)
    validSilence = []
    for start, end in silentRegions:
      if (end - start >= minSilenceSample): {
        validSilence.append((start, end))
      }

    # Determine non-silent segments by inverting silent segments
    nonSilent = []
    prevEnd = 0
    for start, end in validSilence:
      nonSilent.append((prevEnd, start))
      prevEnd = end
    nonSilent.append((prevEnd, len(sound)))

    # Add buffer around non-silent segments
    bufferSamples = int(bufferDuration * rate)
    segments = []
    for start, end in nonSilent:
      segmentStart = max(start - bufferSamples, 0)
      segmentEnd = min(end + bufferSamples, len(sound))
      segments.append(sound[segmentStart:segmentEnd])

    # Concatinate the segments and handle cases where there aren't any silence
    if not segments:
      processedAudio = sound
    else:
      processedAudio = np.concatenate(segments)

    return processedAudio

  def extract_features(audio_path):
    """Extract features from audio file"""
    try:
      # Load audio
      sound, rate = librosa.load(audio_path, sr=SAMPLE_RATE, mono=True)

      # Normalize to -3dB
      sound = pyloudnorm.normalize.peak(sound, -3.0)

      # Apply noise reduction
      sound = noisereduce.reduce_noise(
        y=sound, sr=rate, stationary=False, prop_decrease=1.0
```

```
    )

    # Remove silence
    sound = remove_silence(sound, rate)

    # Extract 3-second segments
    segment_samples = 3 * SAMPLE_RATE
    if len(sound) < segment_samples:
      # Pad with zeros if too short
      sound = np.pad(sound, (0, segment_samples - len(sound)))

    # Use first 3 seconds
    segment = sound[:segment_samples]

    # Extract features with consistent parameters
    features = {}

    # Mel Spectrogram
    mel_spec = librosa.feature.melspectrogram(
      y=segment, sr=rate,
      n_fft=N_FFT, hop_length=HOP_LENGTH,
      window=WINDOW
    )
    features['logmel'] = librosa.power_to_db(mel_spec, ref=np.max)

    # MFCC
    features['mfcc'] = librosa.feature.mfcc(
      y=segment, sr=rate, n_mfcc=N_MFCC
    )

    # Chroma
    features['chroma'] = librosa.feature.chroma_cqt(
      y=segment, sr=rate, n_chroma=N_CHROMA
    )

    # Tonnetz
    features['tonnetz'] = librosa.feature.tonnetz(chroma=features['chroma'])

    # Flatten features
    flattened = {
      'logmel': features['logmel'].flatten().tolist(),
      'mfcc': features['mfcc'].flatten().tolist(),
      'chroma': features['chroma'].flatten().tolist(),
      'tonnetz': features['tonnetz'].flatten().tolist()
    }

    return flattened

  except Exception as e:
    raise Exception(f"Feature extraction failed: {str(e)}")
```

server/main.py

```
from fastapi import FastAPI, UploadFile, File, HTTPException
```

```
from fastapi.middleware.cors import CORSMiddleware
import tempfile
import os
from feature_extractor import extract_features

app = FastAPI(title="Bird Recognition Feature Extraction API")

# Enable CORS for mobile app
app.add_middleware(
  CORSMiddleware,
  allow_origins=["*"],  # Configure this properly for production
  allow_credentials=True,
  allow_methods=["*"],
  allow_headers=["*"],
)

@app.post("/extract-features")
async def extract_audio_features(audio: UploadFile = File(...)):
  """Extract features from uploaded audio file"""
  try:
    # Save uploaded file temporarily
    with tempfile.NamedTemporaryFile(delete=False, suffix='.m4a') as
tmp_file:
      content = await audio.read()
      tmp_file.write(content)
      tmp_file_path = tmp_file.name

    # Extract features
    features = extract_features(tmp_file_path)

    # Clean up temporary file
    os.unlink(tmp_file_path)

    return {
      "success": True,
      "features": features,
      "message": "Features extracted successfully"
    }

  except Exception as e:
    # Clean up on error
    if 'tmp_file_path' in locals():
      try:
        os.unlink(tmp_file_path)
      except:
        pass

    raise HTTPException(status_code=500, detail=str(e))

@app.get("/health")
async def health_check():
  """Health check endpoint"""
  return {"status": "healthy", "message": "Feature extraction service is
running"}
```

```
if __name__ == "__main__":
  import uvicorn
  uvicorn.run(app, host="0.0.0.0", port=os.getenv("PORT", default=8000))
```

reactnative/app/index.tsx

```
import React from "react";
import "@/global.css";
import HomePage from "@/app/HomePage";
import { createNativeStackNavigator } from "@react-navigation/native-stack";
import PastRecordsPage from "@/app/PastRecords";
import ResultPage from "@/app/ResultPage";
import FeedbackPage from "@/app/FeedbackPage";

const Stack = createNativeStackNavigator();

export default function App() {
  return (
    <Stack.Navigator initialRouteName="homePage">
      <Stack.Screen name='homePage' component={HomePage} options={{
headerShown: false }}/>
      <Stack.Screen name='resultPage' component={ResultPage} options={{
headerShown: false }}/>
      <Stack.Screen name='pastRecords' component={PastRecordsPage} options={{
headerShown: false }}/>
      <Stack.Screen name='feedbackPage' component={FeedbackPage} options={{
headerShown: false }}/>
    </Stack.Navigator>
  );

}
```

reactnative/app/HomePage.tsx

```
import "@/global.css";
import { useFonts } from 'expo-font';
import React, { useEffect, useState } from "react";
import { SafeAreaView } from "@/components/ui/safe-area-view";
import { GluestackUIProvider } from "@/components/ui/gluestack-ui-provider";
import { Text } from "@/components/ui/text";
import { Center } from "@/components/ui/center";
import { MicIcon } from "lucide-react-native";
import { Button, ButtonIcon } from "@/components/ui/button";
import DodoIcon from "@/components/custom/DodoIcon";
import { WaveBackground } from "@/components/custom/WaveBackground";
import { BottomButton } from "@/components/custom/BottomButton";
import { Icon } from "@/components/ui/icon";
import { Alert, PermissionsAndroid, Platform, ActivityIndicator } from
"react-native";
import StopIcon from "@/components/custom/StopIcon";
import { AudioClassifier } from "@/src/audio/AudioClassifier";
import { AudioRecorder } from "@/src/audio/AudioRecorder";
import * as FileSystem from 'expo-file-system';
```

```
const HomePage = ({navigation}:{navigation:any}) => {
  //Create an instance of AudioClassifier
  const audioClassifier = new AudioClassifier();
  const [audioRecorder] = useState(new AudioRecorder());

  // Light/dark mode
  const [colorMode, setColorMode] = useState<"dark" | "light">("light");
  // For fonts
  useFonts({
    'LilyScriptOne': require('@/assets/fonts/LilyScriptOne-Regular.ttf')
  });

  // Asking for permission on loading the page
  useEffect(() => {
    requestPermissions();
    // Initialize the audio classifier
    audioClassifier.initialize().catch(error => {
      console.error("Failed to initialize audio classifier:", error);
    });
    // Cleanup
    return () => {
      audioRecorder.cleanup();
      audioClassifier.dispose();
    };
  }, []);


  // Request microphone permissions
  const requestPermissions = async () => {
    if (Platform.OS === 'android') {
      try {
        const grants = await PermissionsAndroid.requestMultiple([
          PermissionsAndroid.PERMISSIONS.RECORD_AUDIO,
        ]);

        if (
          grants['android.permission.RECORD_AUDIO'] !==
PermissionsAndroid.RESULTS.GRANTED
        ) {
          Alert.alert(
            'Permissions Required',
            'Please grant all permissions to use this app',
            [{text: 'OK'}]
          );
        }
      } catch (error) {
        console.log(error);
      }
    }
  };

  // Audio recording and processing states
  const [isRecording, setRecording] = useState(false);
```

```
  const [isProcessing, setProcessing] = useState(false);

  const toggleRecording = async() => {
    // Don't allow action if currently processing
    if (isProcessing) return;

    if (!isRecording) {
      try {
        // Start recording
        await audioRecorder.startRecording();
        setRecording(true);
      } catch (error) {
        console.error("Failed to start recording:", error);
        Alert.alert(
          'Recording Error',
          'Failed to start recording. Please try again later.',
          [{text: 'OK'}]
        );
      }
    } else {
      try {
        // Stop recording and start processing
        setRecording(false);
        setProcessing(true);

        const audioPath = await audioRecorder.stopRecording();
        console.log("Recording saved to:", audioPath);

