Detecting Bats on the Edge – Machine Learning and Citizen Scientists

Daniel Rennie

This dissertation is submitted in partial fulfilment of the requirements for the degree of MSc in the Centre for Advanced Spatial Analysis, Bartlett Faculty of the Built Environment, UCL.

CASA0012
11,658 words
23 August 2021
Supervised by Dr Duncan Wilson

Abstract

Monitoring bats through surveys and passive acoustic microphones is vital in understanding the health of an environment and its biodiversity. As key biodiversity indicators, bats can provide critical insights into the health of an environment. However, due to their small size and nocturnal habits, tracking bats has always been labour intensive and only possible at a broad scale due to a combination of ecologists and a significant population of volunteer citizen scientists.

Advanced machine learning techniques have made monitoring and recognising bat calls less laborious, but these tools can be too expensive or too complicated for many ecologists and citizen scientists to use. This paper aims to create an edge-based bat detection device that, hopefully, non-computer scientists can replicate as it uses relatively inexpensive hardware combined with open-source software and a simple machine learning application. In addition, the paper also hopes to understand the constraints and challenges that using these tools entails.

A Raspberry Pi and Dodotronic Ultramic 192k microphone were used as the hardware, and a combination of Python, sound eXchange and Edge Impulse was used as the software to create the bat detection device. Edge Impulse is a web-based machine learning application that attempts to simplify the machine learning process and allows non-computer scientists to create machine learning-based devices and sensors.

Two bat detection prototypes were made using the tools listed above that approached and even surpassed the accuracy of 95% that the previous more sophisticated models achieved. However, as with any prototype, constraints and weaknesses needed to be worked through, understood, and discussed. Nevertheless, in the end, the prototypes do prove that creating lower-cost and more straightforward bat detection devices for non-computer scientists is possible and that machine learning is not as daunting as it may first appear for those new to it.

Declaration

I hereby declare that this dissertation is all my own original work and that all sources have been acknowledged. It is 11,658 words in length, as computed by Microsoft Word.

Table of Contents

List of tables

List of figures

List of acronyms and abbreviations

2D	Two Dimensional
BLERP	Bandwidth, Latency, Economics, Reliability, Privacy
CASA	Centre for Advanced Spatial Analysis
CNN	Convolutional Neural Network
dB	Decibel
DSP	Digital Signal Processor
GB	Gigabyte
GHz	Gigahertz
HDMI	High-Definition Multimedia Interface
Hz	Hertz
kHz	Kilohertz
MB	Megabyte
MFCC	Mel Frequency Cepstral Coefficients
MicroSD	Micro Secure Digital
NLP	Natural Language Programming
NN	Neural Network
OS	Operating System
RAM	Random-access memory
RGB	Red Green Blue
ROM	Read-only memory
RPi	Raspberry Pi
SDK	Software Development Kit
SoX	Sound eXchange
TinyML	Tiny Machine Learning
VPN	Virtual Private Network
UCL	University College London
UK	United Kingdom
USD	United States Dollar
Wi-Fi	Wireless Fidelity
USB	Universal Serial Bus

Acknowledgements

I would like to thank my supervisor, Dr Duncan Wilson, for all his help and guidance through this project, especially as some of the more technical elements were quite new to me. I would also like to thank my friend Josh Millar for some help in tidying up some ugly Python code to make things run a bit more smoothly. Lastly, I would like to thank my parents and girlfriend for their support throughout this paper and the year. I couldn't have done it without them.

1. Introduction

Technology is everchanging and ever-evolving in every field of research. From physics to medicine, advancements in technology have allowed researchers to progress in their fields and make discoveries. Ecology is no different; this is especially evidenced by the innovations to the technology used to track and measure bat populations.

Tracking bat populations is essential because bats are critical indicators of biodiversity across many different environments, including urban spaces, wetlands, and agricultural land (Dixon, 2012; Park, 2015; Russo and Ancillotto, 2015; Parker *et al.*, 2019). Historically bats have been tracked in the United Kingdom (UK) through various in-person surveys, including hibernation, woodland, sunset, waterway and field surveys alongside other methods like roost counts, physical bat tracking rings and passive acoustic surveys listening for bat call activity (Trust, no date b).

