2021-2022

Develop an IoT Edge device to Capture and Classify Species using Sounds to Support Wildlife Conservation Activities

---

Cardiff University
School of Computer Science and Informatics

CM3203 - One Semester Individual Project - 40 Credits

Author: Ruslan Levond
Supervisor: Charith Perera
Moderator: Victor Gutierrez Basulto

Table of Contents

Table of Figures

Abstract

Wildlife has been in danger for quite some time from various kinds of human activities, ranging from directly destroying habitat to impact from climate change caused by human's unsustainable way of living. Many organisations have established conservation projects and activities to try to mitigate the effects and save wildlife, one of the most important activities of which is monitoring species. Automatic sound recognition systems have proven to be an effective tool used during conservation activities. There are several devices out on the market which can be installed in the wild and record animal sounds. However, they are inaccessible due to being expensive and they only record sounds and require further proprietary software to classify sounds elsewhere. The intention of the project is to create an alternative low-cost device that can be installed in the wild and be able to both record and classify animal sounds right on the edge. This involved developing a machine learning model from scratch that is able to classify bird sounds. The project also created edge frameworks for two architectures, Raspberry Pi and Arduino to which the machine learning model is deployed to. As well, 3D case solutions were designed allowing for both architectures to be safely deployed in the wild. The project then created a gateway device and a framework for it which is used to store results transmitted by edge devices. Afterwards, the created model was evaluated against competitor's model which showed to have a competitive performance, outperforming competition in some cases. The project has also investigated performance of the model running on both architectures and compared architectures to understand which one is more suitable to use when. To achieve the outlined goals, the project has tackled the development of IoT applications, fundamentals of machine learning, architectural design and development of computer aided designs.

Acknowledgements

I would like to thank my supervisor Charith Perera for providing continuous support and guidance throughout the project, and his students for providing expertise in new to me topics.

I would also like to thank Xeno Canto and BBC Sound Effects for providing critically important audio samples used to train the model.

1 Introduction

Automatic sound recognition has been available for a while, the first system being developed back in 1952 by Bell Laboratories called Audrey (Sonix, 2021), which was able to recognise human's speech, identifying spoken numbers. Over the past years, many various systems and projects were developed that performed complex sound recognition using machine learning techniques, including animal sound recognition.

Within the ecosystem, different animals live in the harmony with each other, making the environment stable and sustainable. However, wildlife has become endangered in the recent years due to many factors caused by human's destructive nature such as global warming and poaching. This can be backed up by the report from World Wide Fund for Nature organisation (WWF, 2020), reporting a major 68% decline in species population sizes since 1970. Due to this reason, many conservation projects have been established with the aim to preserve wildlife. Automatic sound recognition has played a crucial part in animal conservation activities, using the technology for monitoring purposes by providing an automatic way to recognise animals using sounds. There are some commercially available systems that support such activities, however most of them provide just a recording device that is installed in the wild with the functionality to only capture sound and requires further proprietary software to classify sounds, both of which are expensive. An example of such device is one of the Song Meter devices from Wildlife Acoustics (Wildlife Acoustics, 2022) starting from $249, which is a waterproof recorder that is required to be retrieved to download the audio and then import audio into their software to perform analysis and classification.

The project aims to solve this inaccessibility issue by creating a low-cost edge device that can be deployed in the wild, with ability to both record and classify sounds right on the edge whilst having the same performance as commercially available products. The device should also be able to send the results over long-range wireless network back to the user without requiring the user to retrieve the device for results.

The project's first objective was to create a machine learning model from scratch, being able to identify a singular bird called Araripe Manakin. The model was evaluated both on paper and real world, against competitor's models to see if it performs as good. The project also designed and developed frameworks for two different architectures, Arduino and Raspberry Pi that records audio, facilitates the model and sends model's results wirelessly to the gateway device. The project tested model's performance on both architectures, analysing architectures' capabilities and identifying strengths and weaknesses of each architecture. Project also included creating a gateway device which is used to listen to results from edge devices and store them. As well as, providing a command line tool for the user, used to manage results stored locally. Lastly, project designed and implemented 3D case solutions for both edge devices which will allow devices to be deployed in the wild, being able to protect electronics from weather elements and facilitate various kinds of installations on different surface areas.

2 Background

This section provides the context for the project and all relevant information needed to be known before reading the specifics of the project.

2.1 Machine Learning Concepts

2.1.1 Balanced and Imbalanced Dataset

A balanced dataset contains an equal or almost equal amount of data samples in each of the classes, whereas imbalanced dataset has one of the classes containing higher number of samples than other classes. An example of both datasets can be seen on Figure 2.1 and Figure 2.2, showing datasets with 2 classes. The project faces imbalanced dataset issue due to the nature of the dataset, which needs to be resolved.

Figure 2.1 Representation of Balanced Dataset

Figure 2.2 Representation of Imbalanced Dataset

2.1.2 Undersampling and Oversampling

Both techniques are used to convert imbalanced datasets into balanced. Oversampling is a technique that is used to duplicate a minority class within the dataset with enough number of samples to have the same amount as majority classes. Comparing to undersampling, it is the opposite which removes samples from the majority class to have the same number of samples as minority classes. In the project, we will only be implementing one of the oversampling techniques because of limited amount of data being available.

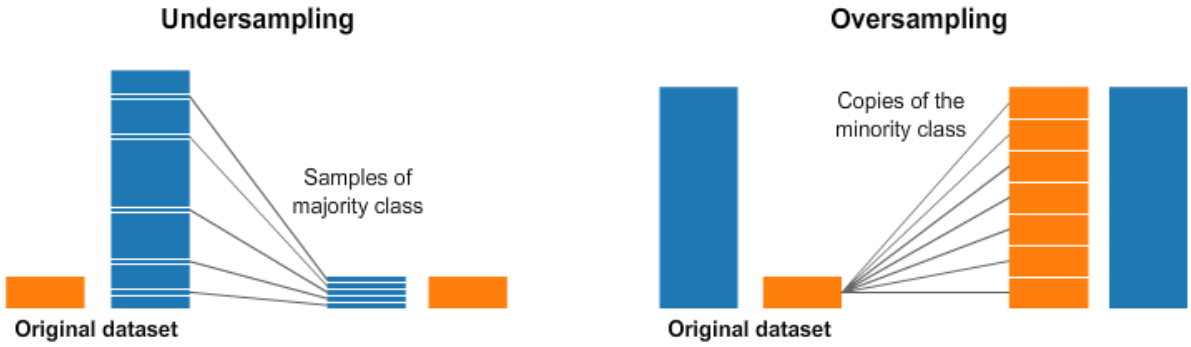


Figure 2.3 Representation of Undersampling and Oversampling (Rahman, 2021)

2.1.3 Data Augmentation

Data augmentation is a technique that is used to increase the amount of data in the dataset by synthetically creating data through copying existing data and slightly modifying it. This technique allows for increasing diversity of the data too, without requiring to collect new data making it very powerful when working with small datasets. It has been proved that using data augmentation can improve the accuracy of the model by 1-3% (Moreno-Barea, et al., 2020) and the generalisation capability of the model. During the project, this technique will be used for oversampling one of the classes in the dataset.

2.1.4 Convolutional Neural Network

CNN is a subset of machine learning, being a Deep Learning algorithm that specialises in processing image type of data. It works by having a hierarchy of layers, where each layer identifies different features and parts of the image. Taking an image as input, CNN passes it through these layers, earlier ones identifying simple features such as colours and later layers identifying bigger elements of the feature such as shapes, until CNN finally recognises the class the image belongs to. This is just an overview of Convolutional Neural Network, for more information please check out an explanation from IBM Cloud Education (IBM Cloud Education, 2020). The project creates a CNN model that is used to classify birds.

2.1.5 Hyperparameters

Hyperparameters are parameters that control the learning process of machine learning model. They are used to determine how the network is trained, examples of hyperparameters are learning rate and number of training cycles. They are set by the practitioner before the training and are neither changed by the model during training nor used by the resulting trained model. During the project, hyperparameters will be changed before each training cycle to try to increase model's performance.

2.1.6 Nyquist-Shannon Theorem

Nyquist-Shannon Theorem is a sampling theorem used in signal processing which states "that a sinusoidal function in time or distance can be regenerated with no loss of information as long as it is sampled at a frequency greater than or equal to twice per cycle" (Colarusso, et al., 1999). Meaning that any sampled analog signal should be collected at twice the frequency of the highest expected frequency at least, otherwise the sample can suffer from loss of information. This theorem is used to decide project's model input audio frequency.

2.1.7 Machine Learning Model Evaluation Metrics

The following evaluation metrics will be used to evaluate machine learning models, to understand each model's strengths and weaknesses.

Accuracy

Accuracy is the score that represents a ratio of correctly predicted observations to total observations, which is calculated by number of classifications a model correctly predicted divided by the total number of predictions made. The metric implies how well can model identify patterns within data. This metric is great for measuring on symmetric datasets where there are the same number of false negatives as false positives.

Precision

Precision is used to show the correctness of classification, saying out of all positive class predictions, how many are actually positive. The score is calculated by having number of correct positive classifications divided by the total number of predicted positive classifications. High precision score ultimately shows that model does not pick up many false positives.

Recall

Recall is used to tell if model is correctly identifying true positives. It can be calculated through number of correct positive classifications divided by the total number of actual positive cases. High recall score tells that the model classifies less of false negatives.

F1 Score

F1 Score is a weighted average of both precision and recall, shows a balanced view of both scores. This means that the F1 Score takes into account both false positives and false negatives.

Confusion Matrix

Confusion matrix is a visualisation of the performance of a model, presented in a tabular view. Specifically, confusion matrix shows the number of correct and incorrect predictions which are represented by a value and separated by classification. Each row in the table represents instances of the actual class and each column represents instances that were predicted as the class.

2.1.8 Confidence Threshold

Each machine learning model's classification returns a value for each of the class, telling how confident the model is about its prediction for the class. Confidence Threshold is a threshold that model's confidence value on the class should match or exceed to be considered as a positive prediction. Confidence threshold will be set by edge framework to decide when model's classification is a positive prediction.

2.1.9 Feature Extraction Techniques

Feature extraction is a very important part of the machine learning model, which is used to find relations and patterns within data. Each technique transforms inputted data into numerical features in its own way, that can be then processed by the model. During the project, we will be exploring a few different audio specific feature extraction techniques to see which one works better for the scenario.

Spectrogram

Spectrogram extracts time and frequency features from the audio signal, capturing features concisely on an image. It is known to perform well on non-voice audio. Edge Impulse's documentation (Edge Impulse, 2022) explains it further on how they extract features from audio using spectrograms.

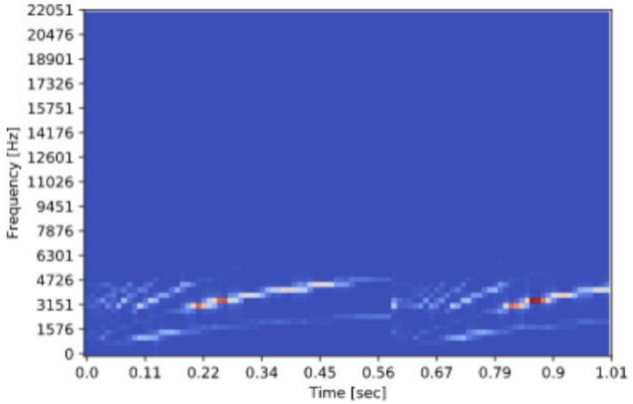


Figure 2.4 Example of Spectrogram extracted from 1 second audio

Mel-filterbank Energies

MFE works similarly to the spectrogram, also extracting time and frequency from the audio signal. Although, this time frequency is represented in a non-linear scale called Mel-scale which is known to perform good on non-voice audio again, especially on audio that humans can hear. For more information, please read Edge Impulse's documentations (Edge Impulse, 2022).

Figure 2.5 Example of MFE extracted from 1 second audio

Mel-frequency Cepstral Coefficient

MFCC extracts coefficients from the audio signal, using non-linear scale called Mel-scale just like MFE. It is mostly used for speech recognition and sometimes performs good on non-human voice audio such as animal communications. For more information, refer to Edge Impulse's documentation (Edge Impulse, 2022).

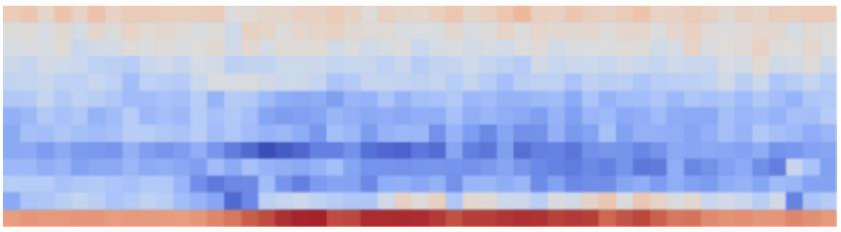


Figure 2.6 Example of MFCC extracted from 1 second audio

2.2 Competitors

2.2.1 Microsoft Acoustic Bird Detection Model

Microsoft Acoustic Bird Detection Model (Microsoft AI for Earth, 2020) is a machine learning model developed by Microsoft which we will refer to as "Microsoft's model". This is the main model that our project's model was compared to. Microsoft's model deals with the limited dataset issue, training the model to classify only one bird Araripe Manakin with small number of audio samples available. Microsoft's model carries out classifications on 2 second audio windows and uses MFCC to extract features from the audio. The model chose SVM

classifier to make predictions on features because of its reputation of performing well on small datasets, which will be covered in more detail in the next subsection. However, due to spectrograms being large and high dimensional, SVM classifier would not be trained well with such a small dataset. Therefore, Microsoft's model decided to reduce input dimensionality with the use of Principal Component Analysis (PCA) (Jolliffe & Cadima, 2016) but at the same time preserving all of the necessary information for class differentiation. Even with PCA, training the model from scratch resulted in poor performance, 87% accuracy and recall score of only 0.61. So, Microsoft decided to use transfer learning instead, using the Biophony Model that previously extracted hundreds of thousands of audio examples of more than 300 species, using the pre-trained model to leverage feature extraction capabilities. Microsoft's model has removed the last layer from the Biophony Model which was used to classify features and replaced it with the trained SVM classifier. This resulted in Biophony Model becoming a feature extraction tool that passes features to the SVM classifier, creating a high accuracy model.