        // Classify the recorded audio
        try {
          const encodedLabel = await
audioClassifier.classifyAudio(audioPath);

          // Navigate to the results page with the classification result
          navigation.navigate('resultPage', {
            encodedLabel: encodedLabel,
            fileName: audioPath.split('/').pop()
          });

          // Reset processing state after navigation
          setProcessing(false);
        } catch (error) {
          // Classification failed - reset to idle state
          setProcessing(false);

          Alert.alert(
            'Classification Error',
            'Failed to classify the audio. Please try again later.',
            [{text: 'OK'}]
          );
          console.log('Classification error:', error);
        }

      } catch (error) {
```

```
      // Recording stop failed - reset to idle state
      setRecording(false);
      setProcessing(false);

      console.error("Failed to stop recording:", error);
      Alert.alert(
        'Recording Error',
        'Failed to stop recording. Please try again later.',
        [{text: 'OK'}]
      );
      console.error('Stop recording error:', error);
    }
  }
}

// Determine button content based on state
const getButtonContent = () => {
  if (isProcessing) {
    return {
      icon: (
        <ActivityIndicator
          size="large"
          color="#a3b18a"
          style={{ width: 80, height: 80 }}
        />
      ),
      text: "Processing...",
      buttonStyle: "bg-[#3a5a40] h-[130] w-[130] rounded-full p-3.5"
    };
  } else if (isRecording) {
    return {
      icon: (
        <ButtonIcon
          as={StopIcon}
          stroke="#3a5a40"
          className="h-[80] w-[80]"
        />
      ),
      text: "Stop Recording",
      buttonStyle: "bg-[#a3b18a] h-[130] w-[130] rounded-full p-3.5"
    };
  } else {
    return {
      icon: (
        <ButtonIcon
          as={MicIcon}
          stroke="#a3b18a"
          className="h-[80] w-[80]"
        />
      ),
      text: "Record Now",
      buttonStyle: "bg-[#3a5a40] h-[130] w-[130] rounded-full p-3.5"
    };
  }
```

```
  };

  const buttonContent = getButtonContent();

  return (
    <SafeAreaView className="flex-1 bg-[#dad7cd] justify-center items-
center">
      <GluestackUIProvider mode={colorMode}>
        {/*Header*/}
        <Center className="bg-[#a3b18a] p-5 h-20 w-full rounded-b-[30px]">
          <Icon as={DodoIcon} className="h-[50] w-[50] absolute left-10 top-
0" />
          <Text size="4xl" style={{ color: '#344e41', fontFamily:
'LilyScriptOne-Regular' }}>Bird Recognition App</Text>
        </Center>

        {/*Center button*/}
        <Center className="flex-1 h-70 justify-center items-center">
          <Button
            className={buttonContent.buttonStyle}
            onPress={toggleRecording}
            disabled={isProcessing}
          >
            {buttonContent.icon}
          </Button>
          <Text size="3xl" className="mt-1 text-
[#344e41]">{buttonContent.text}</Text>
        </Center>

        {/* Bottom button */}
        <BottomButton navigation={navigation} />
        <WaveBackground />

      </GluestackUIProvider>
    </SafeAreaView>
  );
};


export default HomePage;
```

reactnative/app/ResultPage.tsx

```
import "@/global.css";
import { useFonts } from 'expo-font';
import React, { useState } from "react";
import { SafeAreaView } from "@/components/ui/safe-area-view";
import { GluestackUIProvider } from "@/components/ui/gluestack-ui-provider";
import { Text } from "@/components/ui/text";
import { Center } from "@/components/ui/center";
import { ArrowLeft } from "lucide-react-native";
import { Button, ButtonIcon, ButtonText } from "@/components/ui/button";
import { WaveBackground } from "@/components/custom/WaveBackground";
import { BottomButton } from "@/components/custom/BottomButton";
```

```
import birdData from "@/assets/metadata/bird_label.json"
import { HStack } from "@/components/ui/hstack";;
import { Image } from "@/components/ui/image"
import imageIndex from "@/components/custom/BirdImageIndex";
import { Heading } from "@/components/ui/heading";

const ResultPage = ({route, navigation}:{route:any, navigation:any}) => {
  // Result
  const {encodedLabel, fileName} = route.params;
  const birdInfo = birdData.find(item => item.encodedLabel === encodedLabel);
  const birdSpecies = birdInfo?.species
  const birdImage = birdInfo ? (imageIndex as Record<string,
any>)[birdInfo.species] : null;

  // Light/dark mode
  const [colorMode, setColorMode] = useState<"dark" | "light">("light");
  // For fonts
  useFonts({
    'LilyScriptOne': require('@/assets/fonts/LilyScriptOne-Regular.ttf')
  });

  return (
    <SafeAreaView className="flex-1 bg-[#dad7cd] justify-center items-
center">
      <GluestackUIProvider mode={colorMode}>
        <Center className="bg-[#a3b18a] h-20 w-full p-0 m-0">
          {/* Heading */}
          <HStack className="flex-1 justify-between h-full w-full">
            <Center className="ml-5">
              <Button
                className='h-[40] w-[40] bg-[#3a5a40] rounded-full p-3.5'
                onPress={() => {
                  navigation.navigate("homePage");
                }}
              >
                <ButtonIcon size="lg" as={ArrowLeft} stroke={ "#a3b18a" }
className="h-[25] w-[25]"/>
              </Button>
            </Center>
            <Center className="w-100">
              <Text size="6xl" style={{ color: '#344e41', fontFamily:
'LilyScriptOne-Regular' }}>Result</Text>
            </Center>
            <Center className="mr-4">
              <Button
                size="lg"
                variant="solid"
                className='bg-[#3a5a40] rounded-2xl'
                onPress={() => {
                  navigation.navigate('feedbackPage', {fileName: fileName});
                }}
              >
                <ButtonText size="sm" className="text-
[#a3b18a]">Feedback</ButtonText>
```

```
              </Button>
            </Center>
          </HStack>
        </Center>

        {/*Center Item*/}
        <Center className="flex-1 h-70 justify-center items-center">
            <Image className="w-[304px] h-[171px] rounded-2xl mb-2"
source={birdImage} alt="Bird Image"/>
            <Center className="w-full px-6 mb-3">
              <Heading size="2xl" className="text-center text-[#344e41] mb-
3">{birdInfo?.species}</Heading>
            </Center>
            <Text size="2xl" className="text-center" style={{ color:
'#344e41', fontFamily: 'LilyScriptOne-Regular' }}>Credit</Text>
            <Text size="lg" className="text-center text-
[#344e41]">{birdInfo?.attribute}</Text>
          </Center>

        {/* <HStack className="flex-1 justify-between px-6 pt-4 pb-2 bg-black
h-5">

        </HStack> */}



        {/* Bottom button */}
        <BottomButton navigation={navigation} />
        <WaveBackground />

      </GluestackUIProvider>
    </SafeAreaView>
  );
};

export default ResultPage;
```

reactnative/app/PastRecords.tsx

```
import "@/global.css";
import { useFonts } from 'expo-font';
import React, { useEffect, useState } from "react";
import { SafeAreaView } from "@/components/ui/safe-area-view";
import { GluestackUIProvider } from "@/components/ui/gluestack-ui-provider";
import { Text } from "@/components/ui/text";
import { Center } from "@/components/ui/center";
import { WaveBackground } from "@/components/custom/WaveBackground";
import { BottomButton } from "@/components/custom/BottomButton";
import * as FileSystem from "expo-file-system";
import { AudioFileCard } from "@/components/custom/AudioFileCard";
import { AudioRecord } from "@/src/types/AudioRecord";
import { Alert, FlatList } from "react-native";
import { Box } from "@/components/ui/box";
import { Button, ButtonIcon } from "@/components/ui/button";
```

```
import { ArrowLeft } from "lucide-react-native";
import { HStack } from "@/components/ui/hstack";
import { AudioClassifier } from "@/src/audio/AudioClassifier";

const PastRecordsPage = ({navigation}:{navigation:any}) => {
  // Light/dark mode
  const [colorMode, setColorMode] = useState<"dark" | "light">("light");
  // For fonts
  useFonts({
    'LilyScriptOne': require('@/assets/fonts/LilyScriptOne-Regular.ttf')
  });

  const [records, setRecords] = useState<AudioRecord[]>([]);
  const [audioClassifier] = useState(new AudioClassifier());
  const [isClassifierReady, setIsClassifierReady] = useState(false);