Much of the research done to study bats in the UK has only been achievable due to the vast contributions of citizen scientist volunteers. Over eighty bat groups across the UK work to research and conserve bats and their habitats (Trust, no date a). However, much of the leg work for the surveys, counts and listening campaigns mentioned above can only be achieved through these citizen scientists' enormous time and energy commitments (Barlow *et al.*, 2015).

This commitment is no more obvious than during the analysis of the passive acoustic surveys. Hours of recordings must be reviewed visually, through a spectrogram, or audio slowed down to a rate that humans could hear to inspect for the presence of bat calls. However, this pressure on citizen scientists and ecologists has been alleviated in recent years. Acoustic tracking has been advanced by machine learning techniques seen in commercial products like SonoBat, academic research like the Bat Detective project and by computer science and technology enthusiasts on websites like Instructables (Aodha *et al.*, 2018; Twmffat, no date; *SonoBat – Software for Bat Call Analysis*, no date). These devices analyse and predict the presence of bats in the recordings based on deep learning models trained by teams of computer scientists, ecologists, and citizen scientists. These predictions can then be verified without listening to entire recordings, significantly reducing their time commitments.

Using these machine learning techniques has always required a deep understanding of computer science that the average bat citizen scientist likely would not have on their own. Unfortunately, for most people, this meant that these advanced techniques were either wholly inaccessible or came at a steep cost like the SonoBat, which can cost up to USD $1,536.00 simply for the software to analyse the audio (‘Purchase – SonoBat’, no date). This inaccessibility is the issue that this paper attempts to solve. The research aims to create a bat detection prototype using three simple inputs:

1. As inexpensive hardware as possible
2. Free or open-source software
3. Approachable machine learning tools aimed at easing accessibility to non-computer science people

Using these three inputs as a base, the paper will explore the following research question:

What are the constraints of a low cost, easily updated bat identifier for citizen scientists versus previous, more sophisticated machine learning examples and can the accuracy of these more sophisticated models be matched?

The following sections of the paper, beginning with the literature review, explore the journey of working through this question and creating a working prototype.

2. Literature Review

The literature review will cover three broad topics:

1. **Why bats?** – This section will cover the importance of bats as an indicator of biodiversity and overall environmental health and how bats have been studied and tracked in the past in the United Kingdom.
2. **Technology review** – This section will cover computer science led deep learning techniques and how they are being utilised currently, it will investigate edge artificial intelligence and how it differs from the analysis that is done in the cloud, and it will close by looking at previous work done by machine learning and bat detection and tracking.
3. **Citizen scientists and non-computer scientists** – This section will cover the importance of citizen scientists in research today and how the overall democratisation of research is beneficial to the overall quality. It will also cover how modern computer science techniques are being opened to non-computer scientists and what this could mean for research moving forward.

2.1. Why bats?

2.1.1. Bats and their importance

Bats are one of the key indicators used to understand the health of the environment and biodiversity in the United Kingdom (Department for Environment, Food and Rural Affairs, UK, 2020). According to Jones et al., bats make strong biodiversity indicators because their populations are sensitive to many macro-environmental changes ranging from climate change to loss of habitat to the introduction of new diseases, among others (Jones *et al.*, 2009). However, bats also play many other functions in their environment beyond their role as a biodiversity indicator, including pollinators, pest controllers, seed dispersers, and reforesters (Bat Conservation Trust, no date b).

Since 1991, the Bat Conservation Trust has been supporting bat conservation across the United Kingdom, and since 1997, it has been running the National Bat Monitoring Programme to help understand bat numbers as they fluctuate through time and the impact of these fluctuations (Barlow *et al.*, 2015; Bat Conservation Trust, no date a). However, gathering information on bats for the programme and broader use has been and continues to

be a highly manual process. Trained volunteers have to go and physically observe and count bat roosts or hibernation locations or conduct field and waterway surveys listening for bats unique echolocation patterns using specialised bat detectors and counting the numbers and species heard (Barlow *et al.*, 2015). Passive surveys are also completed where recording devices are left in known or suspected bat habitats. The recorded audio, often hours or days long, is then analysed by ecologists either through software, either commercial or public, or through time-intensive manual verification to understand the bat presence in the study area (Gallacher *et al.*, no date).