Support Vector Machine

SVM classifier is a type of deep learning algorithm which is used to analyse data for classifications and make predictions. SVM works by plotting the whole dataset to a high dimensional feature space, where each sample's features are represented by a singular point on that space. Next, SVM looks at these data points and groups them by their positioning, groups are separated into sections using a hyperplane. When making a prediction, SVM will use data's features to plot a point to high dimensional space, seeing which group the point belongs to.

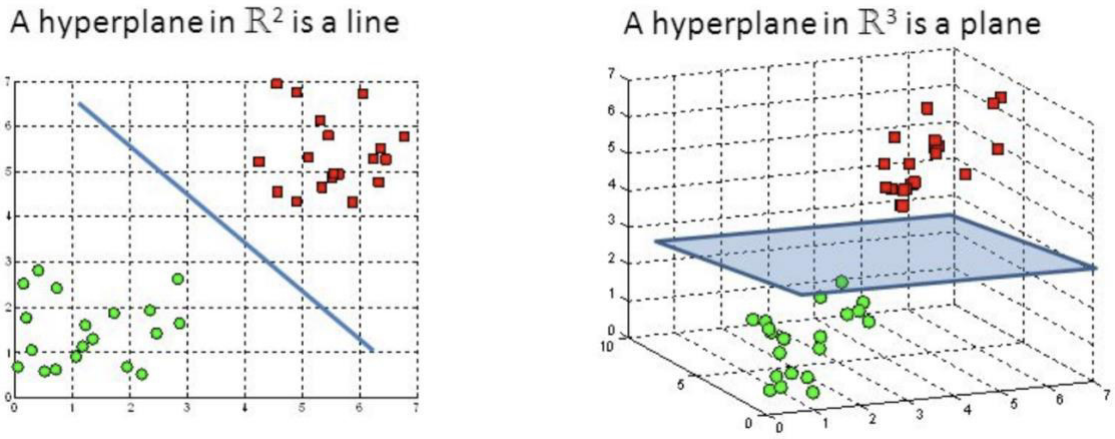


Figure 2.7 Examples of SVM plots in 2D and 3D space

2.2.2 BirdCLEF and DCASE

Both BirdCLEF and DCASE are highly recognised international competitions that occur regularly every year, in which participants develop machine learning algorithms that classify birds using audio. At the end of both challenges, the results are published describing participants' approaches, evaluating their models and discussing learned lessons. During the project, the results of both competitions are used to decide the approach for creation of the project's model.

2.2.3 Wildlife Acoustics

The biggest project's commercial competitor that can be found is Wildlife Acoustics, creating Song Meter devices (Wildlife Acoustics, 2022). They have a range of recorder devices in the price range between $249 and $849, which are waterproof devices that can be deployed in the wild to listen to animal sounds. These devices are expensive and have the capability to only record sounds, requiring additional software to make autonomous predictions on these sounds. A software license is sold separately called "Kaleidoscope Pro Analysis Software" (Wildlife Acoustics, 2020) by the same Wildlife Acoustics organisation, costing $399 per year. The software allows users to quickly label and identify bird songs, also being able to automatically suggest the most likely species. However, the automatic suggestion feature only works for bats currently, not being able to do for any of the birds. The project aims to replace the previously mentioned products from Wildlife Acoustics by providing inexpensive alternative devices that focus on automatic classifications of birds rather than manual classification, making the technology more accessible to everyone.

2.3 3D Modelling and Engineering Principles

2.3.1 Computer-aided Design

CAD is a technology that is used to create CAD models which are computer models defined by the geometrical variables. CAD models are typically in a three-dimensional representation, which can be easily modified by altering parameters using CAD software. These models are vector-based, allowing for them be easily viewed under a wide variety of representations such as enlarged and rotated. During the project, CAD software will be used to create precise 3D models for case solutions.

2.3.2 Cantilever

Cantilever is a very popular snap-fit joint that is used to join two objects together. This type of snap-fit joint consists of protrusion that looks like a hook, coming from one of the objects which is inserted into a slot from another object that deflects the protrusion on insertion. Once it has been fully inserted, the protrusion bends back to its normal position, locking the connection and securing the two objects together. Cantilever snap-fit joint will be used during the project to secure both parts of the case together.

2.3.3 Ingress Protection Rating

IP rating classifies the degree of protection provided by an enclosure for electrical equipment. This standard specifies different levels of sealing against intrusion of foreign bodies including dust, water and objects. The IP rating consists of two digits, where the higher the digit the better protection. The first digit defines the protection of the equipment within the enclosure against ingress of foreign objects (e.g., fingers) and against dust, both of which could damage the circuitry. The second digit indicates the level of protection the enclosure provides against liquids. DSM&T provides an IP Rating reference chart (DSM&T, 2015) that displays how much protection each digit provides. IP rating will be considered when designing case solutions for IoT edge devices.

2.3.4 Architecture Design Techniques to Control Water Movement

For the project's 3D case solutions, we took some inspirations from architecture designs to keep the water out of the openings within the created enclosure. In particular, we will be

exploring drip and overhang methods to control the water movement, refer to YR Architecture and Design for more information (YR Architecture Design, 2020).

Overhang

Water sliding on the surface can be diverted by creating an overhang above the opening, which is an edge protruding outward that provides protection for lower levels of the structure. This way, the water will slide towards the edge and at the end will be forced to fall away from the structure.

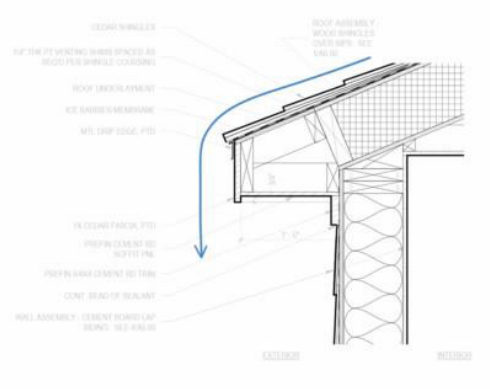


Figure 2.8 Visualisation of Overhang

Drip

Drip is another design technique to keep the water out of the opening. Drip technique is applied to the underside surface where water might slide to and cling depending on the runoff speed, later being drawn into the opening. Drip design technique solves the issue by having a break within the underside surface which causes to reduce water surface tension and cause the water to drop down.

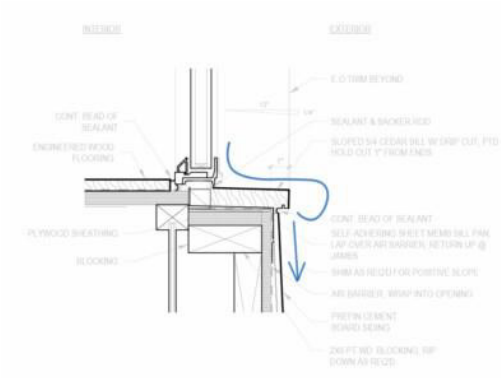


Figure 2.9 Visualisation of Drip

2.4 Bird Vocalisation

The project will analyse and understand how bird vocalisation functions to ensure model generalisation.

2.4.1 Bird Calls vs Songs

Bird calls and songs are birds' vocalisations that differ by the length, complexity and context. Bird songs tend to be longer and much more complex which are used for mating purposes. Comparing to the calls, that are short in duration and much simpler in structure, used for various functions like keeping members of flock in contact or alarms.

2.4.2 Inter-species Variance

As the project is working with bird sounds, it is important to account for inter-species variance. The same bird species in different locations might sound different, which could affect model's performance. Birds work just like humans, in which species communicate differently depending on the country they live in.

2.4.3 Audio Frequency

The audio frequency is measured in hertz, which computes the number of times per second the sound wave's cycle repeats. The greater the frequency, the higher the pitch humans perceive. Audio frequency is calculated by dividing the velocity which is the wave speed by the wavelength which is the distance of one frequency wave peak to the other.

$$\boldsymbol{Frequency = \frac{Velocity}{Wavelength}}$$

$$\boldsymbol{f = \frac{v}{\lambda}}$$

$$\boldsymbol{Unit\ of\ Frequency: Hz}$$

Figure 2.10 Frequency Formula

2.5 Tools and Methods

2.5.1 Moving Average Filter

Moving average filter is calculated by averaging a number of input signals to produce one single output signal as shown on Figure 2.11 where x is the input signal, y is the output signal and M is the number of signals used in the average (Analog, 2022). The moving average filter is really good for reducing random noise and retaining fast response. This filter will be used on the project's model classification predictions to reduce number of false positives produced by the model.

$$y[i] = \frac{1}{M} \sum_{j=0}^{M-1} x[i+j]$$

Figure 2.11 Moving Average Formula Equation

2.5.2 Xeno Canto

Xeno Canto is an online repository where volunteers upload and annotate recordings of bird calls and songs found all around the world. Within the project, we have used this repository to download bird sounds and use them to train our model.

2.5.3 BBC Sound Effects

BBC Sound Effects is an online sound library consisting of over 33,000 various clips of audio found across the world collected over the past 100 years. During the project we used the library to collect environmental sounds which were then used to train the model.

2.5.4 GrabCAD Community

GrabCAD Community is a platform managed by the community of 7 million members where members can share and download CAD files. Within the project, we have used the platform

to download 3D models of Raspberry Pi 4 Model B (Kaparykha, 2021) and Arduino Nano 33 BLE Sense (Peterson, 2021), which were used to design and develop 3D case solutions.

2.5.5 Edge Impulse

Edge Impulse is a development platform for machine learning on edge devices. The platform allows users to collect data by integrating data sources using open APIs, design models and its parameters, test model's performance and finally deploy the developed model to edge devices. During the project, we used the platform just for that, allowing to develop the machine learning model from scratch and then deploy it to both Raspberry Pi and Arduino solutions.

2.6 Technologies

2.6.1 LoRa

LoRa technology was developed by the chip manufacturer called Semtech (Semtech, 2022), which stands for Long Range Radio and is targeted mainly towards IoT networks. LoRa is a wireless protocol that allows for long-range and low-power communications. LoRa provides a way of transmitting radio signals of small byte messages over 10-mile distances. The project uses LoRa technology for transmitting results from edge devices installed remotely in the wild all the way to the gateway device.

2.6.2 Grove Connectors

Grove is a standardised connector system developed by Seeed Studio (Seeed Studio, 2021). Grove uses building blocks approach to connecting hardware together instead of soldering, not requiring any tools to assemble electronics making it easier and quicker to prototype systems. For the project, Grove connectors are used to connect LoRa transceiver to the Arduino device.

2.7 Constraints

The project had a fair share of imposed constraints which we had to work around. Time was one of the biggest limiting factors, having to do so much in the project required careful time management to ensure that all of the critical parts of the system are implemented before the deadline.

Another big constraint the project had was the limited amount of control over the created machine learning model. Edge Impulse was used as a tool to create the model, which had appealing ease of use but had limitations in data management, hyperparameter customisation, model design and more, forcing to create a model which had to be supported by the Edge Impulse platform.

One more constraint the project was imposed on was personal understanding of machine learning and C++ programming language. Both were very new to me and had a steep learning curve due to being complex in nature. Both areas of knowledge were very important to the project due to being extensively used within, requiring me to learn the fundamentals before we could tackle the problem.

Lastly, the project was constrained to the ordered hardware. On choosing the hardware at the start of the project, we were forced to utilise only a limited set of hardware's supported libraries and tools. As will be discussed later in the project, this has posed a big problem on the project, which could not have been counteracted by ordering alternative hardware that would have the supported libraries and tools due to time implications.

2.8 Research Question

In order to achieve the stated aims, the project is required to identify what technologies will be used and design a comprehensive architecture that utilises outlined technologies to create a foundation for the mentioned aims, identify what successful machine learning techniques competitors use to classify birds with high accuracy, identify what classes the model will classify and collect all of the required training data, develop machine learning model and demonstrate how it has similar performance to competitors, implement frameworks that will make use of both machine learning model and technologies to perform the outlined aims and finally identify environmental hazards to the architecture and compose a case solution for the hardware.

3 Approach

The project's overall aim is to replace one of the already mentioned recording devices on the commercial market with an inexpensive IoT edge device that will capture and classify birds using sound.

3.1 IoT Architecture

The first objective is to design hardware architecture for edge device on two different platforms: Arduino and Raspberry Pi. Both solutions are required to take audio inputs, transmit messages wirelessly, ability to run machine learning model and be battery powered. As well, we extended the objectives with another IoT architecture called gateway device, that would read the transmitted messages and store them locally, ready for retrieval.

As can be seen on the Figure 3.1, the architecture starts at the two different edge solutions, Raspberry Pi and Arduino which will be deployed in the wild. Both have identical architecture but with different hardware. They consist of Arduino microcontroller or Raspberry Pi single-board computer themselves, which are responsible for running the machine learning model and IoT framework. Both use microphone for audio input to listen to environment sounds. As they will be deployed remotely, they will have their own rechargeable battery module. Lastly, LoRa protocol was chosen for wireless communication because of its power efficiency and wide coverage range of more than 10 miles in rural areas and up to 3 miles in dense urban environments (LoRa, 2019). Both edge devices use LoRa transceiver that will send classification results as radio messages to the gateway. As for the gateway architecture, it consists of Raspberry Pi that will be used to listen to incoming radio messages by utilising LoRa transceiver and then saving these messages locally on SD card. The gateway device will be connected to the same network as the deployer's computer machine, allowing for the deployer to connect to the gateway using SSH at any given time to retrieve classification results stored on the gateway.