  // Audio directory
  const audioDir = FileSystem.documentDirectory + 'birdappAudio/';

  // Initialize audio classifier
  useEffect(() => {
    const initializeClassifier = async () => {
      try {
        await audioClassifier.initialize();
        setIsClassifierReady(true);
        console.log("Audio classifier initialized in PastRecords");
      } catch (error) {
        console.error("Failed to initialize audio classifier:", error);
        Alert.alert(
          'Initialization Error',
          'Failed to initialize audio classifier. Some features may not
work.',
          [{text: 'OK'}]
        );
      }
    };

    initializeClassifier();

    // Cleanup
    return () => {
      audioClassifier.dispose();
    };
  }, []);

  useEffect(() => {
    const loadAudioFiles = async () => {
      try {
        await FileSystem.makeDirectoryAsync(audioDir, { intermediates: true
});
        const files = await FileSystem.readDirectoryAsync(audioDir);
        const recordsWithTimestamps = [];

        // Collect files with timestamps
```

```
        for (const fileName of files) {
          const uri = `${audioDir}${fileName}`;
          try {
            const fileInfo = await FileSystem.getInfoAsync(uri);
            if (fileInfo.exists && fileInfo.modificationTime) {
              const date = new Date(fileInfo.modificationTime * 1000);
              const localTime = date.toLocaleDateString();
              const hours = date.getHours();
              const minutes = date.getMinutes();
              const seconds = date.getSeconds();
              const formattedTime = `${localTime} ${hours}:${minutes < 10 ?
'0' : ''}${minutes}:${seconds < 10 ? '0' : ''}${seconds}`;

              recordsWithTimestamps.push({
                title: fileName,
                date: formattedTime,
                uri: uri,
                timestamp: fileInfo.modificationTime // Store timestamp for
sorting
              });
            }
          } catch (fileError) {
            console.error(`Error processing ${fileName}:`, fileError);
          }
        }

        // Sort by timestamp descending (newest first)
        recordsWithTimestamps.sort((a, b) => b.timestamp - a.timestamp);

        // Extract sorted records (remove temporary timestamp)
        const sortedRecords = recordsWithTimestamps.map(({ timestamp, ...rest
}) => rest);

        setRecords(sortedRecords);

      } catch (error) {
        console.error("Error loading audio files:", error);
      }
    }
    loadAudioFiles();
  }, []);

  return (
    <SafeAreaView className="flex-1 bg-[#dad7cd] justify-center items-center
h-full">
      <GluestackUIProvider mode={colorMode}>
        {/*Header*/}
        <HStack space="md" className="px-6 pt-4 pb-2">
          <Box>
            <Button
              className='h-[45] w-[45] bg-[#3a5a40] rounded-full p-3.5'
              onPress={() => {
                navigation.navigate("homePage");
              }}
```

```
            >
              <ButtonIcon size="lg" as={ArrowLeft} stroke={ "#a3b18a" }
className="h-[25] w-[25]"/>
            </Button>
          </Box>
          <Box className="flex-1 ml-3 justify-center">
            <Text size="5xl" style={{ color: '#344e41', fontFamily:
'LilyScriptOne-Regular' }}>Past Records</Text>
          </Box>
        </HStack>


        {/* List of past records */}
        <Center className="flex1 mt-3 h-70 pb-[225]">
          <FlatList
            data={records}
            renderItem={ ({item}) => {
              return <AudioFileCard
                navigation={navigation}
                record={item}
                audioClassifier={audioClassifier}
                isClassifierReady={isClassifierReady}
              />
            }}
            className="px-6 w-full"
          />
        </Center>

        {/* Bottom button */}
        <BottomButton navigation={navigation} />
        <WaveBackground />

      </GluestackUIProvider>
    </SafeAreaView>
  );
};


export default PastRecordsPage;
```

reactnative/app/SendFeedback.tsx

```
import * as MailComposer from 'expo-mail-composer';
import * as FileSystem from 'expo-file-system';
import { Alert, Platform } from 'react-native';
import Submission from "@/components/custom/SubmissionInterface";

async function sendFeedback({submission}:{submission: Submission}) {
  // Path to audio
  const audioUri = FileSystem.documentDirectory + 'birdappAudio/' +
submission.fileName;

  let attachments = [];
  try {
```

```
    // Check if the file exists
    const fileInfo = await FileSystem.getInfoAsync(audioUri);
    if (!fileInfo.exists) {
      throw new Error('File does not exist');
    }

    attachments.push(audioUri);

     // Check if email is available
    const isAvailable = await MailComposer.isAvailableAsync();
    if (!isAvailable) {
      Alert.alert('No email app found', 'Please install an email app to send
feedback');
      return;
    }

    // Open the mail composer with pre-filled data
    await MailComposer.composeAsync({
      subject: `Birdaudio Feedback: ${submission.fileName}`,
      recipients: ['80357@siswa.unimas.my'],
      body:
        `Supposed bird
species:\n${submission.birdSpecies}\n\nDescription:\n${submission.description
}`,
      attachments: attachments,
    });
  } catch (error) {
    console.error('Error sending feedback:', error);
  }
}


export default sendFeedback;
```

reactnative/components/custom

```
import React, { useState } from "react";
import "@/global.css";
import { Card } from "@/components/ui/card";
import { Heading } from "@/components/ui/heading";
import { Button } from "@/components/ui/button";
import { HStack } from "@/components/ui/hstack";
import { Text } from "@/components/ui/text";
import { AudioRecord } from "@/src/types/AudioRecord";
import { Box } from "@/components/ui/box";
import { Alert, ActivityIndicator } from "react-native";
import { AudioClassifier } from "@/src/audio/AudioClassifier";

interface AudioFileCardProps {
  navigation: any;
  record: AudioRecord;
  audioClassifier: AudioClassifier;
  isClassifierReady: boolean;
}
```

```
export const AudioFileCard = ({
  navigation,
  record,
  audioClassifier,
  isClassifierReady
}: AudioFileCardProps) => {
  const [isProcessing, setIsProcessing] = useState(false);

  const handleIdentifyAgain = async () => {
    if (!isClassifierReady) {
      Alert.alert(
        'Not Ready',
        'Audio classifier is still initializing. Please try again in a
moment.',
        [{text: 'OK'}]
      );
      return;
    }

    setIsProcessing(true);
    try {
      // Use the audioClassifier instance to classify the audio
      const encodedLabel = await audioClassifier.classifyAudio(record.uri);

      // Navigate to results page
      navigation.navigate('resultPage', {
        encodedLabel: encodedLabel,
        fileName: record.title
      });
    } catch (error) {
      console.error("Failed to classify audio:", error);
      Alert.alert(
        'Classification Error',
        'Failed to classify the audio. Please try again later.',
        [{text: 'OK'}]
      );
    } finally {
      setIsProcessing(false);
    }
  };

  return (
    <Card size="md" variant="filled" className="mb-4 bg-[#a3b18a] rounded-
2xl">
      <Heading size="xl" className="mx-2 mb-0 text-[#344e41]">
        {record.date}
      </Heading>
      <Box>
        <Text size="md" className="text-[#344e41] mx-2 mb-
5">{record.title}</Text>
        <HStack className="flex-1 justify-between">
          <Button
            variant="solid"
```

```
            size="md"
            className="bg-[#3a5a40] rounded-2xl"
            onPress={handleIdentifyAgain}
            disabled={isProcessing}
          >
            {isProcessing ? (
              <HStack space="sm" className="items-center">
                <ActivityIndicator size="small" color="#a3b18a" />
                <Text size="md" className="text-
[#a3b18a]">Processing...</Text>
              </HStack>
            ) : (
              <Text size="md" className="text-[#a3b18a]">Identify
again</Text>
            )}
          </Button>
          <Button
            variant="solid"
            size="md"
            className="bg-[#3a5a40] rounded-2xl"
            onPress={() => {
              navigation.navigate('feedbackPage', {fileName: record.title});
            }}
            disabled={isProcessing}
          >
            <Text size="md" className="text-[#a3b18a]">Send Feedback</Text>
          </Button>
        </HStack>
      </Box>
    </Card>
  );