Studying and understanding bat populations is incredibly challenging because of their small size and nocturnal nature. However, this challenge directly led to the proliferation of understanding bats through the acoustic surveys mentioned above, as this means the bats do not have to be now seen or captured to facilitate their study (Walters *et al.*, 2012). The following subsection discusses how technology, like machine learning, has greatly assisted ecologists and the broader population in studying bats (Gallacher *et al.*, no date).

2.2. Technology review

As acoustic monitoring emerged as one of the predominant methods used to track bat populations, the need for more straightforward, more efficient, and technologically savvy techniques arose. Logically, computer scientists and ecologists ended up collaborating on using machine learning techniques to help identify and classify bat calls (Aodha *et al.*, 2018).

The following subsections will discuss these machine learning techniques, how they are used in other scenarios and how these techniques have been brought back to helping ecologists track and monitor bats in their natural environments.

2.2.1. Machine learning and edge devices

In their book TinyML, Pete Warden and Daniel Situnayake say that machine learning, and subsequently deep learning, ‘is a technique for using computers to predict things based on past observations’ (Warden and Situnayake, 2019). They further break down the typical machine learning application into the following steps:

1. Obtain an input

2. Pre-process an input to extract features suitable to feed into a machine learning model
3. Run inference on the processed input
4. Post-process the model's output to make sense of it
5. Use the resulting information to make things happen (Warden and Situnayake, 2019)

In practical terms, that means that a model must be created and trained based on known inputs and their subsequent values or outcomes so that when those types of features are seen again by the machine learning application, it can then make an accurate prediction to what that is or what will subsequently occur.

There is no more obvious application of machine learning devices like Amazon's Alexa or Apple's Siri. These devices transform audio stimuli, human speech, into something recognised by a computer, process that input, and then hopefully give their users a helpful answer. By breaking down how Alexa or Siri work using Warden and Situnayake's steps, it becomes clear what occurs when it comes to the machine learning application (Warden and Situnayake, 2019):

1. **Obtain an input**
   This is the human speech as an audio stimulus.
2. **Pre-process an input to extract features suitable to feed into a machine learning model**
   A pre-processing technique for audio inputs may be a Mel Frequency Cepstral Coefficients (MFCC) or Fourier transformation (Muda, Begam and Elamvazuthi, 2010). Both changes transform audio into spectrograms generating features that the machine learning model can then analyse and break down if needed.
3. **Run inference on the processed input**
   This is where the device then determines what has been said using the model that it has been trained on.
4. **Post-process the model's output to make sense of it**
   The model then determines what was most likely said by the user, and the model decides based on its training as to what to do next.
5. **Use the resulting information to make things happen**
   In the example of Alexa or Siri, it will ideally give back the desired data or ask the user for clarification or further information if needed.

While this simplifies the entire process, it does indicate the machinations behind a machine learning application. However, for this paper, a deeper understanding of processing techniques like MFCC or Fourier transformations is not required. Instead, it simply needs to be understood that they are part of the machine learning application that allows inputs to be processed and the desired outcome or prediction to be obtained.

Another key feature of both Alexa and Siri is that they use both edge and cloud machine learning to allow the best user experience. In more traditional cloud-based computing, all inputs coming into a computer system are pushed up into a centrally based cloud. Then, they are analysed, processed, and pushed back out onto the device requiring information (Miller, 2018). The main advantage of cloud-based computing is that it allows for vast computer processing powers to be utilised at the servers. For example, cloud-based computing is the only reason small devices like mobile phones and Alexa speakers can process natural language programming (NLP) on such small devices (Gonfalonieri, 2018). While at the same time that this is an advantage, it can also be a massive disadvantage for devices like Alexa or Siri because it would require data to be streamed to these cloud servers 24/7, which would present both immense privacy and bandwidth problems for all users (Warden and Situnayake, 2019). These problems are where edge computing comes into play.