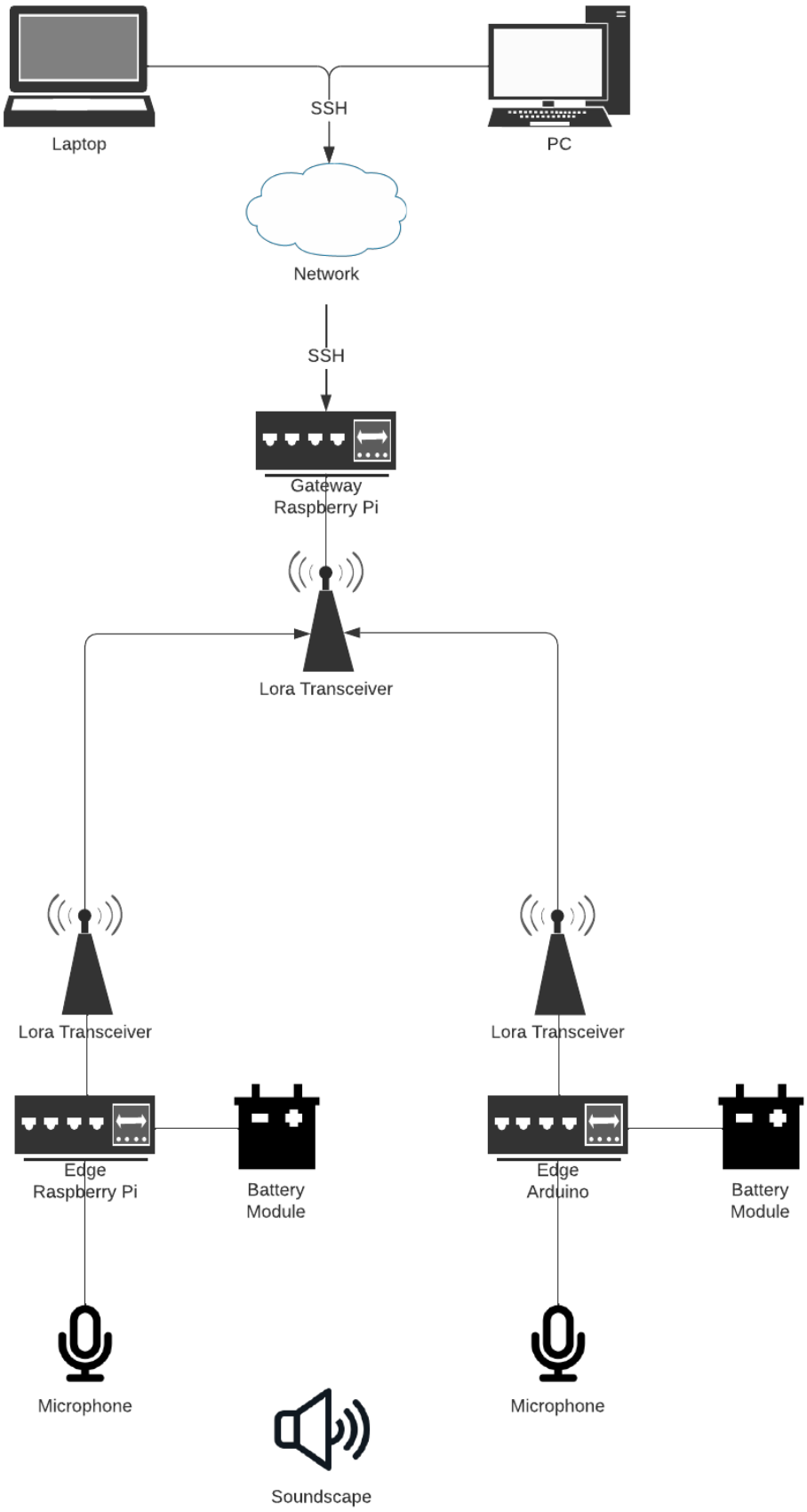


Figure 3.1 IoT Architecture Component Diagram

A list of hardware ordered for the IoT architecture can be seen on Figure 3.2. For edge Raspberry Pi device, it has been decided for it to have a slightly larger 4GB RAM capacity, extra memory for machine learning inference. For edge Arduino device, we decided to use Grove based connectors to connect LoRa transceiver to the Arduino through Grove Shield, due to simplicity and no necessity for soldering. In terms of gateway Raspberry Pi device, it was decided to have a larger 64 GB SD card to store all results on and have a power supply to take infinite supply of power from the socket instead of being battery powered, delivering no downtime. Also, all of the devices have a LoRa transceiver with the same 433 MHz frequency, to allow them to communicate with each other.

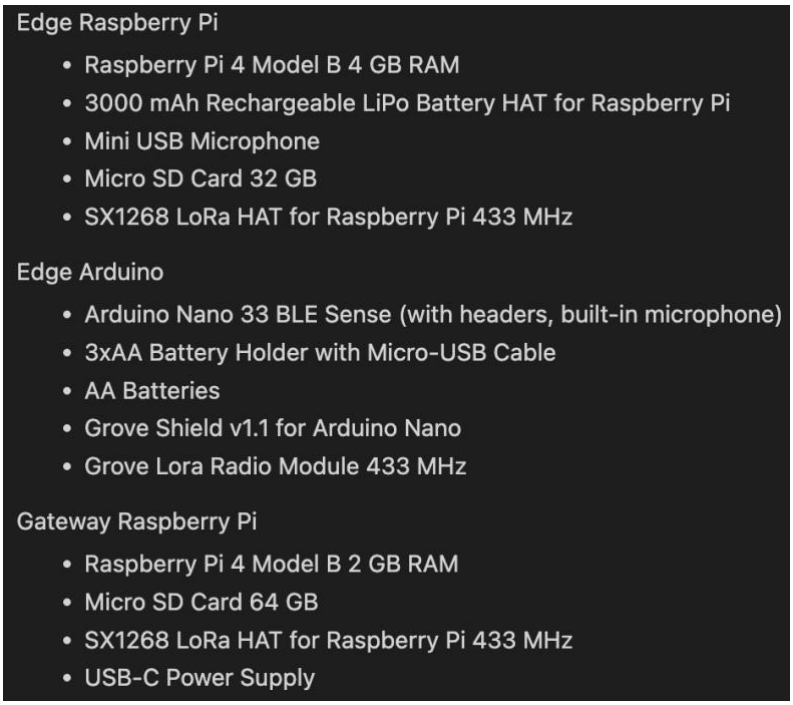


Figure 3.2 List of ordered hardware

3.2 IoT Frameworks

Four different frameworks need to be designed that provide a set of functionalities on IoT architectures.

3.2.1 Edge Framework

Another objective of the project is to be able to run machine learning model inference on IoT edge devices. IoT edge framework is required to take audio input, extract features, feed it to the model for inference, transmit results to the gateway device and store inferred audio locally.

On booting up, the framework starts taking audio inputs straight away, on condition that machine learning model is present and microphone input device is available. The audio is then passed to the machine learning's library for feature extraction. Next the library feeds the features to the model for inference, returning classification results. The audio collection process and model's inference process both are done concurrently, meaning that no audio will be missed during classification. The audio returned by the model is saved locally instead of sending over LoRa due to LoRa's limited 256 bytes message size as can be seen on Figure 3.3. On specified confidence threshold, the results with metadata are wirelessly transmitted

over the LoRa protocol, being converted into binary representation just before. The following results are sent over LoRa: audio file name which will contain path and name of the audio file stored locally on edge device, confidence level which is how confident the model is on the classification between 0.0 and 1.0, classification which is the bird's name the model predicted and date with time which was captured in ISO 8601 (ISO, 2022) format. ISO 8601 was chosen due to being global standard for time and date formats, being clearly understandable by both machines and humans.

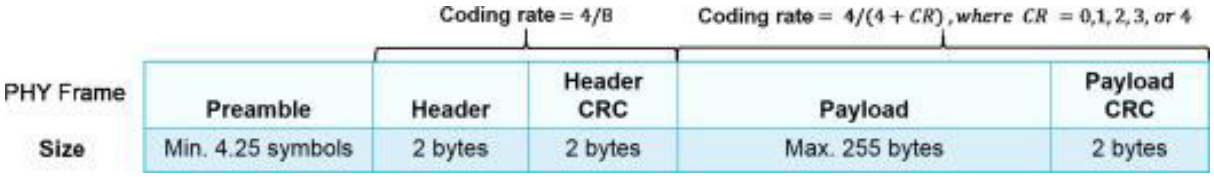


Figure 3.3 LoRa packet size (Pham, et al., 2020)

3.2.2 Gateway Framework

For gateway device, the requirements are for it to listen to edge devices' results transmissions and store them locally.

On boot up, the framework starts constantly listening to LoRa messages. On receiving a LoRa message, the framework converts the message from binary into its original ascii format. After which all of the data is stored within the SQLite database, that has a singular table called "classification_results". The reason for choosing SQLite for storage is because it is energy efficient compared to larger-scale database systems and at the same time has rich ability to query data compared to storing in a text file such as CSV format. As the framework is connected to the same network as the deployer, deployer using SSH can connect to the gateway with CLI for more interactive functionalities as shown on Figure 3.4. The first two functionalities allow the user to list all collected classifications and classifications by specified label respectively. The next functionality allows user to export all of the stored results into the CSV file, requiring user to specify the file name data will be exported to. Another function is to start listening to results which was explained previously. Lastly, gateway framework gives an option to download audio files stored on the edge device, only when that edge device is connected to the same network as the gateway device. The command will go through the local database to retrieve all audio file names, after which will download sound files from the edge device using SSH into the specified directory. This functionality was introduced in response to limitations with LoRa's maximum packet size as discussed in the previous section.

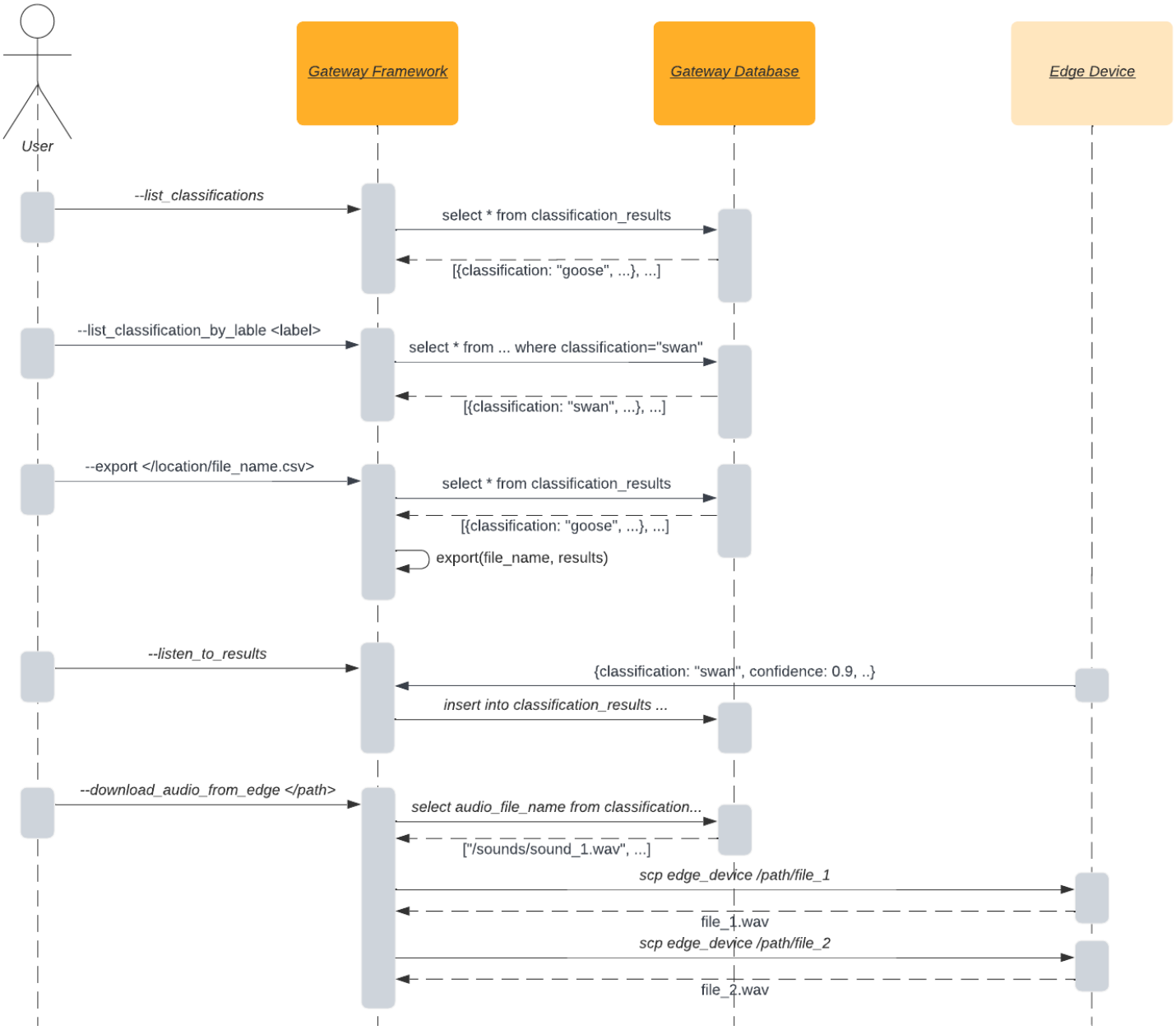


Figure 3.4 Gateway CLI Functionality Sequence Diagram

3.2.3 Microsoft Framework

For evaluation purposes, we are required to deploy Microsoft's acoustic bird detection model on both Arduino and Raspberry Pi devices. Similarly to the edge framework, Microsoft framework takes audio input, extracts features, feeds into Microsoft's model for inference and then prints out the results.

3.3 Machine Learning Model

Another project's objective is to create a machine learning model that would be able to classify bird sounds, being able to consistently distinguish certain types of birds in real world environments where various background noises are present.

As we want to distinguish various birds, this is a multiclass classification type of problem. Reviewing results of the BirdCLEF (Kahl, et al., 2021) and DCASE (DCASE, 2021) that also tackle the same type of problem, gave a lot of insight on how to design project's model. The results showed that all winners and most teams in general used Convolutional Neural Network based models, many used data augmentation techniques and there was a split between using MFE and Spectrograms for feature extraction. Therefore, it has been decided to choose Convolutional Neural Network based model and experiment with MFE and Spectrograms as feature extraction techniques when working on the project's model. As for splitting the dataset, it is split into 60% training set used to change model's neurons, 20% validation set used to estimate model skill and to tune hyperparameters and 20% test set used to give an unbiased estimate of skill of the final tuned model, a ratio of 3:1:1 which is the general rule of thumb.

The model has three different categories of classes it will capture. The first and main category of classes is target sounds, these classes are birds that we are interested in classifying, that include bird sounds with data augmentation applied. The next class category is called noises which comprises of environment sounds found in the habitat of the target birds. This category is introduced to ensure the model is aware of what non-bird sounds are. Lastly, unknown class category is created that includes sounds of all kinds of birds which do not include target birds. This is made so the model can differentiate different birds and ensure it does not classify random birds as targets.