};
```

reactnative/components/custom/BirdImageIndex.tsx

```
// Auto-generated file - DO NOT EDIT MANUALLY
const imagesIndex = {
  "Asian Glossy Starling": require("@/assets/images/bird/Asian Glossy
Starling.jpg"),
  "Ashy Drongo": require("@/assets/images/bird/Ashy Drongo.jpg"),
  "Ashy Tailorbird": require("@/assets/images/bird/Ashy Tailorbird.jpg"),
  "Square-tailed Drongo-Cuckoo": require("@/assets/images/bird/Square-tailed
Drongo-Cuckoo.jpg"),
  "Asian Koel": require("@/assets/images/bird/Asian Koel.jpg"),
  "Barred Eagle-Owl": require("@/assets/images/bird/Barred Eagle-Owl.jpg"),
  "Banded Kingfisher": require("@/assets/images/bird/Banded Kingfisher.jpg"),
  "Bornean Banded Pitta": require("@/assets/images/bird/Bornean Banded
Pitta.jpg"),
  "Black-and-yellow Broadbill": require("@/assets/images/bird/Black-and-
yellow Broadbill.jpg"),
  "Bornean Black-capped Babbler": require("@/assets/images/bird/Bornean
Black-capped Babbler.jpg"),
```

```
  "Blue-crowned Hanging Parrot": require("@/assets/images/bird/Blue-crowned
Hanging Parrot.jpg"),
  "Bornean Bulbul": require("@/assets/images/bird/Bornean Bulbul.jpg"),
  "Black Hornbill": require("@/assets/images/bird/Black Hornbill.jpg"),
  "Black-bellied Malkoha": require("@/assets/images/bird/Black-bellied
Malkoha.jpg"),
  "Blue-banded Pitta": require("@/assets/images/bird/Blue-banded Pitta.jpg"),
  "Black-headed Bulbul": require("@/assets/images/bird/Black-headed
Bulbul.jpg"),
  "Bornean Black Magpie": require("@/assets/images/bird/Bornean Black
Magpie.jpg"),
  "Black-naped Monarch": require("@/assets/images/bird/Black-naped
Monarch.jpg"),
  "Black-sided Flowerpecker": require("@/assets/images/bird/Black-sided
Flowerpecker.jpg"),
  "Black-throated Babbler": require("@/assets/images/bird/Black-throated
Babbler.jpg"),
  "Blue-winged Leafbird": require("@/assets/images/bird/Blue-winged
Leafbird.jpg"),
  "Blyth\'s Hawk-Eagle": require("@/assets/images/bird/Blyth\'s Hawk-
Eagle.jpg"),
  "Blyth\'s Paradise Flycatcher": require("@/assets/images/bird/Blyth\'s
Paradise Flycatcher.jpg"),
  "Blyth\'s Shrike-babbler": require("@/assets/images/bird/Blyth\'s Shrike-
babbler.jpg"),
  "Bornean Blue Flycatcher": require("@/assets/images/bird/Bornean Blue
Flycatcher.jpg"),
  "Bonaparte\'s Nightjar": require("@/assets/images/bird/Bonaparte\'s
Nightjar.jpg"),
  "Bornean Barbet": require("@/assets/images/bird/Bornean Barbet.jpg"),
  "Bornean Bristlehead": require("@/assets/images/bird/Bornean
Bristlehead.jpg"),
  "Bornean Leafbird": require("@/assets/images/bird/Bornean Leafbird.jpg"),
  "Bornean Spiderhunter": require("@/assets/images/bird/Bornean
Spiderhunter.jpg"),
  "Bornean Treepie": require("@/assets/images/bird/Bornean Treepie.jpg"),
  "Bold-striped Tit-Babbler": require("@/assets/images/bird/Bold-striped Tit-
Babbler.jpg"),
  "Brown-backed Flowerpecker": require("@/assets/images/bird/Brown-backed
Flowerpecker.jpg"),
  "Brown Barbet": require("@/assets/images/bird/Brown Barbet.jpg"),
  "Bronzed Drongo": require("@/assets/images/bird/Bronzed Drongo.jpg"),
  "Brown Fulvetta": require("@/assets/images/bird/Brown Fulvetta.jpg"),
  "Rusty-breasted Cuckoo": require("@/assets/images/bird/Rusty-breasted
Cuckoo.jpg"),
  "Blue-throated Bee-eater": require("@/assets/images/bird/Blue-throated Bee-
eater.jpg"),
  "Black-throated Wren-Babbler": require("@/assets/images/bird/Black-throated
Wren-Babbler.jpg"),
  "Bushy-crested Hornbill": require("@/assets/images/bird/Bushy-crested
Hornbill.jpg"),
  "Blue-eared Barbet": require("@/assets/images/bird/Blue-eared Barbet.jpg"),
  "Buff-rumped Woodpecker": require("@/assets/images/bird/Buff-rumped
Woodpecker.jpg"),
```

```
  "Black-winged Flycatcher-shrike": require("@/assets/images/bird/Black-
winged Flycatcher-shrike.jpg"),
  "Charlotte\'s Bulbul": require("@/assets/images/bird/Charlotte\'s
Bulbul.jpg"),
  "Sunda Scimitar Babbler": require("@/assets/images/bird/Sunda Scimitar
Babbler.jpg"),
  "Chestnut-crested Yuhina": require("@/assets/images/bird/Chestnut-crested
Yuhina.jpg"),
  "Chestnut-hooded Laughingthrush": require("@/assets/images/bird/Chestnut-
hooded Laughingthrush.jpg"),
  "Chestnut-rumped Babbler": require("@/assets/images/bird/Chestnut-rumped
Babbler.jpg"),
  "Checker-throated Woodpecker": require("@/assets/images/bird/Checker-
throated Woodpecker.jpg"),
  "Grey-hooded Babbler": require("@/assets/images/bird/Grey-hooded
Babbler.jpg"),
  "Cinereous Bulbul": require("@/assets/images/bird/Cinereous Bulbul.jpg"),
  "Common Iora": require("@/assets/images/bird/Common Iora.jpg"),
  "Crimson-breasted Flowerpecker": require("@/assets/images/bird/Crimson-
breasted Flowerpecker.jpg"),
  "Crested Serpent Eagle": require("@/assets/images/bird/Crested Serpent
Eagle.jpg"),
  "Cream-vented Bulbul": require("@/assets/images/bird/Cream-vented
Bulbul.jpg"),
  "Crimson-winged Woodpecker": require("@/assets/images/bird/Crimson-winged
Woodpecker.jpg"),
  "Dark-necked Tailorbird": require("@/assets/images/bird/Dark-necked
Tailorbird.jpg"),
  "Dark-throated Oriole": require("@/assets/images/bird/Dark-throated
Oriole.jpg"),
  "Dulit Frogmouth": require("@/assets/images/bird/Dulit Frogmouth.jpg"),
  "Dusky Munia": require("@/assets/images/bird/Dusky Munia.jpg"),
  "Crimson Sunbird": require("@/assets/images/bird/Crimson Sunbird.jpg"),
  "Common Emerald Dove": require("@/assets/images/bird/Common Emerald
Dove.jpg"),
  "Eurasian Tree Sparrow": require("@/assets/images/bird/Eurasian Tree
Sparrow.jpg"),
  "Fluffy-backed Tit-Babbler": require("@/assets/images/bird/Fluffy-backed
Tit-Babbler.jpg"),
  "Ferruginous Babbler": require("@/assets/images/bird/Ferruginous
Babbler.jpg"),
  "Ferruginous Partridge": require("@/assets/images/bird/Ferruginous
Partridge.jpg"),
  "Finsch\'s Bulbul": require("@/assets/images/bird/Finsch\'s Bulbul.jpg"),
  "Golden-naped Barbet": require("@/assets/images/bird/Golden-naped
Barbet.jpg"),
  "Golden-whiskered Barbet": require("@/assets/images/bird/Golden-whiskered
Barbet.jpg"),
  "Green Broadbill": require("@/assets/images/bird/Green Broadbill.jpg"),
  "Greater Coucal": require("@/assets/images/bird/Greater Coucal.jpg"),
  "Green Iora": require("@/assets/images/bird/Green Iora.jpg"),
  "Green Imperial Pigeon": require("@/assets/images/bird/Green Imperial
Pigeon.jpg"),
```