Edge computing is defined as when computing and storage are done at the internet's edge near the mobile devices and sensors (Satyanarayanan, 2017). In layman's terms, this refers to all computing and analysis done directly on tiny devices as they pick up data; in most examples, these types of devices may not even require an active internet connection to operate (Warden and Situnayake, 2019). However, when applied in a real scenario, this works for Alexa and Siri because it allows them to listen and sample audio continuously on small chips on the device itself and then only engage the higher levels of cloud computing power after it hears a wake word like Alexa or "Hey Siri." The combination of edge and cloud-based computing allows these applications to be run on much smaller, less powerful computing devices than would typically be possible if they were required to be constantly sending all data to the cloud all analysis or storing and processing all data on the device itself (Warden and Situnayake, 2019; *Continuous audio sampling*, no date).

The main advantages of edge computing are summarised best by the acronym BLERP, which Jeff Bier created in the EE times (Bier, 2020). BLERP is described below:

- **B**andwidth – As discussed earlier, edge devices significantly decrease the levels of bandwidth as the analysis is done on the device locally and not in the cloud. This means that less data must be transmitted between the device and the cloud, giving Internet bandwidth back to other tasks.
- **L**atency – Latency describes the time it takes for a system to take an input and respond to it. While sensor inputs can be sent to the cloud very quickly, there is still a delay when this happens. If the data can be processed on the device, it decreases the time required to input, making the desired outcome quicker.
- **E**conomics – Cloud computing costs money, and if the amount of computing that must happen in the cloud can be decreased, that will inevitably save money.
- **R**eliability – If a bat detection model is run on an edge device, it will detect bats if the internet goes down. This means that the device itself will be more reliable and will be able to run in more situations.
- **P**rivacy – The less data moves between the cloud, and the device, the less likely a privacy breach will occur. Data and information are much safer when it is processed and consumed locally (Bier, 2020).

2.2.2. Shazam for bats

Previous work done by Gallacher et al. titled Shazam for bats utilised machine learning to capture, record and identify bat calls (Gallacher et al., no date). Bat calls are distinctive as they only occur above the limits of human hearing, with the calls beginning at 20 kHz and ranging to 110 kHz for British species of bats (*Frequencies | Staffordshire Bat Group*, no date). The research captured bat calls using speciality microphones that could detect high-frequency bat calls. The calls were subsequently transformed using the Discrete Fast Fourier Transform method to create a spectrogram to create a visualisation. An example spectrogram can be seen below in Figure 2.1. At the same time, while different bat species' calls are unique in their way, when they appear as spectrograms, most occur in a shape like a hockey stick.

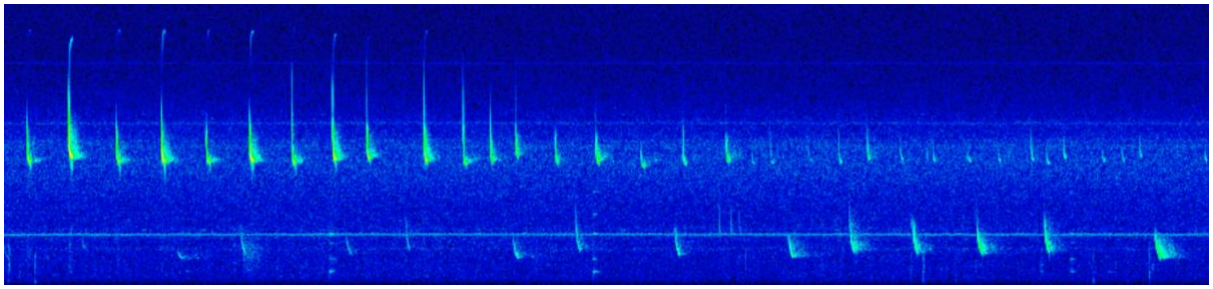


Figure 2.1: Spectrogram showing bat calls created using the Discrete Fast Fourier Transform method, provided by Gallacher et al., no date.

The Shazam for the bats' project is the main inspiration for this paper. For their research, the team created smart bat sensors called Echo Boxes. The Echo Boxes were made of a waterproof enclosure that held a Dodotronic Ultramic 192k microphone, an Intel Edison with Arduino breakout board, and the power supply components allowing the device to run off the mains power source (Gallacher *et al.*, no date).