In terms of collecting data, dataset needs to be very diverse and representative of real-world settings. As highlighted by the "Audio Based Bird Species Identification using Deep Learning Techniques" paper (Sprengel, et al., 2016), there are many challenges with classifying birds by sound. These include having background noises when capturing bird sounds, birds can make calls and songs which are different by former being shorter and latter being longer, multiple birds singing at the same time and inter-species variance. Therefore, to combat all of these challenges, different examples of bird songs and calls captured in the natural environment are collected over a number of years, conditions and regions. Furthermore, target bird sounds' dataset is enhanced with data augmentation, using some of the successful techniques outlined by the BirdCLEF with DCASE and previously mentioned paper to improve generalisation performance of the system. These techniques include time shift to avoid overfitting, adding noise to improve generalisation, small 5% pitch shift and speed change to account for the inter-species variance. At the same time, the collected dataset needs to be balanced to ensure the dataset is not biased or skewed that imbalanced dataset suffers from (Brownlee, 2019).

3.4 3D Case Solutions

The last objective of the project is to have the ability to deploy the IoT edge devices in the wild environment. For this, an enclosed 3D case solution for both types of edge devices need to be designed that is required to be weatherproof, securely capture all hardware inside, provide openings for the microphone and support different kinds of installations on various surface areas such as tree branches and bushes.

Both 3D case solutions are designed to achieve IP23 rating which provides protection against solid objects greater than 12.5mm in diameter such as a finger and protection from

vertical sprays of water up to 60 degrees. They also both follow the same design principles, where everything starts at the base of the case. The base captures all of the hardware and has four mounting points for Raspberry Pi and two for Arduino with screw holes for 3 mm wide and 5 mm long screws. At the bottom and outside of the base, there are additional four screw holes for 3 mm screws and two zip tie mounting points that support up to 7.6 mm sized zip ties (Hont, 2022) designed for installation purposes. A lid for the base is designed to enclose the case, which is connected with a snap-fit joint called cantilever (Bayer, 2020). The reason for choosing cantilever snap-fit joint is due to being time-saving and low-cost connection method because of their simple design that also supports rapid assembly and disassembly. However, cantilever snap-fit joint is not waterproof on its own hence both cases require water seal design. The top of the base is designed to have O-ring groves for 2mm sized O-ring which is used to seal the case from water when the top snaps on and compresses rubber. Furthermore, case solutions have small openings that allow microphones to capture sounds from environment, that slightly differ by the case. For Raspberry Pi case, a rectangular opening is used to allow insertion of USB microphone which is then surrounded by walls. These walls use drip design technique to relieve water pressure and cause water to drip instead of gliding into the microphone opening. Case for Raspberry Pi can be installed only one way down, having microphone opening pointing downwards to ensure the water comes from the top only. For Arduino case, as the microphone is built into the microcontroller, small holes are designed at the top part of the case on the lid, where Arduino itself will be located at. To make it waterproof, a small roof on top of the holes is created that uses overhang design technique to force the water to fall away from the surface of the case. For this reason, the Arduino case can only be installed vertically, allowing roof to protect the openings from rainfall falling from the sky.

In terms of hardware assembly, for Raspberry Pi it is very simple because all of the components are HATs that snap on top of the Raspberry Pi and do not require any further mounting, leaving the Raspberry Pi to be screwed in to the previously mentioned four mounting holes. For Arduino assembly, all components apart from the shield are separate and only battery holder has screw holes. Therefore, only the battery holder is screwed into previously mentioned case's mounting points and Arduino connected to Grove Shield is placed on top, being supported by the battery holder from the bottom. The lid will be then snapped on which will touch the Arduino, providing support from the top. To make sure that Arduino and Grove Shield stay in place and have horizontal support, the lid has four extended walls around the shield to hold it in place and further four alignment pillars that go through four holes in Arduino to give further horizontal support.

4 Implementation

This section discusses the implementation of the approach previously described. Before starting any of the implementations, a wiki document was created for every implementation with the design explaining what and how something is going to be implemented. As well, a highly detailed README file was created providing documentation on how to install frameworks, how to deploy machine learning model, device configurations and more. After implementing features, they were pushed to the GitHub repository for version control which can be found at Appendix A – GitHub Repository for Final Year Project.

4.1 Designing IoT Architecture

The first iteration of the IoT architecture did not include the gateway device which listened to edge devices for results. Initially, the project only required edge devices to send the results wirelessly with an assumption a PC is able to read the signals. However, due to choosing LoRa as a wireless communication protocol, a LoRa transceiver was required to be purchased to allow to listen to LoRa communications as normal PCs cannot listen to LoRa transmissions. However, due to lack of affordable LoRa transceivers for PCs, it has been decided to extend the project to include a gateway device that would listen to edge devices' messages. Also, this made the project a lot more practical as it no longer required PC to be turned on actively to ensure that no messages will be missed, handing down the task to a more power efficient IoT device that can be powered 24/7.

Another problem occurred with the IoT architecture was when we received the hardware. Wrong v1.0 version of the Grove Shield was received which was not compatible with Arduino Nano 33 BLE Sense due to not supporting 5 V signals (Arduino, 2022) which v1.0 Grove Shield operates at. Furthermore, the commercial market had no v1.1 Grove Shields available that has 3.3 V power mode. Therefore, due to time constraints it has been decided to manually change the Grove Shield to operate on 3.3 V by soldering the power cable of LoRa transceiver to 3.3 V pin on Grove Shield instead of taking power from 5 V pin.

4.2 Implementing IoT Frameworks

The edge framework for Arduino has been developed in C++ and all gateway, Microsoft and edge frameworks for Raspberry Pi were developed in Python. To make the development process easy, we connected to the Raspberry Pi using SSH and VNC which allowed to implement IoT framework from the laptop remotely, not requiring any physical connection. Developing IoT framework on Arduino required a USB connection to the laptop, through which code was uploaded to the Arduino and then using serial monitor to see the framework's output running on Arduino device for debugging purposes. For all frameworks, I ensured the code has some exception handling and help messaging, to ensure that either of frameworks are used correctly. An example of error handling can be seen on Figure 4.1, a piece of code from gateway's framework that handles wrong and empty inputs from user when running the library. We also followed good programming practices when writing frameworks, so they are easy to maintain in the future. The practices included commenting complicated pieces of code, following DRY principle, consistent naming and more.

```python
def main(argv):
    try:
        # For more information about opts - https://docs.python.org/3/library/getopt.html
        opts, args = getopt.getopt(argv, "h", ["help", "list_classifications",
        "list_classifications_by_label=", "export=", "listen_to_results",
        "download_audio_from_edge="])
    except getopt.GetoptError:
        # On passed in arguments not being in the allowed list of args.
        help()
        sys.exit(2)

    for opt, arg in opts:
        if opt in ('-h', '--help'):
            help()
            sys.exit()
        elif opt in ('--list_classifications'):
            list_classifications()
        elif opt in ('--list_classifications_by_label'):
            list_classifications_by_label(arg)
        elif opt in ('--export'):
            export(arg)
        elif opt in ('--listen_to_results'):
            listen_to_results()
        elif opt in ('--download_audio_from_edge'):
            download_audio_from_edge(arg)
        else:
            assert False, "unhandled option"

    if len(opts) == 0:
        help()
        sys.exit(2)
```

Figure 4.1 Error handling example in Gateway Framework

4.2.1 Edge Framework

For both frameworks, I started from examples provided by Edge Impulses' documentation (Edge Impulse, 2021) and extended the implementation of them further.

For Raspberry Pi's version of the framework, two arguments are required to run the framework. The first is path to the .eim file which is the learning model's file that can be exported from Edge Impulse. The second argument is device id for the microphone, with which audio collection will be carried out. Edge Impulse supplies Python SDK (Edge Impulse, 2021) which is used by the framework that provides API functions to collect audio through microphone and run inference on the model. On running the framework, this library would be initialised and start taking audio samples and run inference concurrently. The library uses a buffer to continuously fill audio in two second increments. While there is data in that buffer, library will generate features and pass them into the model, returning classification for that two second audio and the audio itself. Originally, the library returned the extracted features instead of audio as a result. We had to change the library's code to return audio instead, by tapping into audio recorder function within the provided SDK. For each classification returned by the model as seen on Figure 4.2, prediction with the highest confidence is chosen. On condition of the predicted class has confidence over or equal to the confidence threshold and if prediction's label does not start with underscore (i.e. not

capturing generic classes such as _noise), then the result is sent to the gateway. Right before sending the classification results, audio is saved locally using predicted label and time as filename to ensure uniqueness. After creating a result dictionary that will be sent over LoRa, a lightweight asynchronous process is started in the background for sending the results over LoRa. The process will finish at its own pace without blocking audio collection and model inference processes.

```
for res, features in runner.classifier(device_id=selected_device_id):
    print_classification_result(res, labels)

    predictions = res['result']['classification']
    prediction_label, prediction_confidence = highest_prediction(predictions)
    threshold = 0.8

    if prediction_confidence >= threshold and not prediction_label.startswith("_"):
        print("Classified " + prediction_label + " at confidence level of " +
            str(round(prediction_confidence, 2)))
        current_time = datetime.datetime.now().astimezone().isoformat()
        frequency = model_info['model_parameters']['frequency']
        filename = save_audio(features, prediction_label, current_time, frequency)

        res = {
                "filename": filename,
                "confidence_level": prediction_confidence,
                "classification": prediction_label,
                "time": current_time
        }

        process = multiprocessing.Process(target=send_lora_message, args=(node, res))
        process.start()
        all_processes.append(process)
```

Figure 4.2 Classification Handler in Edge Framework

Looking at Figure 4.3, the asynchronous send LoRa message function starts off by converting the results dictionary into JSON and then further into binary. It then creates metadata required by LoRa to send to the correct node at the right frequency. Gateway's address being 0 and its own address being 100, the function specifies the high and low 8-bit address for both addresses, as well as the offset frequency. Alongside with metadata, the results in binary representation will be sent as LoRa message using the sx126x library provided by the manufacturer (Waveshare, 2020).

```python
def send_lora_message(node, payload):
    # Starts a process that will send result as Lora message over 433MHz to node with address 0.

    # Converts result into json string and encodes into binary.
    binary_payload = json.dumps(payload).encode('utf-8')

    # Splitting address into two 8 bit numbers.
    receiving_node_high_8bit_address = bytes([0>>8])
    receiving_node_low_8bit_address = bytes([0&0xff])
    receiving_offset_frequency = bytes([433 - 410])

    own_high_8bit_address = bytes([100>>8])
    own_low_8bit_address = bytes([100&0xff])
    own_offset_frequency = bytes([node.offset_freq])

    data = receiving_node_high_8bit_address + receiving_node_low_8bit_address + \
        receiving_offset_frequency + own_high_8bit_address + own_low_8bit_address + \
        own_offset_frequency + binary_payload
    node.send(data)
    print("Sent Lora message.")
```

Figure 4.3 Send LoRa Message Function in Edge Framework

We also guarantee a safe exit when user does an interrupt using “CTRL + C” combination. To make sure that no processes are left in the background after the exit, the model and all spawned processes that send LoRa messages are stopped before terminating the session.

```python
def signal_handler(sig, frame):
    # On interrupting, will shut down the model and exit the program.
    print('Interrupted')
    if (runner):
        runner.stop()

    print("Stopping all processes")
    for process in all_processes:
        process.terminate()
    print("All processes have been terminated")
    sys.exit(0)
```

Figure 4.4 Exit Handler in Edge Framework

In terms of the initialisation, it has been decided to execute the framework as Python file during boot up. The Python script execution is added to the bashrc file which will start running the script on every shell initialisation. Raspberry Pi’s configuration is also altered to run a shell session on start-up, which will in turn start the framework.

As for the Arduino edge framework, the framework relies on the Arduino compiled model file that is exported from Edge Impulse. The model exported will be EON Compiled (Jongboom, 2020), optimising the model by compiling the model to C++ source code, resulting in up to 55% less RAM and 35% less ROM usage whilst having the same model accuracy. As with any Arduino library, everything including inference task and buffers are initialised within the setup block, as well as starting the audio sampling process. Then as

could be seen on Figure 4.5, within the loop block a buffer with audio is collected, which is inferenced using the model and results are printed, whilst ensuring that tasks are ready to run and errors are handled appropriately. Inference process is run in parallel to audio collection process, both being run as separate threads ensuring that sound is still being collected during the inference. The inference process calls “run_classifier_continuous” function from Edge Impulse’s C++ SDK (Edge Impulse, 2021) to extract features and do the inference, passing on audio slices to it. The function has its own time sequential FIFO (First In First Out) buffer which is filled with audio slices. After each iteration the oldest audio slice is removed from the buffer and new one is inserted at the beginning, resulting in audio slice being inferenced multiple times improving the accuracy. This introduces moving average filter, filtering out false positives.

```
void loop()
{
    // Checks if buffers are ready to record.
    bool m = microphone_inference_record();
    if (!m) {
        ei_printf("ERR: Failed to record audio...\n");
        return;
    }

    signal_t signal;
    signal.total_length = EI_CLASSIFIER_SLICE_SIZE;
    // Gets right buffer with full of data.
    signal.get_data = &microphone_audio_signal_get_data;
    ei_impulse_result_t result = {0};

    // Starts running inference.
    EI_IMPULSE_ERROR r = run_classifier_continuous(&signal, &result, debug_nn);
    if (r != EI_IMPULSE_OK) {
        ei_printf("ERR: Failed to run classifier (%d)\n", r);
        return;
    }

    if (++print_results >= (EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW)) {
        // print the predictions
        ei_printf("Predictions ");
        ei_printf("(DSP: %d ms., Classification: %d ms., Anomaly: %d ms.)",
            result.timing.dsp, result.timing.classification, result.timing.anomaly);
        ei_printf(": \n");
        for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
            ei_printf("Result: %d", result.classification[ix].label);
            ei_printf("    %s: %.5f\n", result.classification[ix].label,
                      result.classification[ix].value);
        }
```

Figure 4.5 Loop Block in Edge Framework

Within this framework, the audio collection is done manually. Two buffers are going to be used to store audio data, one is used by the inference process to extract features and run inference on, and another one used by audio sampling process for filling the buffer with new audio data. Both buffers will be switched between. When the sampling buffer becomes

full of audio slices, the process will pass full buffer to the inference process and then will clear and fill up the old buffer with new audio. This can be seen on Figure 4.6, when buffer becomes full after enough of audio collection, the buffers are switched by “inference.buf_select” and marks the old buffer as ready to inference by “inference.buf_ready”.

```
static void pdm_data_ready_inference_callback(void)
{
    // Callback which is used by Audio Sampling process to switch buffers
    // when enough data is collected.
    int bytesAvailable = PDM.available();

    // read into the sample buffer
    int bytesRead = PDM.read((char *)&sampleBuffer[0], bytesAvailable);

    if (record_ready == true) {
        for (int i = 0; i<bytesRead>> 1; i++) {
            inference.buffers[inference.buf_select][inference.buf_count++] = sampleBuffer[i];

            if (inference.buf_count >= inference.n_samples) {
                inference.buf_select ^= 1;
                inference.buf_count = 0;
                inference.buf_ready = 1;
            }
        }
    }
}
```

Figure 4.6 Function to switch buffers in Edge Framework

Afterwards, the inference process selects that full of audio buffer before doing the inference, this could be seen on Figure 4.7.

```
static int microphone_audio_signal_get_data(size_t offset, size_t length, float *out_ptr)
{
    numpy::int16_to_float(&inference.buffers[inference.buf_select ^ 1][offset], out_ptr, length);

    return 0;
}
```

Figure 4.7 Select Inference Buffer Function in Edge Framework

There has been some difficulty at sending results over LoRa using LoRa transceiver on Arduino edge framework. The manufacturer of LoRa transceiver does provide a library to send LoRa messages, however it does not support the chosen Arduino device. As well, there are a few general use libraries that supports common data radio protocols including LoRa on a range of microprocessors. However, because of Nano 33 BLE Sense being relatively new device with a completely different approach to other Arduino boards using Mbed OS, neither of libraries work on the device. Due to time constraints, it has been decided to drop the ability to send LoRa messages on Arduino edge framework for the time being.