```
  "Greater Racket-tailed Drongo": require("@/assets/images/bird/Greater
Racket-tailed Drongo.jpg"),
  "Grey-bellied Bulbul": require("@/assets/images/bird/Grey-bellied
Bulbul.jpg"),
  "Grey-breasted Spiderhunter": require("@/assets/images/bird/Grey-breasted
Spiderhunter.jpg"),
  "Grey-chinned Minivet": require("@/assets/images/bird/Grey-chinned
Minivet.jpg"),
  "Grey-headed Babbler": require("@/assets/images/bird/Grey-headed
Babbler.jpg"),
  "Grey-headed Canary-flycatcher": require("@/assets/images/bird/Grey-headed
Canary-flycatcher.jpg"),
  "Grey-rumped Treeswift": require("@/assets/images/bird/Grey-rumped
Treeswift.jpg"),
  "Grey-throated Babbler": require("@/assets/images/bird/Grey-throated
Babbler.jpg"),
  "Hook-billed Bulbul": require("@/assets/images/bird/Hook-billed
Bulbul.jpg"),
  "Horsfield\'s Babbler": require("@/assets/images/bird/Horsfield\'s
Babbler.jpg"),
  "Hose\'s Broadbill": require("@/assets/images/bird/Hose\'s Broadbill.jpg"),
  "Indian Cuckoo": require("@/assets/images/bird/Indian Cuckoo.jpg"),
  "Blyth\'s Frogmouth": require("@/assets/images/bird/Blyth\'s
Frogmouth.jpg"),
  "Dark Hawk-Cuckoo": require("@/assets/images/bird/Dark Hawk-Cuckoo.jpg"),
  "Large-tailed Nightjar": require("@/assets/images/bird/Large-tailed
Nightjar.jpg"),
  "Lesser Green Leafbird": require("@/assets/images/bird/Lesser Green
Leafbird.jpg"),
  "Lesser Coucal": require("@/assets/images/bird/Lesser Coucal.jpg"),
  "Lesser Cuckooshrike": require("@/assets/images/bird/Lesser
Cuckooshrike.jpg"),
  "Little Green Pigeon": require("@/assets/images/bird/Little Green
Pigeon.jpg"),
  "Little Spiderhunter": require("@/assets/images/bird/Little
Spiderhunter.jpg"),
  "Mangrove Blue Flycatcher": require("@/assets/images/bird/Mangrove Blue
Flycatcher.jpg"),
  "Maroon-breasted Philentoma": require("@/assets/images/bird/Maroon-breasted
Philentoma.jpg"),
  "Malaysian Hawk-Cuckoo": require("@/assets/images/bird/Malaysian Hawk-
Cuckoo.jpg"),
  "Mangrove Whistler": require("@/assets/images/bird/Mangrove Whistler.jpg"),
  "Rail-babbler": require("@/assets/images/bird/Rail-babbler.jpg"),
  "Maroon Woodpecker": require("@/assets/images/bird/Maroon Woodpecker.jpg"),
  "Mountain Imperial Pigeon": require("@/assets/images/bird/Mountain Imperial
Pigeon.jpg"),
  "Moustached Babbler": require("@/assets/images/bird/Moustached
Babbler.jpg"),
  "Mountain Barbet": require("@/assets/images/bird/Mountain Barbet.jpg"),
  "Scaly-breasted Munia": require("@/assets/images/bird/Scaly-breasted
Munia.jpg"),
  "Olive-backed Sunbird": require("@/assets/images/bird/Olive-backed
Sunbird.jpg"),
```

```
  "Orange-bellied Flowerpecker": require("@/assets/images/bird/Orange-bellied
Flowerpecker.jpg"),
  "Pale Blue Flycatcher": require("@/assets/images/bird/Pale Blue
Flycatcher.jpg"),
  "Pacific Swallow": require("@/assets/images/bird/Pacific Swallow.jpg"),
  "Philippine Cuckoo-Dove": require("@/assets/images/bird/Philippine Cuckoo-
Dove.jpg"),
  "Malaysian Pied Fantail": require("@/assets/images/bird/Malaysian Pied
Fantail.jpg"),
  "Pied Triller": require("@/assets/images/bird/Pied Triller.jpg"),
  "Plaintive Cuckoo": require("@/assets/images/bird/Plaintive Cuckoo.jpg"),
  "Plain Sunbird": require("@/assets/images/bird/Plain Sunbird.jpg"),
  "Brown-throated Sunbird": require("@/assets/images/bird/Brown-throated
Sunbird.jpg"),
  "Puff-backed Bulbul": require("@/assets/images/bird/Puff-backed
Bulbul.jpg"),
  "Pygmy White-eye": require("@/assets/images/bird/Pygmy White-eye.jpg"),
  "Raffles\'s Malkoha": require("@/assets/images/bird/Raffles\'s
Malkoha.jpg"),
  "Red-bearded Bee-eater": require("@/assets/images/bird/Red-bearded Bee-
eater.jpg"),
  "Red-billed Malkoha": require("@/assets/images/bird/Red-billed
Malkoha.jpg"),
  "Red-breasted Partridge": require("@/assets/images/bird/Red-breasted
Partridge.jpg"),
  "Red-crowned Barbet": require("@/assets/images/bird/Red-crowned
Barbet.jpg"),
  "Red-legged Crake": require("@/assets/images/bird/Red-legged Crake.jpg"),
  "Reddish Scops Owl": require("@/assets/images/bird/Reddish Scops Owl.jpg"),
  "Red-throated Barbet": require("@/assets/images/bird/Red-throated
Barbet.jpg"),
  "Red-throated Sunbird": require("@/assets/images/bird/Red-throated
Sunbird.jpg"),
  "Rufous-crowned Babbler": require("@/assets/images/bird/Rufous-crowned
Babbler.jpg"),
  "Rufous-collared Kingfisher": require("@/assets/images/bird/Rufous-collared
Kingfisher.jpg"),
  "Rufous-fronted Babbler": require("@/assets/images/bird/Rufous-fronted
Babbler.jpg"),
  "Rufous Piculet": require("@/assets/images/bird/Rufous Piculet.jpg"),
  "Rufous Woodpecker": require("@/assets/images/bird/Rufous Woodpecker.jpg"),
  "Rufous-tailed Shama": require("@/assets/images/bird/Rufous-tailed
Shama.jpg"),
  "Rufous-tailed Tailorbird": require("@/assets/images/bird/Rufous-tailed
Tailorbird.jpg"),
  "Rufous-winged Philentoma": require("@/assets/images/bird/Rufous-winged
Philentoma.jpg"),
  "Scarlet Minivet": require("@/assets/images/bird/Scarlet Minivet.jpg"),
  "Scarlet-breasted Flowerpecker": require("@/assets/images/bird/Scarlet-
breasted Flowerpecker.jpg"),
  "Scaly-crowned Babbler": require("@/assets/images/bird/Scaly-crowned
Babbler.jpg"),
  "Scarlet-rumped Trogon": require("@/assets/images/bird/Scarlet-rumped
Trogon.jpg"),
```

```
  "Short-tailed Babbler": require("@/assets/images/bird/Short-tailed
Babbler.jpg"),
  "Bornean Frogmouth": require("@/assets/images/bird/Bornean Frogmouth.jpg"),
  "Slender-billed Crow": require("@/assets/images/bird/Slender-billed
Crow.jpg"),
  "Sooty-capped Babbler": require("@/assets/images/bird/Sooty-capped
Babbler.jpg"),
  "Spectacled Bulbul": require("@/assets/images/bird/Spectacled Bulbul.jpg"),
  "Spectacled Spiderhunter": require("@/assets/images/bird/Spectacled
Spiderhunter.jpg"),
  "Spotted Dove": require("@/assets/images/bird/Spotted Dove.jpg"),
  "Spotted Fantail": require("@/assets/images/bird/Spotted Fantail.jpg"),
  "Streaked Bulbul": require("@/assets/images/bird/Streaked Bulbul.jpg"),
  "Sunda Cuckoo": require("@/assets/images/bird/Sunda Cuckoo.jpg"),
  "Sunda Frogmouth": require("@/assets/images/bird/Sunda Frogmouth.jpg"),
  "Temminck\'s Babbler": require("@/assets/images/bird/Temminck\'s
Babbler.jpg"),
  "Temminck\'s Sunbird": require("@/assets/images/bird/Temminck\'s
Sunbird.jpg"),
  "Thick-billed Spiderhunter": require("@/assets/images/bird/Thick-billed
Spiderhunter.jpg"),
  "Van Hasselt\'s Sunbird": require("@/assets/images/bird/Van Hasselt\'s
Sunbird.jpg"),
  "Verditer Flycatcher": require("@/assets/images/bird/Verditer
Flycatcher.jpg"),
  "Hooded Pitta": require("@/assets/images/bird/Hooded Pitta.jpg"),
  "White-browed Shortwing": require("@/assets/images/bird/White-browed
Shortwing.jpg"),
  "White-breasted Waterhen": require("@/assets/images/bird/White-breasted
Waterhen.jpg"),
  "White-bellied Woodpecker": require("@/assets/images/bird/White-bellied
Woodpecker.jpg"),
  "White-breasted Woodswallow": require("@/assets/images/bird/White-breasted
Woodswallow.jpg"),
  "White-bellied Erpornis": require("@/assets/images/bird/White-bellied
Erpornis.jpg"),
  "White-chested Babbler": require("@/assets/images/bird/White-chested
Babbler.jpg"),
  "White-crowned Hornbill": require("@/assets/images/bird/White-crowned
Hornbill.jpg"),
  "Whitehead\'s Broadbill": require("@/assets/images/bird/Whitehead\'s
Broadbill.jpg"),
  "Whitehead\'s Spiderhunter": require("@/assets/images/bird/Whitehead\'s
Spiderhunter.jpg"),
  "Wreathed Hornbill": require("@/assets/images/bird/Wreathed Hornbill.jpg"),
  "Wrinkled Hornbill": require("@/assets/images/bird/Wrinkled Hornbill.jpg"),
  "Changeable Hawk-Eagle": require("@/assets/images/bird/Changeable Hawk-
Eagle.jpg"),
  "Yellow-breasted Flowerpecker": require("@/assets/images/bird/Yellow-
breasted Flowerpecker.jpg"),
  "Yellow-bellied Prinia": require("@/assets/images/bird/Yellow-bellied
Prinia.jpg"),
  "Yellow-bellied Warbler": require("@/assets/images/bird/Yellow-bellied
Warbler.jpg"),
```