The boxes constantly listened for bat calls and converted the continuous stream into three-second chunks, subsequently converted into spectrograms. Once converted, a computer vision deep learning model called Convolutional Neural Networks (CNNs) processed and classified the images looking for bat calls. The Echo Box then attempted to classify which bat species made the call when a bat call is detected. Following classification, the results are pushed into the cloud for storage, and the raw recording .wav file was deleted (Gallacher et al., no date).

The Shazam for bats machine learning model was trained on a database of over 50,000 tagged bat calls, all labelled by specialist volunteers, and achieved an accuracy of over 95% for bat detection. However, the model struggled with the accuracy of bat species classification (Gallacher *et al.*, no date). The workflow of continuous audio recording, a combination of edge and cloud computing, and subsequent classification is the combination of techniques that this paper attempts to emulate through the later methodology section.

2.3. Citizen scientists and non-computer scientists

2.3.1. The importance of citizen scientists

The engagement of citizen scientists and volunteers' involvement in research allows researchers to gain access to studies at a geographical scale that they simply could not

achieve on their own (Dickinson, Zuckerberg and Bonter, 2010). Engaging with citizen scientists is an established practice in bat research in the United Kingdom. The UK Bat's Conservation Trust is the longest-running systematic bat monitoring programme globally (Gallacher *et al.*, no date). Between 1997 and 2012, over 3,500 volunteers assisted in the National Bat Monitoring Programme in the United Kingdom (Barlow *et al.*, 2015).

However, as bat monitoring evolves from more simple counting and listening for bats to the more advanced machine learning techniques, the cost to these citizen scientists rises (Gallacher *et al.*, no date). For example, the SonoBat, a market-leading bat call analysis software, can cost up to USD $1,536.00 for a perpetual license, not including the hardware required to record the audio (*SonoBat – Software for Bat Call Analysis*, no date). Hardware, on the other hand, while cheaper, range in price from something like the handheld Batbox Baton Bat detector at £82.50 to a complete standalone passive audio device like the Song Meter Mini Bat detector priced at £755.00 (*Bat Survey & Monitoring | Wildlife Survey & Monitoring | NHBS*, no date). To allow a higher proportion of citizen scientists to continue in this research, and not simply the wealthiest, the cost of entry must remain low. The stated goal of the Shazam research was "to implement a low-cost technical platform to give ecologists better insight into urban bat activity through continuous longitudinal monitoring" (Gallacher *et al.*, no date). This goal is continued through the research in this paper as well.

2.3.2. Non-computing science background

One of the goals of this paper is to assess whether machine learning and especially machine learning on the edge can be made accessible to those without computing science backgrounds. Machine learning can benefit researchers, citizen scientists, and everyday people in numerous ways when applied correctly. A new application called Edge Impulse aims to bring machine learning to these types of people and will be used to create machine learning models for the bat detection device (*Edge Impulse*, no date).

2.3.3. What is Edge Impulse?

Edge Impulse allows people "to solve real problems using machine learning on edge devices without a PhD in machine learning" (*Edge Impulse*, no date). Edge Impulse simplifies the machine learning process and allows users to create models without writing complex code or, in some deployments, even write code at all. Existing completely online, Edge Impulse

enables users to collect and submit their own data, create and train machine algorithms, test these algorithms, and then deploy them onto edge devices in a simple, straightforward manner.

2.4. So why is this research necessary?

The importance of this research lies in its focus on opening machine learning techniques to those who have not been exposed to it and breakdown some of the potentially intimidating barriers to entry. Furthermore, the paper aims to show that people who have a problem appropriate for machine learning should not be afraid to solve it themselves through free and open-source platforms like Edge Impulse.

Research and science must continue to be democratised and open to as broad an audience as possible. As seen in the work done by the Bat Conservation Trust, when research is available to all new and previously unattainable goals can be achieved.

3. Methodology

3.1. Dataset and study area

The dataset used to build and train the machine learning model came from tagged audio recordings gathered in the Queen Elizabeth Olympic Park in East London from 2017 from fifteen Echo Boxes. The data was recorded as part of the Nature-Smart Cities research project done in conjunction with the Shazam for Bats project (Gallacher *et al.*, no date; *Nature-Smart Cities*, no date).