4.2.2 Gateway Framework

Gateway framework has a few interesting implementations. Firstly, listen to results function (can be seen on Figure 4.8) which is used to listen to LoRa messages. The function starts off by initialising LoRa transceiver at 433 MHz and address 0 using the same sx126x library, meaning that it will only read messages sent at that frequency and to that address. Next, in

the infinite while loop, the framework will continuously listen to LoRa messages and on receiving, it will decode the message from binary to utf-8 format and insert it into the database. The function also handles exceptions by ignoring broken messages received from the edge device, which could have happened due to packet loss or for any other reasons.

```
def listen_to_results():
    # Listens to Lora messages and saves them to the database.
    address = 0
    node = sx126x.sx126x(serial_num="/dev/ttyS0", freq=433, addr=address,
        power=22, rssi=False, air_speed=2400, relay=False)
    print(f'Listening to Lora messages at address {address}')

    while True:
        result = node.receive()
        if result != None:
            try:
                decoded_result = json.loads(result.decode('utf-8'))
                insert_result(decoded_result)
                print(f'Saved result - {decoded_result}')
            except:
                # In case of packet loss or mixed packets.
                print(f'Packet loss - {result}')
```

Figure 4.8 Listen to results Function in Gateway Framework

As for the rest of the functions, they all use SQL to manipulate data in the database. As an example, “export/1” function that exports data within the database into CSV file (can be seen on Figure 4.9), it first creates a database connection and then selects all data from the table using a cursor. After the function is done working with data, it closes the connection with the database.

```
def export(path):
    dbconnect, cursor = connect_to_database()
    data = cursor.execute('SELECT * FROM classification_results')
    column_names = [desc[0] for desc in data.description]

    with open(path, "w") as file:
        writer = csv.writer(file)
        writer.writerow(column_names)
        for row in data:
            writer.writerow(row)

    dbconnect.close();
    print(f"Successfully exported to - {path}")
```

Figure 4.9 Export Function in Gateway Framework

In terms of the initialisation, it works the same as edge framework for Raspberry Pi, executing the framework as Python script during boot up, calling framework's listen to results function.

4.2.3 Microsoft Framework

For this framework, two external files are required to run the framework which are the feature model that extracts features and SVM classifier that performs the classifications, both need to be exported from the Microsoft's acoustic repository. As the framework is for evaluation purposes, the tasks will be run synchronously meaning it will not start inference whilst recording audio. For audio recording part of the framework, a wav file is created or overwritten with the captured audio. The feature extraction function as seen on Figure 4.10, then reads the audio file and splits the audio in two second clips. After, the framework pads clips that are less than 2 seconds with zeros, making sure that model gets features with the correct dimensions. Lastly, for each clip Mel-frequency spectrograms are generated which are then returned as features by the function.

```
def convertAudioToFeatures():
    # Reading sounds
    wav, fs = soundfile.read('./sound.wav')
    window_len = 2 # seconds
    samples_per_window = fs * window_len
    curr_data = None

    # generate all 2 seconds windows as a list, then round down
    # the label start time to the nearest 2 second window.
    all_windows = np.arange(0, np.ceil(len(wav) / samples_per_window).astype(np.int))

    for w in all_windows:
        start = w * samples_per_window
        end = start + samples_per_window
        length = end - start

        window = wav[start:end]
        if (not len(window) == length):
            padding = length - len(window)
            window = np.pad(window, (0, padding), 'constant')

        if curr_data is None:
            curr_data = np.array([window])
        else:
            curr_data = np.append(curr_data, [window], axis=0)
    fbank_import = np.array([cmi.make_fbank(x) for x in curr_data])
    return fbank_import
```

Figure 4.10 Feature Extraction Function in Microsoft Framework

The features are then passed to the inference function (is shown on Figure 4.11), which first normalizes input to be in line with what the pre-trained model expects by scaling the features and dropping the final frequency bin. Afterwards, feature model and SVM classifier are loaded and utilised to return classifications results, the former being used to decompose audio spectrograms into feature vector and the latter being used to carry out the inference.

```
def runInference(fbank):
    scale = 33.15998
    features_normal = fbank[:,:40,:] / scale
    # reshape the batches for the model.
    normal_batch = features_normal.reshape(features_normal.shape[0],
                                           features_normal.shape[1],
                                           features_normal.shape[2],
                                           1)
    # Importing SVM classifier and feature model.
    imported_svm = joblib.load('./svm.pkl')
    imported_model = tf.keras.models.load_model('./microsoft_model')

    # Carrying out inference.
    features = imported_model(normal_batch)
    results = imported_svm.predict(features[-1])
    return results
```

Figure 4.11 Inference Function in Microsoft Framework

4.3 Training Machine Learning Model

Edge Impulse development platform was used to develop and train the machine learning model. For data collection, Xeno-Canto was used to accumulate bird samples and BBC Sound Effects was used to collect environment sounds. For the model we chose to have only one target bird called Araripe Manakin, this is because the project aims to demonstrate proof-of-concept model that can be trained further if required. Araripe Manakin is one of the rarest birds in the world found in Brazil, with only a handful of audio samples available. Due to target bird's song and call duration being slightly longer than a second, the two second window size for the model was chosen.

The training procedure in Edge Impulse consisted of first importing all of the audio samples into the repository. Then, split and crop tools provided by Edge Impulse were used to split the imported audio into two second samples. Afterwards, we generated features for all collected audio, right before which the audio was up or down sampled to the chosen audio frequency. After feature generation, we trained the model and viewed results. Before next training iteration, we changed model's hyperparameters to try to improve model's performance.

The very first dataset for the model consisted of two classes. First, "araripe_manakin" which contained the whole 6 minutes of all bird's sound samples available on Xeno-Canto. The second class called "_noise" which consisted of 6 minutes of random environment sounds found in the wild; 1 minute of stream/waterfall, 2 minutes of forest, 1 minute of wind, 1 minute of village and 1 minute of people talking. In terms of input parameters, the highest audio frequency 44100 Hz was chosen to ensure that no audio would be down sampled and lose quality before feature extraction. For feature extraction, MFE was selected just for the first iteration. As for the model, 1D Convolutional Neural Network model was chosen due to having small number of classes within the dataset and to start off with the simplest model as possible. The training results returned 100% accuracy with 0 loss, which is a clear

indication of overfitting. On deploying and testing in real world, model could not distinguish any birds from Araripe Manakin, classifying all birds as the target class.

The next iteration introduced a new class to the dataset called "_unknown" which included 6 minutes of many random birds singing together. However, that still resulted in the same performance and was classifying silence as the Araripe Manakin class. The model was overfitting this time due to the issue within the dataset, the unknown and sound classes were busy and had sounds playing constantly, whereas the target class had a lot of breaks with silence.

For the next iteration, it was decided to increase the dataset by 50% and add more varied and quieter data to the unknown and noise classes. For the noise class, 2 minutes of silence and static noises were collected from BBC Sound Effects and additional 1 minute of random crackles and bangs sounds were collected manually. For unknown class, six random birds were selected that made calls and songs on their own just like in Araripe Manakin class, these were; Boat-tailed Grackle, Mute Swan, Song Sparrow, Tricolored Heron, Common Blackbird and Ocellated Crake, each 30 seconds long. Due to limited available data for Araripe Manakin class, we had to perform oversampling to ensure the dataset is kept balanced. Using random oversampling technique, we duplicated half of the Araripe Manakin class. The training performance dropped to 95.7% and 0.33 loss which had less signs of overfitting.

For the next few iterations, we have experimented with feature extraction methods, changing between MFE, Spectrogram and MFCC. MFCC performed best, with training performance being slightly lower 93% accuracy but with a way better loss 0.18. As well, visually features were a lot more separated when plotting them on a 3D graph, making it easier for the model to classify audio.

The next iteration experimented with input audio frequency. We chose to change the frequency down to 16000 Hz because most bird sounds have frequency ranges between 1000 Hz and 8000 Hz (All About Birds, 2009) and considering Nyquist-Shannon sampling theorem, 16000 Hz should be just enough to capture all relative data in the audio sample. This has not changed model's training performance; however, it has simplified feature extraction process making it faster and cheaper to compute features.

The one after iteration aimed to improve model's generalisation performance. The previously duplicated 3 minutes of Araripe Manakin audio samples were replaced by randomly augmented data. Once again, 50% of data from original Araripe Manakin class were randomly duplicated and then augmented with one of the following techniques randomly selected; change pitch by 1 step higher/lower, change speed by 0.2 faster/slower, inject static background noise with 0.001 noise factor or shift time forward/backward by a maximum of 0.2 seconds. For time shift and speed change, data is zero padded to ensure the length of the sample stays the same. This has increased the performance of the model drastically to 97.7% accuracy and 0.09 loss.

For the last lot of iterations, we were experimenting with changing hyperparameters to see what would increase model's performance. This included changing to 2D Convolutional Neural Network, applying various data augmentation techniques provided by Edge Impulse such as masking random blocks from the frequency axis, modifying number of epochs and changing learning rate. Only one hyperparameter was able to improve model's performance, which was low masking of random blocks from the time axis. Model's training performance has changed to a lower loss of 0.06 and the accuracy stayed the same.

The final version of model's properties can be seen on Figure 4.12 and model's neural network settings on Figure 4.13. Then the Figure 4.14 shows how model's MFCC features generated from training dataset are visually separated when plotted on a 3D graph. As well, Figure 4.15 shows model's confusion matrix that was done on validation set, with overall training performance of 97.7% accuracy and 0.06 loss. Lastly, Figure 4.16 shows the whole dataset for the final version of the model that can be previewed using audio sample palette on Figure 4.17.

Property	**Value**
Window Size	2000 ms
Window Increase	500 ms
Frequency	16000 Hz
Feature Extraction Block	MFCC
Learning Block	Classification Keras
Output Features	_noise, _unknown and araripe_manakin
Dataset Split (Training:Validation:Test)	60:20:20

Figure 4.12 Machine Learning Model's Properties Overview

Property	**Value**
Number of Training Cycles	100
Learning Rate	0.005
Further Data Augmentation Techniques	Mask time bands (low)
Neural Network Architecture	1D Convolutional

Figure 4.13 Machine Learning Model's Neural Network Settings

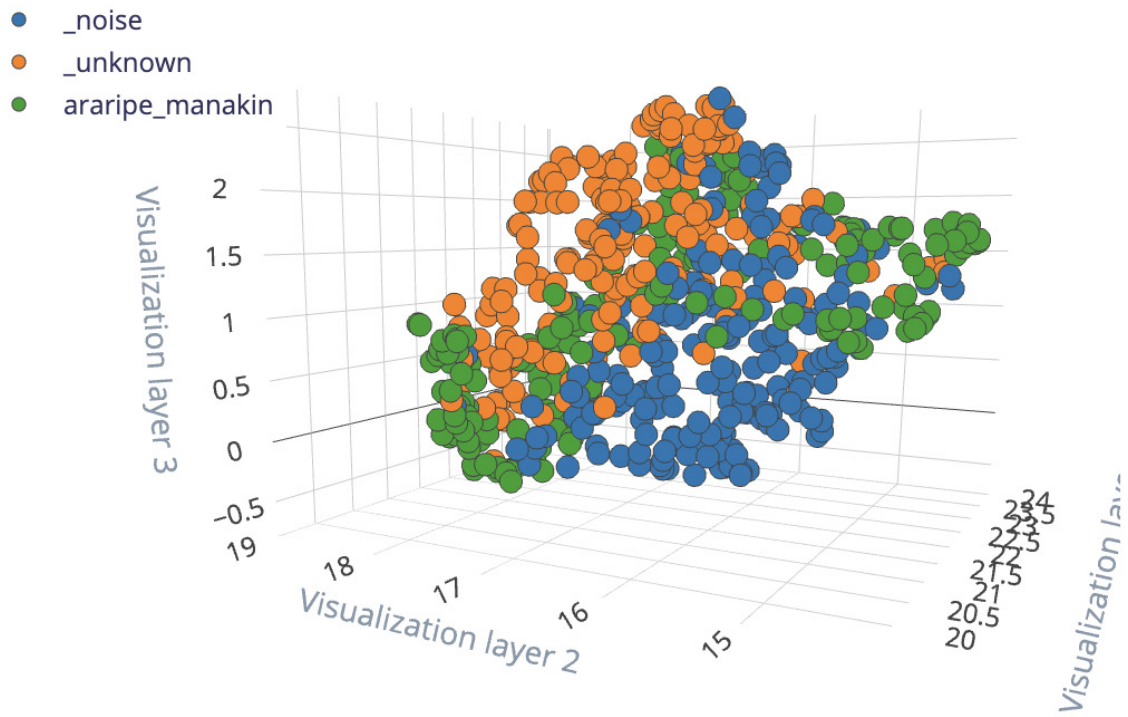


Figure 4.14 Machine Learning Model's MFCC features plotted on 3D Visual Graph

	_noise	_unknown	araripe_manakin
_noise	97.8%	2.2%	0%
_unknown	0%	97.4%	2.6%
araripe_manakin	0%	2.2%	97.8%
F1 Score	0.99	0.96	0.98