```
  "Yellow-crowned Barbet": require("@/assets/images/bird/Yellow-crowned
Barbet.jpg"),
  "Yellow-rumped Flowerpecker": require("@/assets/images/bird/Yellow-rumped
Flowerpecker.jpg"),
  "Yellow-vented Bulbul": require("@/assets/images/bird/Yellow-vented
Bulbul.jpg"),
  "Zebra Dove": require("@/assets/images/bird/Zebra Dove.jpg"),
};

export default imagesIndex;
```

reactnative/components/custom/BottomButton.tsx

```
import React from "react";
import { Center } from "@/components/ui/center";
import { Button, ButtonIcon, ButtonText } from "@/components/ui/button";
import { ArrowLeftIcon, MenuIcon } from "lucide-react-native";
import { Drawer, DrawerContent } from "../ui/drawer";
import "@/global.css";
import { Box } from "@/components/ui/box";
import { VStack } from "@/components/ui/vstack";
import { Text } from "@/components/ui/text";
import { useFonts } from 'expo-font';
import RNExitApp from 'react-native-exit-app';

export const BottomButton = ({navigation}:{navigation:any}) => {
  const [showDrawer, setShowDrawer] = React.useState(false);
  {/* For fonts */}
    useFonts({
      'LilyScriptOne': require('@/assets/fonts/LilyScriptOne-Regular.ttf')
    });
  return (
    <Center className='flex-1 absolute bottom-20 w-full items-center'>
      {/*Button*/}
      <Button
        className='h-[80] w-[80] bg-[#3a5a40] rounded-full p-3.5'
        onPress={() => {
          setShowDrawer(true)
        }}
      >
        <ButtonIcon size="xl" as={MenuIcon} stroke={ "#a3b18a" }
className="h-[40] w-[40]"/>
      </Button>

      <Drawer
        isOpen={showDrawer}
        onClose={() => {
          setShowDrawer(false)
        }}
        size="full"
        anchor="bottom"
      >
        <DrawerContent className='h-full w-full p-0 bg-[#fefae0]'>
          {/* Menu items */}
```

```
          <Box className='flex-1 justify-center py-10 px-10 mb-[150px]'>
            <VStack space='sm' reversed={false}>
              <Center>
                <Text className='text-[70px] text-[#bc6c25] font-
[LilyScriptOne-Regular] focus'>Menu</Text>
              </Center>

              <Button
                variant="link"
                className="h-[50px] p-0"
                onPress={() => {
                  navigation.navigate("homePage");
                  setShowDrawer(false);
                }}
              >
                <ButtonText className="m-0 p-0 text-[30px] text-[#dda15e]
leading-[50px] w-[100%] data-[hover=true]:no-underline data-
[hover=true]:text-[#bc6c25] data-[active=true]:no-underline data-
[active=true]:text-[#bc6c25]">
                  Record Now
                </ButtonText>
              </Button>

              <Button
                variant="link"
                className="h-[50px] p-0"
                onPress={() => {
                  navigation.navigate("pastRecords");
                  setShowDrawer(false);
                }}
              >
                <ButtonText className="m-0 p-0 text-[30px] text-[#dda15e]
leading-[50px] w-[100%] data-[hover=true]:no-underline data-
[hover=true]:text-[#bc6c25] data-[active=true]:no-underline data-
[active=true]:text-[#bc6c25]">
                  Past Records
                </ButtonText>
              </Button>

              <Button
                variant="link"
                className="h-[50px] p-0"
                onPress={() => RNExitApp.exitApp()}
              >
                <ButtonText className="m-0 p-0 text-[30px] text-[#dda15e]
leading-[50px] w-[100%] data-[hover=true]:text-[#bc6c25] data-
[active=true]:no-underline data-[active=true]:text-[#bc6c25]">
                  Exit App
                </ButtonText>
              </Button>

              <Center className='mt-10'>
                <Text className='text-[13px] text-[#bc6c25]'>Made by Nelson
with ♡</Text>
```

```
              </Center>

            </VStack>
          </Box>

          {/* Back button */}
          <Center className='flex-1 absolute bottom-20 w-full items-center'>
            <Button
              className='h-[80] w-[80] bg-[#3a5a40] rounded-full p-3.5'
              onPress={() => {
                setShowDrawer(false)
              }}
            >
              <ButtonIcon as={ArrowLeftIcon} stroke={ "#a3b18a" }
className="h-[40] w-[40]"/>
            </Button>
          </Center>
        </DrawerContent>
      </Drawer>
    </Center>
  );

}
```

reactnative/components/custom/DodoIcon.tsx

```
import React from "react";
import Svg, { G, Path } from "react-native-svg";
import {createIcon} from '@gluestack-ui/icon';

const DodoIcon = createIcon(
  {
    Root: Svg,
    viewBox: '0 0 512 512',
    path: (
    <>
      <G translate={[0, 0]} >
        <Path
          fill="rgba(96, 108, 56, 0.5)"
          fill-opacity="1"
          d="M272.707 426.95c-26.005 20.19-10.044 48.44-10.044 48.44s22.858
18 74.775-4.46l18.029 23.67-181.924-1.92 11.302-19.31 45.555 4.95-9.983-
48.63zm104.64-290.068c2.471 49.377 128.226 132.735 54.755 197.376l-1.562-
31.615C367.701 426.78 196.513 422.43 137.732 392.677 7.55 326.776 143.64
50.138 330.303 212.926c16.361-15.75-12.617-60.152-34.995-93.871-49.251-74.212
13.036-100.877 63.58-91.489-10.094 3.919-17.163 9.349-22.026 15.432-9.691
12.125-11.195 27.43-9.07 41.526 2.924 19.397 12.585 36.134 12.585 36.134l.927
1.608 1.523 1.062c11.848 8.266 23.408 12.168 34.52 13.554zM225.674 292.354c-
5.392 1.379-10.052 3.593-13.51 6.3-5.166 4.043-8.028 9.212-8.399 14.627-.329
4.803 1.215 10.095 5.927 15.25a31.96 31.96 0 0 0 3.098 2.942c-13.869 4.423-
28.425 13.781-34.678 32.733l-2.612 7.914 7.914 2.611c54.508 17.987 98.5
14.265 127.793 2.047 21.097-8.799 34.847-22.173 40.499-35.431 4.377-10.266
4.225-20.643-.519-29.754-4.756-9.134-14.667-17.359-30.731-22.108l19.384-7.899-
10.732-12.751-37.608 31.653 19.374 2.11c30.192 3.29 42.059 16.829 35.501
```