The dataset consists of 16,000 three-second audio recordings split evenly classified as either including or not a bat call. All audio classification was done using the machine learning model completed in the Shazam for Bats paper. The dataset is held on a UCL server and was accessed through a VPN to conduct the analysis for this paper.

As mentioned in the literature review, the dataset that the Shazam for Bats model was built upon consisted of 50,000 audio samples of bats tagged by citizen scientist volunteers (Gallacher *et al.*, no date). As the dataset used for this paper is based upon audio classified by the Shazam for Bats model, it is an obvious continuation of this research.

Shown below in figure 3.1 are examples of spectrograms created using Sound eXchange (SoX) software of the Shazam for Bats audio recordings, including bat calls (*SoX - Sound eXchange | HomePage*, no date). Shown below in figure 3.2 are spectrograms of audio recordings that do not include bats. The SoX software will be explained in more detail in the software subsection.

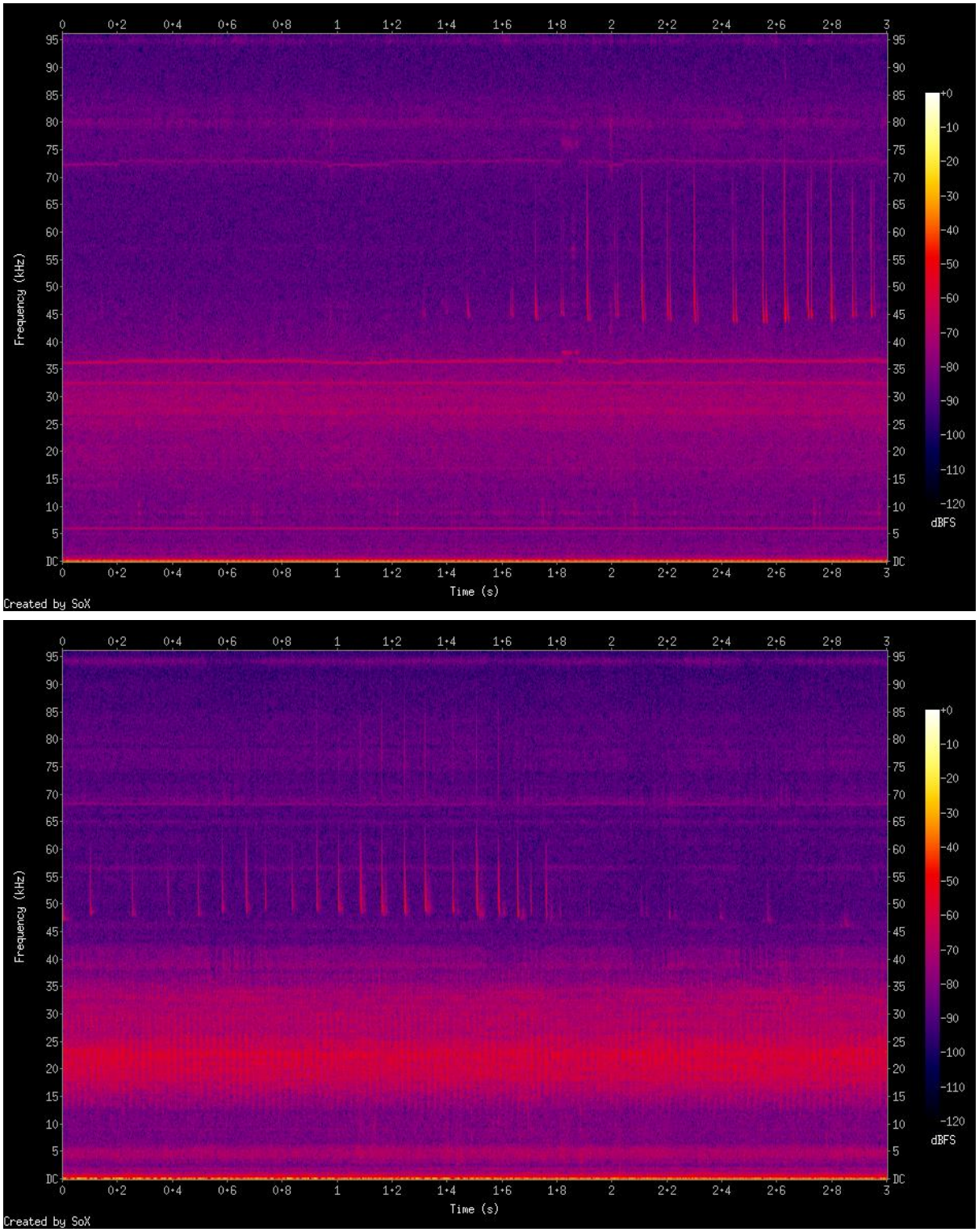


Figure 3.1: Spectrograms created using Sound eXchange (SoX) highlighting examples of audio clips, including bat calls.