Figure 4.15 Machine Learning Model's Confusion Matrix (validation set)

araripe_manakin

- 6 minutes from Xeno-Canto
- 3 minutes of randomly duplicated data with data augmentation (either change to lower/higher pitch by 1 step, change to faster/slower speed by 0.2, inject static background noise with 0.001 noise factor or shift time forward/backward by max of 0.2 seconds)

_noise

- 1 minute manually collected data of random crackles/bangs
- 8 minutes from BBC Sound Effects (1 min of stream/waterfall, 2 min of forest, 1 min of wind, 1 min of village, 1 min of people talking, 2 minutes of silence and static noise)

_unknown

- 6 minutes of many random birds singing together
- 30 seconds of Boat-tailed Grackle
- 30 seconds of Mute Swan
- 30 seconds of Song Sparrow
- 30 seconds of Tricolored Heron
- 30 seconds of Common Blackbird
- 30 seconds of Ocellated Crake

Figure 4.16 Machine Learning Model's Dataset

Class	Sound	Sample Audio
araripe_manakin	Raw Araripe Manakin	
araripe_manakin	Augmented Araripe Manakin (speed changed)	
_noise	Random crackle/bang	
_noise	Stream/waterfall	
_noise	Forest	
_noise	Wind	
_noise	Village	
_noise	People Talking	
_noise	Silence and static noise	
_unknown	Many random birds singing together	
_unknown	Boat-tailed Grackle	
_unknown	Mute Swan	
_unknown	Song Sparrow	
_unknown	Tricolored Heron	
_unknown	Common Blackbird	
_unknown	Ocellated Crake	

Figure 4.17 Audio Sample Palette of Machine Learning Model's Dataset

4.4 Designing 3D Case Solutions

We used Fusion 360 (Autodesk, 2013) a 3D modelling platform to create all of the designs. Before creating case models, it was required to prototype all the physical hardware that will be going inside the case as CAD models. CAD models for Raspberry Pi and Arduino devices have been retrieved from GrabCAD. However, for the rest of the hardware we were not able to find both 3D CAD models and blueprints, having to produce physical hardware as CAD models. We used a ruler to measure the dimensions of hardware which were then used in CAD software to create 3D models. Afterwards, all models were assembled within Fusion 360 as they would be in real life and then both Raspberry Pi and Arduino case solutions were created around them (see Appendix B – 3D Case Solution for Raspberry Pi and Appendix C – 3D Case Solution for Arduino).

Throughout the whole case development, most of the good principles outlined by Hubs (Wall, 2021) were followed, these include; a minimum 2mm wall thickness to ensure the wall does not break due to being too thin, adding fillets to corners to help reduce stress at corners, having 0.5mm component clearance around all of the internal hardware to compensate for printer tolerances, uniform wall thickness as good design practice, 2mm port clearance to ensure cables fit through the opening and lastly subtracting 0.25mm from the screw hole diameter to allow the screw to make its own thread during installation.

5 Results and Evaluation

This section covers an evaluation of the systems created to achieve the outlined project's goals. Before carrying out any evaluation, a well-defined evaluation plan was created that included information about what will be captured and how the evaluation will be done.

5.1 Model Evaluation using metrics

In this section, the project's developed model is compared to the Microsoft's one using purely metrics. As both models have been trained with balanced datasets, the following metrics were captured to evaluate the models; accuracy, precision, recall and F1 score. To obtain metrics, both models were evaluated against test data which consisted of 20% of unseen data from the dataset. The results showed that the overall accuracy of the Microsoft's model rounded up to the nearest whole percentage is 96% compared to the final year project's model 97% being slightly ahead. This implies that both models are really good at identifying patterns in test data. Microsoft's model can only classify two classes being araripe_manakin and everything else, therefore the latter class will be compared to final year project's model's both _noise and _unknown classes when comparing confusion matrices of both models because both classes also capture all of the noises outside of the target bird.

As could be seen on Figure 5.1, the final year project's model's precision for target class is better by 0.05, meaning that it captures less false positives than the Microsoft's model, capturing more relevant Araripe Manakins than irrelevant sounds. In terms of non-target classifications, the precision is pretty much the same for both models being around 0.97, having the same number of false positives. For recall, Microsoft's model has considerably lower score for target class than the final year project's model, less by 0.07. This implies that Microsoft's model classifies more false negatives for the target class, capturing less of actual Araripe Manakins. For non-target classes, final year project captures all of the true positives from _noise class and only 0.93 for _unknown class, however roughly on average having the same 0.97 recall score as the Microsoft's model. Having combined both previous metrics using F1 Score, final year project's model outperforms the Microsoft's model on target class by 0.06, capturing the target class more accurately and precisely. As for the non-target class, on average final year project's model has slightly less score by 0.005 which is not significant, having about the same performance as the Microsoft's model.

In conclusion, on paper final year project' model has a better performance than the Microsoft's model in both capturing less false negatives and false positives for the target class. As for the identifying non-target sounds, both models perform nearly identical.

Model	_noise Precision	_unknown Precision	araripe_manakin Precision
Microsoft's Model	0.97	0.97	0.91
Final Year Project's Model	0.95	1	0.96

Figure 5.1 Precision Metric of Microsoft's and Final Year Project's Model

Model	_noise Recall	_unknown Recall	araripe_manakin Recall
Microsoft's Model	0.97	0.97	0.91
Final Year Project's Model	1	0.93	0.98

Figure 5.2 Recall Metric of Microsoft's and Final Year Project's Model

Model	_noise F1 Score	_unknown F1 Score	araripe_manakin F1 Score
Microsoft's Model	0.97	0.97	0.91
Final Year Project's Model	0.97	0.96	0.97

Figure 5.3 F1 Score Metric of Microsoft's and Final Year Project's Model

5.2 Model Evaluation in real world

In this section, both final year project's model and Microsoft's model are compared to each other after running them in real world, seeing how they perform under realistic circumstances. Both models were run on two different architectures Raspberry Pi and Arduino with the exception of Microsoft's Model running on Arduino due to the absence of support for Python's SVM classifier module in C++ which is used by the model. This was done to see if the performance changes depending on the hardware and the framework. Two different evaluations were carried out, which were done manually with mobile phone's speakers used as source of audio. These tests were subjected to various environments and conditions, simulating the real habitat.

5.2.1 False Negative Test

For this evaluation, models were tested to see how well they classify target class in different conditions. The test involved playing five different sounds of the target class for each condition variation and measuring how many times did the model classify the bird correctly, with confidence threshold of 0.8.

There were four different test categories for the controlled environment per architecture and model combination where tests were carried out in a silent room. These tests involved background noise being played using mobile phone device, to see how well the model can classify target sounds whilst having various noises playing at the same time in the background. For each test category, four tests with the same background noise playing at 70 dB volume were carried out, playing the target sound starting at 70 dB all the way down to 40 dB, both sounds being played at constant 1 metre distance away from the device running the model. The results showed that Microsoft's model could classify target sound 0.75 times on average for tests with people talking sound as the background noise. Comparing to a lot higher 3.25 times on average for final year project's model running on Raspberry Pi and 2 times on average when running on Arduino. For tests with water stream in the background, once again Microsoft's model's 0.75 average was a lot lower than final year project's model's 2.5 times on Raspberry Pi and 1.75 times on Arduino. For tests with wind whistling as background noise, both models performed really well at all ranges of target volume, Microsoft's model's 3.75 average compared to final year project's model's slightly higher 4.5 average on Raspberry Pi and slightly lower 3 average on Arduino. This happened most likely because of background sound being much quieter than previous ones, being able to hear

the target sound over it much easily. Lastly, tests with many random birds singing as the background noise, Microsoft's model did not perform that well with 2.3 times on average compared to final year project's model's 3.5 times on Raspberry Pi and 2.5 times on Arduino.

For the tests carried out in the wild environment which was done in one of the forests North of Cardiff, tests were only subjected to the natural habitat noise with no additional background noise added. Five different tests were carried out per platform and model, with target sound being played at consistent 50 dB volume and distance ranging from 1 to 20 metres. The results revealed that up to 15 metres, both models performed very well, final year project's model running on Raspberry Pi classifying every sound and when running on Arduino classifying 80% of sounds compared to Microsoft's model classifying 87% of sounds. Then between 15 and 20 metres, performance of final year project's model running on Raspberry Pi dropped to 50% and running on Arduino down to 20% compared to 40% for Microsoft's model.

In conclusion, final year project's model running on Raspberry Pi had the best performance out of all in both types of tests, considerably outperforming Microsoft's model and having an upper edge over the same model being ran on Arduino. Microsoft's model comparing to final year project's model running on Arduino, on average performed worse in controlled environment tests but performed better in the wild environment. The reason for final year project's model's performance on Arduino is due to the approach of the framework. The framework filters out true positives with the implemented moving average filter, which was supposed to help filter out false positives. On choosing which architecture should the final year project's model be run on, choosing Arduino when requiring to classify birds within short distance is fine as the difference between Raspberry Pi architecture is not that significant. However, if accuracy of identifying true positives is of high importance or if classifying birds on longer distances is required, then Raspberry Pi architecture should be chosen.

5.2.2 False Positive Test

For this evaluation, models were tested against all kinds of sounds not including the target class sounds, to see if the model classifies them incorrectly as target. The test played five different variations of the sound, measuring how many times did the model classify them incorrectly as target class with the same 0.8 confidence threshold. All of the sounds were evaluated under the same conditions of being in controlled environment with sound being played 1 metre away from the device and at the volume of 60 dB.

Two different categories of sounds were played during this evaluation. The first one includes sounds that can be found in the habitat but do not include sounds of birds, they are: people having conversations, environmental sounds (water stream, wind and forest) and random mammal animal sounds. The second category of sounds includes sounds of specific birds. The category has sounds of three randomly picked birds which are Ivory-billed Woodcreeper, Olivaceous Saltator and Prairie Warbler. The category also includes Helmeted Manakin which is a bird from the same family and another bird called Caroline Wren which has similar pitch and call to the targeted bird, to see how models perform on similarly sounding birds.

The results for this evaluation showed that both models performed very well on non-bird sounds from the former class, not classifying any as the target sound apart from the final year project's model running on Arduino classified random mammal animal sound once as the target. For the randomly selected birds, the final year project's model running on both architectures classified birds as target class 2.3 times on average compared to 1 for Microsoft's Model, more than twice less accurate at filtering out random birds. When it came down to birds with similar sounds, both models could not differentiate the bird from the same family due to having a nearly identical call to the target. For the bird with similar pitch and call, final year project's model was twice more accurate than Microsoft's model, identifying incorrectly twice on both platforms compared to four times for the latter model.

In conclusion, both models performed well on not identifying random sounds as the target bird. Microsoft's model performs best with random birds, whereas final year project's model performs best with similar sounding birds. The reason for Microsoft's model's performance most likely lies within the approach, being very good at identifying non-target sounds due to being far apart in the vector space of SVM, but not so good on similar sounding audios due to them being close to the target in space. For the final year project's model, it performs nearly identical on both Raspberry Pi and Arduino platforms, so neither of the frameworks affected model's performance on identifying non-target sounds incorrectly. This proves that the moving average filter technique used by edge framework for Arduino has not worked as intended, having the same performance on filtering out false positives as edge framework for Raspberry Pi that does not use such technique.

5.3 Architecture Evaluation

In this evaluation, the final year project's model was evaluated running on both Raspberry Pi and Arduino architectures, to see which architecture has better performance and efficiency. When measuring model's performance on the device, Edge Impulse's provided "On-device performance" estimates were used which were further confirmed by manually running the model on both architectures and taking an average over 10 classifications. For measuring energy and RAM usage, PowerTOP (Ven, 2010) diagnostic tool was used on Raspberry Pi. However, due to lack of Arduino libraries that measure power consumption, Arduino's official datasheet for Arduino Nano 33 BLE Sense (Arduino, 2021) was used to calculate estimates.

In terms of efficiency, as could be seen on Figure 5.4 Raspberry Pi has 50% less battery capacity and significantly more energy usage than the Arduino architecture. This makes the operating time of Raspberry Pi to only 3 hours compared to Arduino's nearly 10 days. This makes the Arduino architecture way more practical for long term deployments, being able to deploy the device for longer amount of time. Raspberry Pi in comparison will have a lot more downtimes due to the need for frequent recharging, during which missing potential classifications.

Platform	**Battery Capacity**	**Energy Usage**	**Operating Time**
Raspberry Pi	3000 mAh	967 mA	3 hours and 6 min
Arduino	4500 mAh	19 mA	236 hours and 50 min

Figure 5.4 Energy Metrics of Raspberry Pi and Arduino Architectures

As for the performance, model running on Arduino architecture in total takes 24 times longer classifying sounds than model running on Raspberry Pi, most of the time taken extracting features. However, Arduino's RAM usage is considerably lower being 26 KB compared to Raspberry Pi's 304 MB. This implies that Raspberry Pi has way better performance than Arduino architecture even through it uses more resources, which explains the bad energy efficiency. Raspberry Pi is substantially quicker at classifying than Arduino, allowing for results being received faster. As well, this means that Raspberry Pi architecture is more scalable compared to Arduino, allowing for more complex models to be deployed with greater amount of target classes.

Platform	**Processing time for feature extraction**	**Processing time for inference**	**Total processing time**	**RAM usage average**
Raspberry Pi	8 ms	1 ms	9 ms	304 MB
Arduino	206 ms	10 ms	216 ms	26 KB

Figure 5.5 Performance Metrics of Raspberry Pi and Arduino Architectures

In conclusion, each architecture has its own advantages and disadvantages that need to be considered when choosing which architecture to deploy. For long term deployments, Arduino architecture is the one to choose due to its long operating time. If performance is a concern, deploying a new version of the model with more target classes and higher complexity, then Raspberry Pi architecture should be chosen.

6 Future Work

Within this section, we talk about what can be done next that would improve the project and solve any of the previously outlined issues.