```
32.213-4.337 10.171-15.398 19.834-31.584 26.585-24.996 10.426-61.756 13.49-
107.155.301 5.605-8.563 14.414-12.799 22.389-14.99 10.857-2.982 20.473-2.256
20.473-2.256l3.078-16.282c-9.445-2.865-15.942-5.997-19.954-9.344-1.945-1.623-
3.333-2.888-3.229-4.406.125-1.827 2.105-2.918 4.26-4.055 6.087-3.21 15.409-
4.119 24.718-1.296l5.183-15.836c-3.24-1.139-6.009-2.424-8.316-3.813-1.63-
.982-3.01-2.003-4.133-3.057-1.011-.948-1.793-1.886-2.277-2.839-.17-.332-.596-
.575-.369-.802.935-.934 2.558-1.266 4.586-1.467 3.54-.351 7.976.181 13.244
1.716l4.661-16.002c-12.193-3.552-21.553-2.85-27.591-.382-5.964 2.437-9.464
6.614-10.859 11.279-1.293 4.325-.849 9.389 2.174 14.489zm-115.661-95.945c-
18.612-123.322-140.144-13.656-43.562-6.438-37.548 48.185 7.946 120.58 7.946
120.58s-22.445-66.16 35.616-114.142zm278.21-158.16.064.039c-6.751 20.123
20.117 44.03 43.21 35.462 34.833-12.924 62.19 11.059 46.296 62.052-4.644-
9.155-10.177-14.779-16.168-18.193-11.174-6.365-24.739-5.004-40.103-1.769-
19.552 4.117-42.845 11.178-67.572-5.104-2.153-4.12-7.718-15.696-9.677-28.696-
1.459-9.677-1.045-20.312 5.608-28.635 6.512-8.147 18.434-13.679 38.342-
15.156z"/>
      </G>
    </>
    ),
  }
)

export default DodoIcon;
```

reactnative/components/custom/StopIcon.tsx

```
import React from 'react';
import {Svg, Path} from 'react-native-svg';
import {createIcon} from '@gluestack-ui/icon';

const StopIcon = createIcon({
  Root: Svg,
  viewBox: '0 0 384 512',
  path: (
    <>
      <Path
        fill="#3a5a40"
        d="M0 128c0-35.3 28.7-64 64-64h256c35.3 0 64 28.7 64 64v256c0 35.3-
28.7 64-64 64H64c-35.3 0-64-28.7-64-64z"/>
    </>
  ),
});

StopIcon.displayName = 'StopIcon';

export default StopIcon;
```

reactnative/components/SubmissionInterface.tsx

```
interface Submission {
  fileName: string,
  birdSpecies: string,
  description: string
}
```

```
export default Submission;
```

reactnative/components/WaveBackground.tsx

```
import React from "react";
import { Box } from "@/components/ui/box";
import Svg, { Path } from "react-native-svg";

export const WaveBackground = () => {
  return (
    <Box className="absolute bottom-0 w-full h-20 z-0 m-0 p-0">
      <Svg height="100%" width="100%" viewBox="0 0 1440 20"
preserveAspectRatio="xMinYMin meet">
      <Path fill="#588157" fill-opacity="1"
d="M0,224L48,218.7C96,213,192,203,288,176C384,149,480,107,576,112C672,117,768
,171,864,202.7C960,235,1056,245,1152,240C1248,235,1344,213,1392,202.7L1440,19
2L1440,320L1392,320C1344,320,1248,320,1152,320C1056,320,960,320,864,320C768,3
20,672,320,576,320C480,320,384,320,288,320C192,320,96,320,48,320L0,320Z"></Pa
th>
      </Svg>
    </Box>
  );

};
```

reactnative/src/audio/AudioClassifier.tsx

```
// AudioClassifier.ts
import * as FileSystem from 'expo-file-system';
import { ServerFeatureExtractor } from '@/src/audio/ServerFeatureExtractor';
import { ModelInference } from '@/src/audio/ModelInference';
import {
  FlattenedFeatures,
  FEATURE_DIMS
} from '@/src/types/audio.types';
import { Asset } from 'expo-asset';


export class AudioClassifier {
  private modelInference: ModelInference;
  private modelsDirectory: string;
  private serverExtractor: ServerFeatureExtractor;

  constructor() {
    this.modelsDirectory = `${FileSystem.documentDirectory}models/`;
    this.modelInference = new ModelInference(this.modelsDirectory);
    this.serverExtractor = new ServerFeatureExtractor();
  }

  /**
   * Initialize classifier (only loads scalers)
   */
```

```
  async initialize(): Promise<void> {
    try {
    const dirInfo = await FileSystem.getInfoAsync(this.modelsDirectory);
    if (!dirInfo.exists) {
      await FileSystem.makeDirectoryAsync(this.modelsDirectory, {
intermediates: true });
      console.log(`Created models directory: ${this.modelsDirectory}`);
    }

    // Define model files
    const modelAssets = [
      { name: 'model_fold_0.onnx', module:
require('@/assets/models/model_fold_0.onnx') },
      { name: 'model_fold_1.onnx', module:
require('@/assets/models/model_fold_1.onnx') },
      { name: 'model_fold_2.onnx', module:
require('@/assets/models/model_fold_2.onnx') },
      { name: 'model_fold_3.onnx', module:
require('@/assets/models/model_fold_3.onnx') },
      { name: 'model_fold_4.onnx', module:
require('@/assets/models/model_fold_4.onnx') }
    ];

    // Copy model files with proper error handling
    await Promise.all(modelAssets.map(async (modelAsset) => {
      try {
        const asset = Asset.fromModule(modelAsset.module);
        await asset.downloadAsync();

        if (!asset.localUri) {
          throw new Error(`No local URI available for ${modelAsset.name}`);
        }

        await FileSystem.copyAsync({
          from: asset.localUri,
          to: `${this.modelsDirectory}${modelAsset.name}`
        });

        console.log(`Successfully copied model: ${modelAsset.name}`);
      } catch (error) {
        console.error(`Failed to process model ${modelAsset.name}:`, error);
        throw error;
      }
    }));

    await this.modelInference.loadScalers();
    console.log('AudioClassifier initialized and scalers loaded');

    // Test server health
    const connectionStatus = await this.serverExtractor.checkHealth();
    if (!connectionStatus) {
      console.warn('Server feature extractor is not reachable. Using local
models only.');
    } else {
```

```
      console.log('Server feature extractor is reachable');
    }

    } catch (error) {
      console.error('Initialization error:', error);
      throw error;
    }
  }

  /**
   * Classify audio file
   */
  async classifyAudio(fileUri: string): Promise<number> {
    try {
      console.log(`Processing audio file: ${fileUri}`);

      let flattenedFeatures: FlattenedFeatures;
      flattenedFeatures = await
this.serverExtractor.extractFeatures(fileUri);

      // Validate dimensions
      this.validateFeatureDimensions(flattenedFeatures);

      // Run inference (models loaded on demand)
      console.log('Running inference...');
      const result = await this.modelInference.classify(flattenedFeatures);

      return result;

    } catch (error) {
      console.error('Classification error:', error);
      throw error;
    }
  }


  /**
   * Validate feature dimensions
   */
  private validateFeatureDimensions(features: FlattenedFeatures): void {
    const dims = {
      logmel: features.logmel.length,
      mfcc: features.mfcc.length,
      chroma: features.chroma.length,
      tonnetz: features.tonnetz.length
    };

    console.log('Feature dimensions:', dims);

    const total = dims.logmel + dims.mfcc + dims.chroma + dims.tonnetz;
    console.log(`Total features: ${total} (expected: ${FEATURE_DIMS.logmel +
FEATURE_DIMS.mfcc + FEATURE_DIMS.chroma + FEATURE_DIMS.tonnetz})`);
  }
```

```
  /**
   * Clean up resources
   */
  async dispose(): Promise<void> {
    await this.modelInference.dispose();
  }

}
```

reactnative/src/audio/AudioRecorder.tsx

```
import AudioRecorderPlayer, {
  AVEncoderAudioQualityIOSType,
  AVEncodingOption,
  AudioEncoderAndroidType,
  AudioSourceAndroidType,
  OutputFormatAndroidType,
} from 'react-native-audio-recorder-player';
import * as FileSystem from 'expo-file-system';
import { Platform } from 'react-native';

export class AudioRecorder {
  private audioRecorderPlayer: AudioRecorderPlayer;
  private currentPath: string = '';

  constructor() {
    this.audioRecorderPlayer = new AudioRecorderPlayer();
  }

  /**
   * Start recording audio in WAV format
   */
  async startRecording(): Promise<void> {
    try {
      // Ensure audio directory exists
      const audioDir = `${FileSystem.documentDirectory}birdappAudio/`;
      const dirInfo = await FileSystem.getInfoAsync(audioDir);

      if (!dirInfo.exists) {
        await FileSystem.makeDirectoryAsync(audioDir, { intermediates: true
});
      }