As shown above, the distinctive hockey stick-shaped bat calls are apparent in both examples. However, the examples show how they vary in their height, frequency, and strength of

colour. These differences show potential variations in proximity to the microphone, bat species, or time the call was recorded (early, mid, or late call).

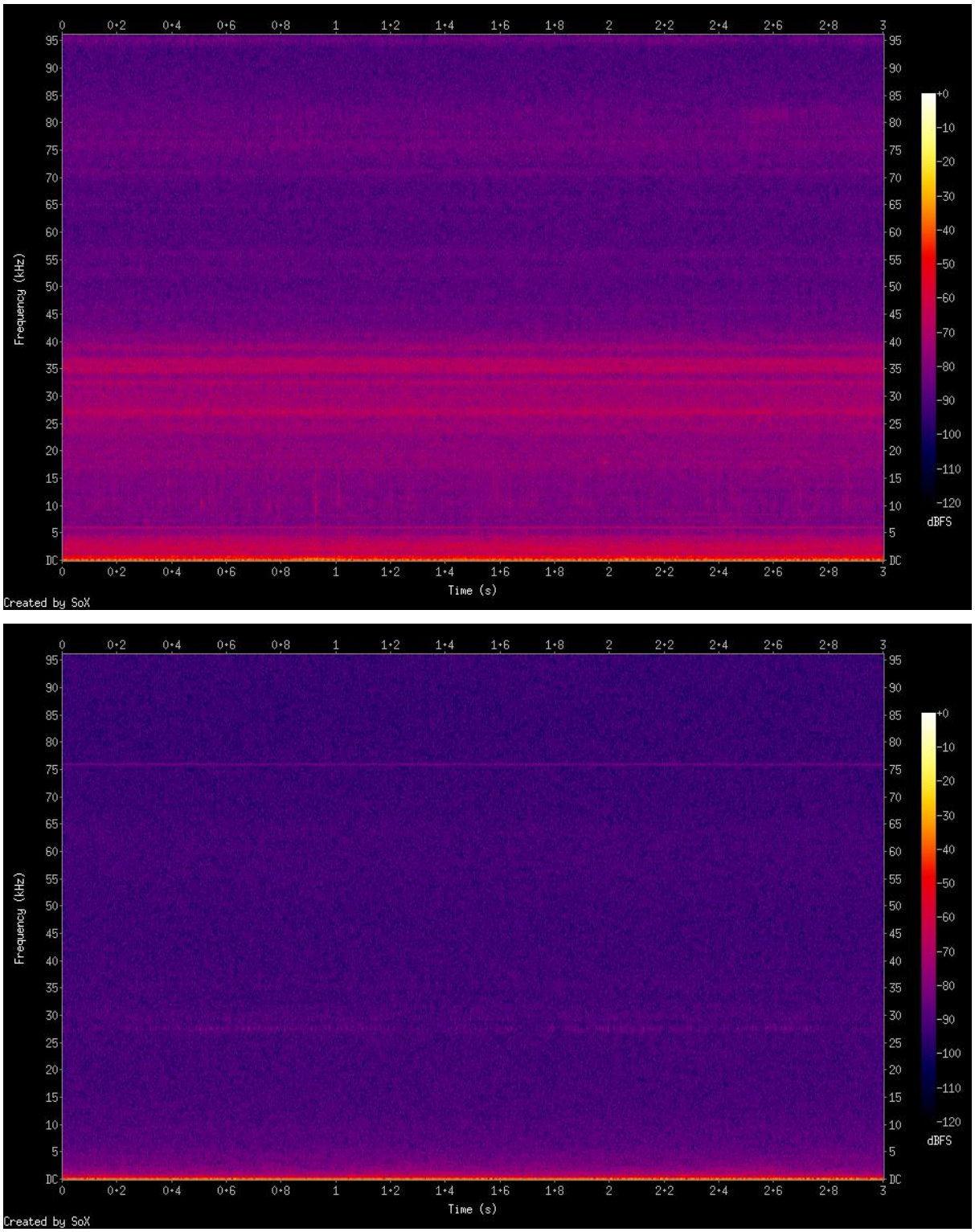


Figure 3.2: Spectrograms created using Sound eXchange (SoX) highlighting examples of audio clips without bat calls.