6.1 Exhaustive Evaluation of Model Performance

The evaluation of both models was not fully exhaustive especially the real-world tests and quite likely did not give a full picture of the models' performances. Therefore, it is recommended to perform more exhaustive tests of both models running on both architectures, this time including Microsoft's model running on Arduino by finding a way to import SVM library over to C++. The test should include more conditions and combinations of these conditions that would be more representative of the environment within the real habitat. As well, both models should be evaluated within the actual habitat where the target birds can be found, testing the models against real birds in real environments. This will solidify the findings by having a more accurate evaluation of models against real data, as this is what the model will be working against when it will be deployed in the future.

To take the evaluation further, tests can be automated through a list of exhaustive unit tests. A range of conditions can be collected as audio samples which can be then combined with samples of target birds, that would pass the model through different environments and see how it performs. This way, evaluation will be done a lot faster and will require less of manual work.

6.2 Arduino Library for LoRa Transceiver

As outlined previously by the implementation section, there were difficulties finding a library for Arduino Nano 33 BLE Sense that would be able to send radio messages using LoRa transceiver. Due to this difficulty, the project currently does not send Arduino edge device's results over to the gateway device. It is encouraged to write a specialised library from scratch or fork and adapt one of the libraries out there such as RadioHead (Crespo, 2018) for Arduino Nano 33 BLE Sense to enable usage of LoRa transceiver for sending classification results over to the gateway device. An alternative to this would be finding another Arduino device that is capable of running machine learning model and that already has support for using LoRa transceiver by one of the libraries online. However, this would then require rewriting edge framework for Arduino and modifying 3D case design, to support then new Arduino device.

6.3 Power Optimise Raspberry Pi Edge Architecture

The biggest issue with Raspberry Pi edge architecture currently is that it uses a lot of power to run the framework, having a very small operating time compared to Arduino architecture. Currently, the Raspberry Pi boots up into normal mode with GUI present, using that extra energy to load the graphics when it is not required to. This could be avoided by running the device in the headless mode, which would load only terminal and disable GUI access to the device.

Additionally, according to article on blues wireless (Lauer, 2021), power can be saved by disabling Wi-Fi and Bluetooth interfaces as they will not be used by the edge framework, saving up to 40 mA. Furthermore, all onboard LEDs can be disabled which will not be seen

by anyone after the deployment, that could save around 10 mA. As well, a few mA can be saved by underclocking Raspberry Pi's CPU. As with the current model, the device utilises only 19.6% of the CPU on average, so slowing down the CPU might not reduce model's performance on the device. However, this should be evaluated and tested before deploying the architecture.

Reviewing all of the running processes during bootup could also help with power consumption by finding irrelevant background processes that are not used by the framework during deployment. Another way to increase operating time on remote deployments is by introducing supplemental power. A small solar panel such as PiJuice Solar (Pi Supply, 2020) can be added on top of the Raspberry Pi as a HAT that would extend battery life of the edge architecture. However, this would then require revising the case for Raspberry Pi as it will need another waterproof opening for the solar panel.

6.4 Send Audio over LoRa

As was mentioned previously, it was not possible to send audio from edge framework over to the gateway using LoRa due to the maximum LoRa message size of 256 bytes. Measuring the size of the 2 second raw audio returned by the model, the average size of the audio is 110 kilobytes which is considerably larger than the LoRa's maximum message size. To send audio over LoRa, the audio can be split up into smaller segments and sent in multiple goes. This could work by having a queue on edge framework which would be filled with audio and results when there is a new classification. When there is something in the queue, a process could be spawned which would work parallelly to the inference and data collection process, that sends the classification results with the first LoRa message as done currently. Then for the audio, the process could start off with a simple handshake to ensure that the gateway device is turned on and is ready to listen. The handshake would work by having edge device calculate how big the audio is and how many parts the audio is split into, which would send that information to the gateway device, waiting for the confirmation response. After receiving confirmation from the gateway device, the edge device would sequentially send the audio in parts of 256 bytes, until the full audio is sent. The gateway framework would then re-assemble the audio and save locally. To combat packet loss, TCP's fast retransmit (Medhi & Ramasamy, 2018) technique could be used, which introduces acknowledge mechanism to the receiver (i.e. gateway device) used on each received packet. Each packet will be numbered and on receiving out of order packet, the receiver will send the acknowledgment for the last packet it has successfully received once again. The sender (i.e. edge device) would get a duplicate acknowledgement and would know to send the lost packets again. This feature could be implemented to both architectures, so there will be no need for retrieving edge devices to extract audio stored on them.

6.5 Integration with Helium Network

Currently, the project is limited by the LoRa's range, allowing for edge devices to be deployed only within 10 miles of the gateway device. This can be improved by integrating with Helium network (Helium, 2022) which provides secure, robust and affordable network powered by the Helium blockchain. Helium network is made up of devices called hotspots which can be purchased by anyone and installed anywhere. Hotspots relay data over long distances using the same LoRa protocol. Helium hotspot providers get rewarded by the users of the network in cryptocurrency called HNT and are constantly verified with Proof of

Coverage consensus (Helium, 2022) to ensure the validity of the nodes. The project can use Helium network to relay edge devices' LoRa messages over to the gateway device, allowing to be deployed over longer distances without previously mentioned restrictions. The integration will also require additional funding to pay for the network usage, paying for each relayed message. However, before integrating with the network, an analysis must be made to ensure that Helium network is physically present within the deployment location, otherwise the network will not be able to receive the messages.

7 Conclusions

The project's model overall has been very successful, being able to confidently classify target birds within real world environments. Model's performance on both architectures is as good or better than an already existent Microsoft's model which is currently used for bird conservations. On paper, final year project's model achieves the same accuracy as the Microsoft's model and has a better F1 score, outperforming the competition at both detecting less of irrelevant data and capturing more of the real data. In practice, it has been discovered that actually the final year project's model captures more false positives than the Microsoft's model. However, it is at the trade of having a lot less false negatives, being able to detect birds in noisy environments and differentiate similar sounding birds better than the Microsoft's model. Comparing final year project's model running on both architectures, Raspberry Pi proved to have a better performance over Arduino in all kinds of tests. Model running on Raspberry Pi was able to capture more of actual bird sounds than model running on Arduino device. As well, Raspberry Pi performed equal to Arduino in terms of filtering out false positives, which Arduino was supposed to have an upper edge on. Overall, the final year project's model is ready to be retrained for other types of birds if needed and then deployed to see how it performs against real data.

As for the frameworks, the project has not achieved all of the outlined aims. The edge framework for Arduino was not able to transmit results over the wireless network due to difficulties with the hardware, making it not ready for deployment. Although, the edge framework for Raspberry Pi was able to satisfy all requirements which are taking audio input, facilitating model to carry out inference and finally transmit results wirelessly over LoRa. Due to unforeseen issues with receiving LoRa signals, the project has been extended to implement another framework for the gateway device, which listens to the results and saves them locally.

In terms of the architectures, both Arduino and Raspberry Pi solutions have achieved all project's objectives. The design of the architectures has been successful, making it theoretically possible to transmit results wirelessly, take audio inputs, run machine learning model and be battery powered. On comparing both architectures, the Arduino architecture is a lot more power efficient, being able to be deployed for longer periods of time compared to Raspberry Pi. Nevertheless, Raspberry Pi framework proved to have a better performance, making it a lot more scalable for the future models.

For 3D case solutions, both designs have achieved all of the project's requirements of being weather resistant, provide openings for the microphone, support various installation methods and capture all of the hardware securely within itself.

Overall, the project was considered a success. All of the main aims of the project have been achieved to a sufficient degree, being able to be deployed in real habitats and used for conservation activities. A quick demo of the project can be found at Appendix D – Final Year Project Demo.

8 Reflection

The project has challenged me a lot, requiring to do a lot of research and learning a lot of concepts prior starting working on the project. Before the project, I only had general knowledge about machine learning. To start working on the model, I had to learn basic concepts of machine learning and understand machine learning's more in-depth mechanics relating to my problem which includes feature extraction techniques for bird audio, over sampling due to small dataset, audio data augmentation and more. I was also required to gain an understanding on bird vocalisation and certain vocal features that could differentiate birds from one another. As well, learning about possible issues with bird vocalisation that could impair the model's performance such as inter-species variance and bird calls compared to songs. Learning about how sounds work was required too, making sure that all of the bird's vocal details are captured during audio collection and feature extraction. Before starting to work on 3D case solutions, I had to understand general design principles for 3D printing to ensure that the case after being printed would be structurally stable. I also had to research common engineering designs for creating a snap fit joint and making casing waterproof, from which I unexpectedly learnt a lot about 3D modelling.

As for the skills that I have developed throughout the project development, I was able to learn new to me programming language C++. Programming at lower level was unusual to me, from which I was able to gain knowledge on how to do memory management by implementing buffers. As well, I have developed knowledge in source control, database, data structures and algorithms, radio networking, testing and architecture design by implementing the frameworks for IoT architectures. I had little experience in 3D modelling prior to the project. During the project, I was able to learn how to read schematics, how and what tools to use in CAD software, how to take proper references from real life objects to remodel them virtually and understanding geometry concepts such as curvature and projection by creating 3D models of hardware and cases within CAD software.

If I was to redo this project from scratch, I would spend a lot more time researching about the technologies that will be used in the project. As mentioned previously, the project had to be extended to support LoRa signals due to my incorrect assumption that regular personal devices can read radio signals. If I had spent more time understanding LoRa technology, I would have been able to take actions early, whether to continue with this technology or not. However, due to being late with the realisation of the issue, I was forced to extend my project's scope to suit this technology. Also, choosing LoRa protocol was another mistake I would like to avoid next time when working with long range wireless transmissions. Due to limited size of LoRa's messages, I was unable to send full results over to the gateway device, being forced to work around by saving audio locally and retrieving it physically. This problem occurred once again due to not fully understanding the LoRa technology at the time of designing the architecture, knowing LoRa's maximum packet size I would have redesigned the architecture or accounted for during early stages of the project. Another issue that I had with the project was having trouble with finding a library for LoRa transceiver that would support the chosen Arduino microcontroller. Next time, I would carry out more thorough research and ensure that all of the components including libraries for them are compatible. At the very least, I would include developing a specialised library within my initial plan, having some arranged time to make the hardware compatible which I did not have previously.

During the project, there were many things that went well which I would like to reuse when working on something similar in the future. Firstly, the creation of machine learning model was very successful as proved by the evaluation. Data augmentation techniques have played a big part at increasing model's accuracy, helping to generate more data which did not overfit the model and instead brought more variation to the class. In the future, data augmentation will definitely be of use when working with small datasets. As well, creating two extra classes for the model, one being generic environment noises and another for non-target birds have really helped the model to differentiate target birds from all other sounds. Giving context to the model of the outside world has played a tremendous part in making the model competitive against other models, which I will consider when working on machine learning models in the future. Another task that went well was designing 3D case solutions for the architectures. All of the applied design principles for 3D printing have successfully helped these cases become sturdy, which I will reuse next time I will be doing 3D modelling. Also, learning how to create cantilever snap fit joint allowed to enclose casing without requiring extra tools, accelerating the prototyping and development processes. Cantilever snap fit joint proved to be an easy and rapid way of assembling and disassembling cases, which I will also reuse in the future.

9 References

All About Birds, 2009. *Do Bird Songs Have Frequencies Higher Than Humans Can Hear?.* [Online]
Available at: https://www.allaboutbirds.org/news/do-bird-songs-have-frequencies-higher-than-humans-can-hear
[Accessed 02 04 2022].

Analog, 2022. *Moving Average Filters.* [Online]
Available at: https://www.analog.com/media/en/technical-documentation/dsp-book/dsp_book_ch15.pdf
[Accessed 02 04 2022].

Arduino, 2021. *Arduino Nano 33 BLE Sense datasheet.* [Online]
Available at:
https://docs.arduino.cc/static/702c3afe4d443ad7fc171d434ba0ee4a/ABX00031-datasheet.pdf
[Accessed 13 03 2022].

Arduino, 2022. *About Nano boards with disabled 5 V pins.* [Online]
Available at: https://support.arduino.cc/hc/en-us/articles/360014779679-About-Nano-boards-with-disabled-5-V-pins
[Accessed 12 04 2022].

Autodesk, 2013. *Fusion 360.* [Online]
Available at: https://www.autodesk.com/products/fusion-360/overview
[Accessed 20 03 2022].

Bayer, 2020. *Snap-Fit Joints for Plastics.* [Online]
Available at:
http://fab.cba.mit.edu/classes/S62.12/people/vernelle.noel/Plastic_Snap_fit_design.pdf
[Accessed 02 03 2022].

Brownlee, J., 2019. *A Gentle Introduction to Imbalanced Classification.* [Online]
Available at: https://machinelearningmastery.com/what-is-imbalanced-classification/
[Accessed 25 02 2022].

Colarusso, P., Kidder, L., Levin, I. & Lewis, N., 1999. *Raman and Infrared Microspectroscopy.* [Online]
Available at: https://www.sciencedirect.com/science/article/pii/B0122266803004026
[Accessed 05 03 2022].

Crespo, E., 2018. *RadioHead.* [Online]
Available at: https://github.com/jecrespo/RadioHead
[Accessed 06 04 2022].

DCASE, 2021. *Few-shot Bioacoustic Event Detection Challenge Results.* [Online]
Available at: https://dcase.community/challenge2021/task-few-shot-bioacoustic-event-detection-results
[Accessed 01 04 2022].

DSM&T, 2015. *IP Rating Chart.* [Online]
Available at: https://www.dsmt.com/resources/ip-rating-chart/
[Accessed 20 02 2022].

Edge Impulse, 2021. *Documentation.* [Online]
Available at: https://docs.edgeimpulse.com/docs
[Accessed 02 04 2022].

Edge Impulse, 2021. *Edge Impulse DSP and Inferencing SDK.* [Online]
Available at: https://github.com/edgeimpulse/inferencing-sdk-cpp
[Accessed 04 04 2022].

Edge Impulse, 2021. *Edge Impulse Linux SDK for Python.* [Online]
Available at: https://github.com/edgeimpulse/linux-sdk-python
[Accessed 20 03 2022].

Edge Impulse, 2022. *Audio MFCC.* [Online]
Available at: https://docs.edgeimpulse.com/docs/tutorials/processing-blocks/audio-mfcc
[Accessed 21 04 2022].

Edge Impulse, 2022. *Audio MFE.* [Online]
Available at: https://docs.edgeimpulse.com/docs/tutorials/processing-blocks/audio-mfe
[Accessed 21 04 2022].