      // Generate file path
      const fileName = `${Date.now()}.ogg`;
      this.currentPath = `${audioDir}${fileName}`;

      // Configure recording options for M4A format
      const audioSet = {
        AudioEncoderAndroid: AudioEncoderAndroidType.OPUS,
        AudioSourceAndroid: AudioSourceAndroidType.MIC,
        AVEncoderAudioQualityKeyIOS: AVEncoderAudioQualityIOSType.high,
        AVNumberOfChannelsKeyIOS: 1,
        AVFormatIDKeyIOS: AVEncodingOption.lpcm,
```

```
        OutputFormatAndroid: OutputFormatAndroidType.OGG,
        AudioSamplingRateAndroid: 32000,
        AudioChannelsAndroid: 1,
      };

      // Platform-specific path handling
      const path = Platform.select({
        ios: this.currentPath,
        android: this.currentPath.replace('file://', ''),
      });

      // Start recording
      const uri = await this.audioRecorderPlayer.startRecorder(path,
audioSet);
      console.log('Recording started:', uri);

    } catch (error) {
      console.error('Failed to start recording:', error);
      throw error;
    }
  }

  /**
   * Stop recording and return the file path
   */
  async stopRecording(): Promise<string> {
    try {
      const result = await this.audioRecorderPlayer.stopRecorder();
      console.log('Recording stopped:', result);

      return this.currentPath;
    } catch (error) {
      console.error('Failed to stop recording:', error);
      throw error;
    }
  }

  /**
   * Clean up resources
   */
  async cleanup(): Promise<void> {
    this.audioRecorderPlayer.removeRecordBackListener();
  }

}
```

reactnative/src/audio/ModelInference.tsx

```
import AudioRecorderPlayer, {
  AVEncoderAudioQualityIOSType,
  AVEncodingOption,
  AudioEncoderAndroidType,
  AudioSourceAndroidType,
  OutputFormatAndroidType,
```

```
} from 'react-native-audio-recorder-player';
import * as FileSystem from 'expo-file-system';
import { Platform } from 'react-native';

export class AudioRecorder {
  private audioRecorderPlayer: AudioRecorderPlayer;
  private currentPath: string = '';

  constructor() {
    this.audioRecorderPlayer = new AudioRecorderPlayer();
  }

  /**
   * Start recording audio in WAV format
   */
  async startRecording(): Promise<void> {
    try {
      // Ensure audio directory exists
      const audioDir = `${FileSystem.documentDirectory}birdappAudio/`;
      const dirInfo = await FileSystem.getInfoAsync(audioDir);

      if (!dirInfo.exists) {
        await FileSystem.makeDirectoryAsync(audioDir, { intermediates: true
});
      }

      // Generate file path
      const fileName = `${Date.now()}.ogg`;
      this.currentPath = `${audioDir}${fileName}`;

      // Configure recording options for M4A format
      const audioSet = {
        AudioEncoderAndroid: AudioEncoderAndroidType.OPUS,
        AudioSourceAndroid: AudioSourceAndroidType.MIC,
        AVEncoderAudioQualityKeyIOS: AVEncoderAudioQualityIOSType.high,
        AVNumberOfChannelsKeyIOS: 1,
        AVFormatIDKeyIOS: AVEncodingOption.lpcm,
        OutputFormatAndroid: OutputFormatAndroidType.OGG,
        AudioSamplingRateAndroid: 32000,
        AudioChannelsAndroid: 1,
      };

      // Platform-specific path handling
      const path = Platform.select({
        ios: this.currentPath,
        android: this.currentPath.replace('file://', ''),
      });

      // Start recording
      const uri = await this.audioRecorderPlayer.startRecorder(path,
audioSet);
      console.log('Recording started:', uri);

    } catch (error) {
```

```
      console.error('Failed to start recording:', error);
      throw error;
    }
  }

  /**
   * Stop recording and return the file path
   */
  async stopRecording(): Promise<string> {
    try {
      const result = await this.audioRecorderPlayer.stopRecorder();
      console.log('Recording stopped:', result);

      return this.currentPath;
    } catch (error) {
      console.error('Failed to stop recording:', error);
      throw error;
    }
  }

  /**
   * Clean up resources
   */
  async cleanup(): Promise<void> {
    this.audioRecorderPlayer.removeRecordBackListener();
  }

}
```

reactnative/src/audio/ServerFeatureExtractor.tsx

```
import { FlattenedFeatures } from '@/src/types/audio.types';
import { SERVER_CONFIG } from '@/src/config/serverConfig';

export class ServerFeatureExtractor {
  private serverUrl: string;

  constructor(serverUrl: string = SERVER_CONFIG.URL) {
    this.serverUrl = serverUrl;
  }

  async extractFeatures(audioUri: string): Promise<FlattenedFeatures> {
    try {
      console.log('Uploading audio to server for feature extraction...');

      // Create form data
      const formData = new FormData();
      formData.append('audio', {
        uri: audioUri,
        type: 'audio/wav',
        name: 'recording.wav'
      } as any);

      // Upload to server with timeout
```

```
      const controller = new AbortController();
      const timeoutId = setTimeout(() => controller.abort(),
SERVER_CONFIG.TIMEOUT.FEATURE_EXTRACTION);

      // Upload to server
      const response = await fetch(`${this.serverUrl}/extract-features`, {
        method: 'POST',
        body: formData,
        headers: {
          'Content-Type': 'multipart/form-data',
        },
        signal: controller.signal
      });

      clearTimeout(timeoutId);

      if (!response.ok) {
        const error = await response.text();
        throw new Error(`Server error: ${response.status} - ${error}`);
      }

      const result = await response.json();

      if (!result.success) {
        throw new Error(result.message || 'Feature extraction failed');
      }

      console.log('Features extracted successfully from server');
      return result.features;

    } catch (error) {
      console.error('Server feature extraction failed:', error);
      throw new Error(`Failed to extract features: ${error instanceof Error ?
error.message : 'Unknown error'}`);
    }
  }

  async checkHealth(): Promise<boolean> {
    try {
      const controller = new AbortController();
      const timeoutId = setTimeout(() => controller.abort(),
SERVER_CONFIG.TIMEOUT.HEALTH_CHECK);

      const response = await fetch(`${this.serverUrl}/health`, {
        method: 'GET',
        signal: controller.signal
      });

      clearTimeout(timeoutId);
      return response.ok;
    } catch {
      return false;
    }
  }
```

```
}
```

reactnative/src/config/serverConfig.tsx

```
export const SERVER_CONFIG = {
  URL: '/url/to/server',
  TIMEOUT: {
    HEALTH_CHECK: 5000,    // 5 seconds
    FEATURE_EXTRACTION: 30000  // 30 seconds
  }

} as const;
```

reactnative/src/types/audio.types.tsx

```
// types/audio.types.ts
export interface AudioFeatures {
  mfcc: number[][];
  melSpectrogram: number[][];
  chroma: number[][];
  tonnetz: number[][];
}

export interface FeatureDimensions {
  logmel: number;    // 48128
  mfcc: number;      // 3760
  chroma: number;    // 2256
  tonnetz: number;   // 1128
}

export interface FlattenedFeatures {
  logmel: number[];
  mfcc: number[];
  chroma: number[];
  tonnetz: number[];
}

export interface ScalerParams {
  mean: number[];
  std: number[];
  var: number[];
  n_features: number;
  feature_dim: number;
}

export interface FeatureScalers {
  logmel: ScalerParams;
  mfcc: ScalerParams;
  chroma: ScalerParams;
  tonnetz: ScalerParams;
}

interface Metadata {
  n_folds: number;
```

```
  feature_types: string[];
  feature_dims: FeatureDimensions;
  total_dim: number;
}

export interface AllScalerParams {
  [foldKey: string]: FeatureScalers | Metadata;
  metadata: Metadata;
}

export interface Complex {
  re: number;
  im: number;
}

// Feature dimensions constants
export const FEATURE_DIMS: FeatureDimensions = {
  logmel: 48128,
  mfcc: 3760,
  chroma: 2256,
  tonnetz: 1128
};


export const TOTAL_FEATURE_DIM = 55272;
```

reactnative/src/types/AudioRecord.tsx

```
export interface AudioRecord {
  title: string;
  date: string;
  uri: string;

                                        }
```