Edge Impulse, 2022. *Spectrogram.* [Online]
Available at: https://docs.edgeimpulse.com/docs/tutorials/processing-blocks/spectrogram
[Accessed 21 04 2022].

Helium, 2022. *Proof of Coverage.* [Online]
Available at: https://docs.helium.com/blockchain/proof-of-coverage/
[Accessed 04 30 2022].

Helium, 2022. *Use the Network. The People's Network delivers secure, ubiquitos, and affordable wireless connectivity..* [Online]
Available at: https://www.helium.com/enterprise
[Accessed 04 30 2022].

Hont, 2022. *Complete Plastic Cable Ties Sizes (in MM) Summary Chart.* [Online]
Available at: https://hont-electric.com/what-are-the-plastic-cable-ties-sizes-in-mm/
[Accessed 13 03 2022].

IBM Cloud Education, 2020. *Convolutional Neural Networks.* [Online]
Available at: https://www.ibm.com/cloud/learn/convolutional-neural-networks
[Accessed 10 04 2022].

ISO, 2022. *ISO 8601 Date and Time Format.* [Online]
Available at: https://www.iso.org/iso-8601-date-and-time-format.html
[Accessed 25 03 2022].

Jolliffe, I. & Cadima, J., 2016. *Principal component analysis: a review and recent developments.* [Online]
Available at: https://royalsocietypublishing.org/doi/10.1098/rsta.2015.0202
[Accessed 18 03 2022].

Jongboom, J., 2020. *Introducing EON: Neural Networks in Up to 55% Less RAM and 35% Less ROM.* [Online]
Available at: https://www.edgeimpulse.com/blog/introducing-eon
[Accessed 04 04 2022].

Kahl, S. et al., 2021. *Overview of BirdCLEF 2021: Bird call identification in soundscape recordings.* [Online]
Available at: http://ceur-ws.org/Vol-2936/paper-123.pdf
[Accessed 20 04 2022].

Kaparykha, S., 2021. *Raspberry Pi 4 Model B.* [Online]
Available at: https://grabcad.com/library/raspberry-pi-4-model-b-03-1
[Accessed 10 04 2022].

Lauer, R., 2021. *Optimizing Raspberry Pi Power Consumption.* [Online]
Available at: https://blues.io/blog/tips-tricks-optimizing-raspberry-pi-power/
[Accessed 22 03 2022].

LoRa, 2019. *What are LoRa and LoRaWAN?.* [Online]
Available at: https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan/
[Accessed 07 04 2022].

Medhi, D. & Ramasamy, K., 2018. *Packet Queueing and Scheduling.* [Online]
Available at: https://www.sciencedirect.com/science/article/pii/B978012800737200020X
[Accessed 21 04 2022].

Microsoft AI for Earth, 2020. *Tutorial: Accurate Bioacoustic Species Detection from Small Numbers of Training Clips Using the Biophony Model.* [Online]
Available at: https://github.com/microsoft/acoustic-bird-detection
[Accessed 24 03 2022].

Moreno-Barea, F., Jerez, J. & Franco, L., 2020. *Improving classification accuracy using data augmentation on small data sets.* [Online]
Available at: https://www.sciencedirect.com/science/article/abs/pii/S0957417420305200
[Accessed 20 02 2022].

Peterson, A., 2021. *Arduino Nano 33 BLE.* [Online]
Available at: https://grabcad.com/library/arduino-nano-33-ble-1
[Accessed 10 04 2022].

Pham, C., Bounceur, A., Clavier, L. & Noreen, U., 2020. *Radio channel access challenges in LoRa low-power wide-area networks.* [Online]
Available at:
https://www.sciencedirect.com/science/article/pii/B9780128188804000041#f0025
[Accessed 02 04 2022].

Pi Supply, 2020. *PiJuice Solar.* [Online]
Available at: https://uk.pi-supply.com/products/pijuice-solar
[Accessed 28 03 2022].

Rahman, S., 2021. *Undersampling and oversampling: An old and a new approach.* [Online]
Available at: https://medium.com/analytics-vidhya/undersampling-and-oversampling-an-old-and-a-new-approach-4f984a0e8392
[Accessed 05 04 2022].

Seeed Studio, 2021. *Grove System.* [Online]
Available at: https://wiki.seeedstudio.com/Grove_System/
[Accessed 22 03 2022].

Semtech, 2022. *What Is LoRa?.* [Online]
Available at: https://www.semtech.com/lora/what-is-lora
[Accessed 04 03 2022].

Sonix, 2021. *A short history of speech recognition.* [Online]
Available at: https://sonix.ai/history-of-speech-recognition
[Accessed 04 04 2022].

Sprengel, E., Jaggi, M., Kilcher, Y. & Hofmann, T., 2016. *Audio Based Bird Species Identification using Deep Learning Techniques.* [Online]
Available at: http://ceur-ws.org/Vol-1609/16090547.pdf
[Accessed 03 03 2022].

Ven, A. v. d., 2010. *PowerTOP.* [Online]
Available at: https://github.com/fenrus75/powertop
[Accessed 05 04 2022].

Wall, J., 2021. *Enclosure design for 3D printing.* [Online]
Available at: https://www.hubs.com/knowledge-base/enclosure-design-3d-printing-step-step-guide/
[Accessed 03 03 2022].

Waveshare, 2020. *SX1268 433M LoRa HAT.* [Online]
Available at: https://www.waveshare.com/wiki/SX1268_433M_LoRa_HAT
[Accessed 06 04 2022].

Wildlife Acoustics, 2020. *Kaleidoscope Pro Pro Analysis Software.* [Online]
Available at: https://www.wildlifeacoustics.com/products/kaleidoscope-pro
[Accessed 24 4 2022].

Wildlife Acoustics, 2022. *Song Meter Comparison.* [Online]
Available at: https://www.wildlifeacoustics.com/products/song-meter-sm4-vs-mini-vs-micro
[Accessed 05 03 2022].

WWF, 2020. *68% Average Decline in Species Population Sizes Since 1970, Says New WWF Report.* [Online]
Available at: https://www.worldwildlife.org/press-releases/68-average-decline-in-species-population-sizes-since-1970-says-new-wwf-report
[Accessed 05 04 2022].

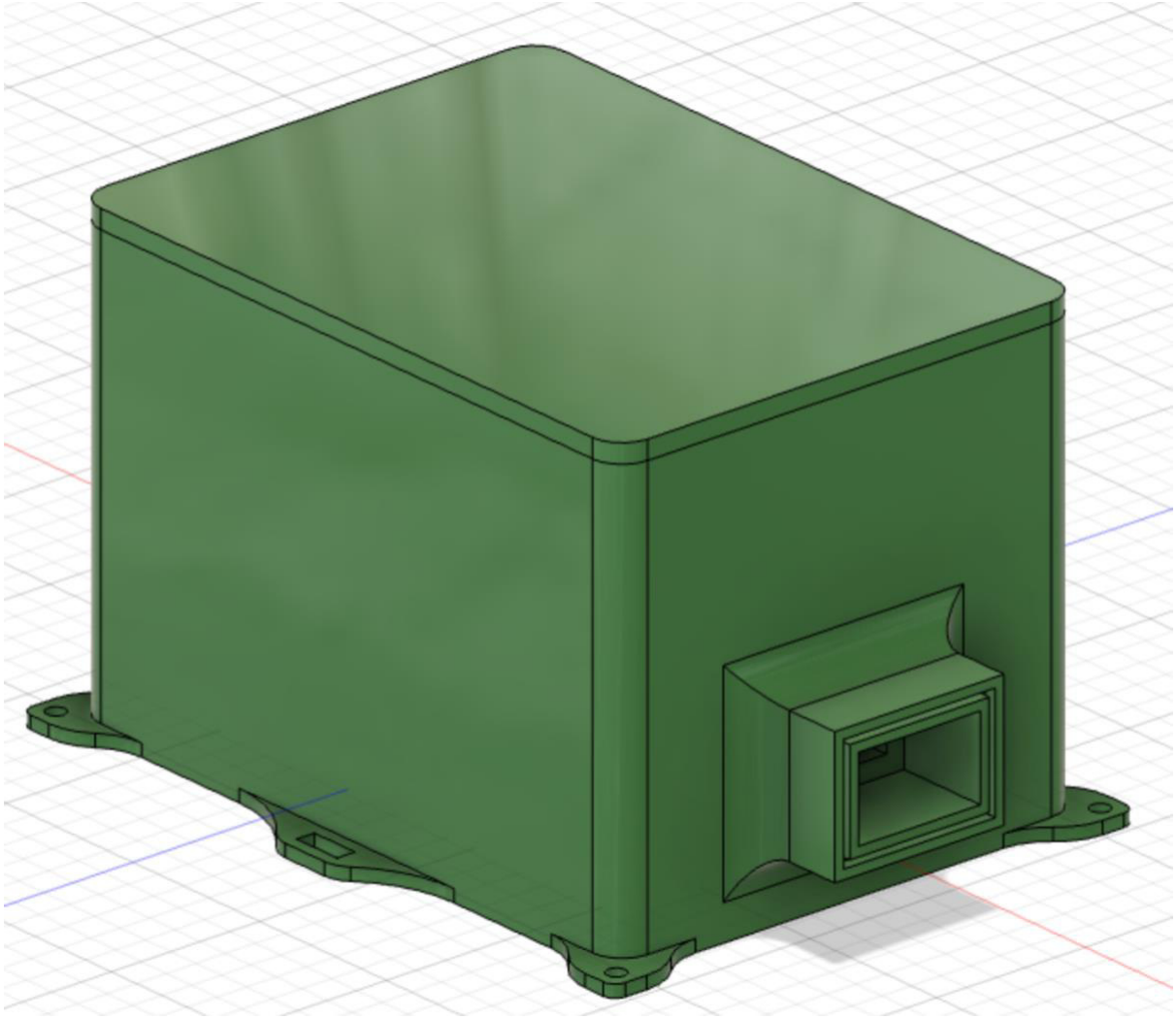
YR Architecture Design, 2020. *Design Techniques to Control Water Movement.* [Online]
Available at: https://yr-architecture.com/keeping-water-out-7-design-techniques-to-control-water-movement/
[Accessed 22 03 2022].

10 Appendix

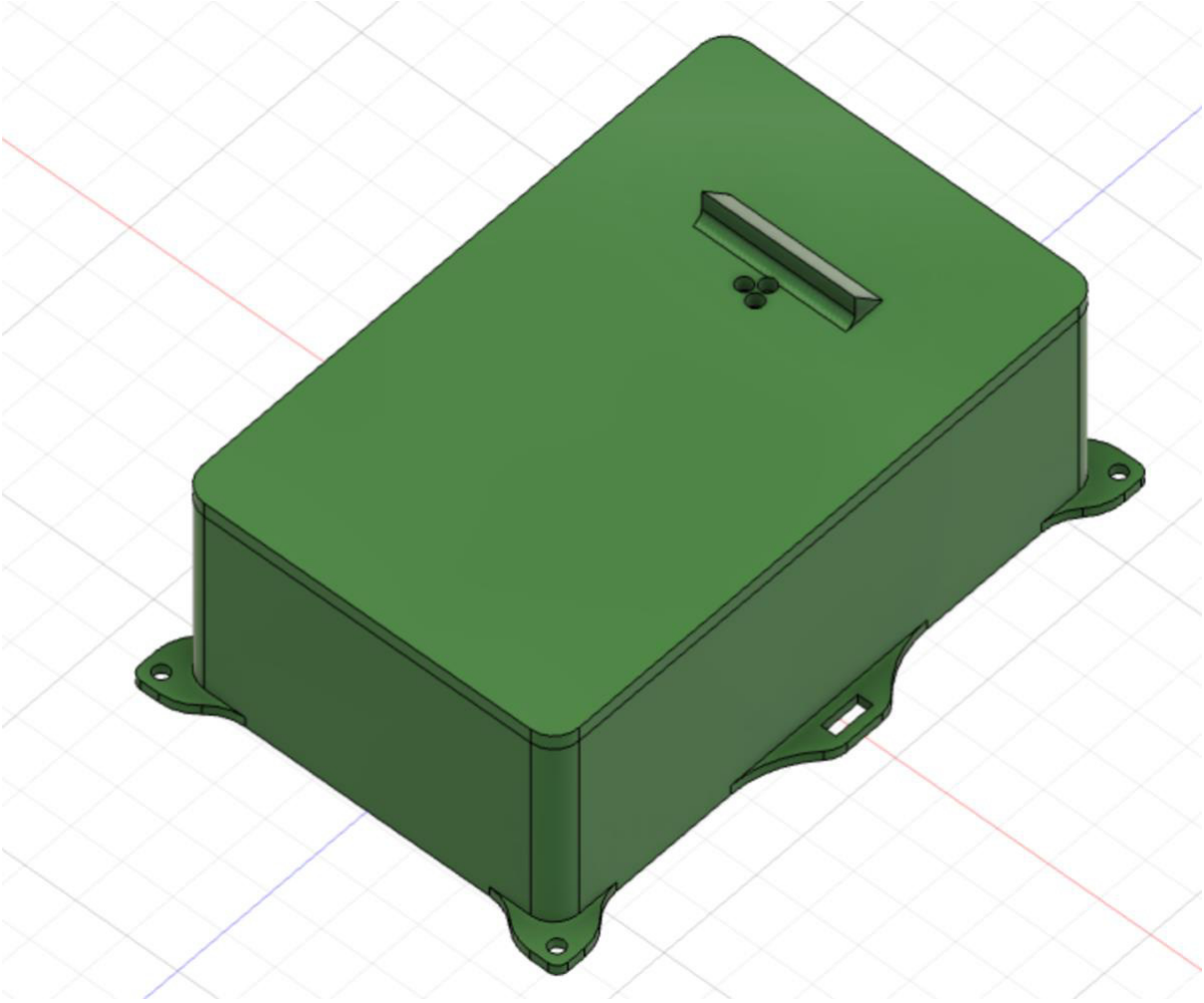
10.1 Appendix A – GitHub Repository for Final Year Project

- https://github.com/RuslanLevond/final-year-project

10.2 Appendix B – 3D Case Solution for Raspberry Pi



10.3 Appendix C – 3D Case Solution for Arduino



10.4 Appendix D – Final Year Project Demo

- https://www.youtube.com/watch?v=M5sv8-thzCE