Figure 3.2 shows a distinct lack of hockey stick-shaped calls in both examples. However, when comparing figures 3.1 and 3.2, the background noise, all parts of the spectrograms that are not the hockey stick-shaped calls, are similar. This background noise includes all other noise present at the different parts of the Queen Elizabeth Olympic Park, like whether pedestrians are in the area or there is construction in the background. This noise will have to be ignored by the machine learning model created later, as it does help differentiate between whether a bat is present.

3.2. Hardware

The hardware section of the methodology will cover the actual devices required to capture, process, and analyse the calls to determine whether a bat was present. Please note that all hardware was chosen in collaboration with UCL and CASA.

As this paper aims to understand the constraints of creating a low-cost and more straightforward device for citizen scientists, one of the main objectives when selecting hardware was ensuring, where possible, low price and opensource hardware were used.

3.2.1. Raspberry Pi

Unlike in the Shazam for Bats project, where an Intel Edison with Arduino breakout board is used, this project utilises a Raspberry Pi (RPi) 4 Model B with 4GB of memory in its deployment (*Raspberry Pi 4 Model B Starter Kit*, no date). RPi's have become ubiquitous in the maker space and have been used in the past already for projects like bat detectors, mobile weather stations or simply as smaller, less expensive alternatives to desktop computers (*Raspberry Pi Bat Project*, no date; 'What is a Raspberry Pi?', no date).

With this in mind, a RPi was chosen for this project because it matches what was needed from a computing power and hardware point of view. Its marketing leading position also means plenty of helpful guides and manuals exist online and can be easily referenced when things inevitably do not work through the building process. A RPi can also run all required software, including all Edge Impulse models, Python and SoX.

While not distinctly an edge device as it requires a mains power supply to run a RPi is still a low price, mobile, and opensource platform. RPi's offer great flexibility and reconfigurability

as they can also function as desktop computers. As mentioned above, if connected to a monitor through one of their two HDMI ports, they also have both Wi-Fi and hardwired internet access, as well as multiple USB ports to connect devices like a mouse, keyboard, microphone or anything else that may be required (The Raspberry Pi Foundation, no date).

A MicroSD card is required to run a RPi as it holds the operating system for the RPi, now known as Raspberry Pi OS (previously Raspbian), and any other files or software that may need to be saved or run on the device. For this project, a 64GB MicroSD card was used, which would hold approximately 198 hours and 57 minutes of audio recordings ('Audio & Memory Cards: How much record time do I have?', 2011).

A RPi starter kit, which includes everything needed to start working with a RPi apart from a monitor, mouse and keyboard, is priced at £78.50 from the Pi Hut (*Raspberry Pi 4 Model B Starter Kit*, no date). This package includes the following:

- Raspberry Pi 4 Model B
- 32GB MicroSD
    - Please note a 64GB MicroSD, as used in this project, costs £14.62 on Amazon (*SanDisk Ultra 64 GB microSDXC Memory Card + SD Adapter with A1 App Performance Up to 100 MB/s, Class 10, U1 - SanDisk*, no date)
- Official Raspberry Pi Case
- 2-metre black Micro-HDMI cable
- Official Raspberry Pi Power Supply

Figure 3.3 below shows what is included in the RPi starter kit.

Figure 3.3: Raspberry Pi starter kit from the Pi Hut, image provided by the Pi Hut website (Raspberry Pi 4 Model B Starter Kit, no date).

A Raspberry Pi 4 Model B with 4GB of memory, not including the other parts of the starter kit, is priced at £54 from the Pi Hut (*Raspberry Pi 4 Model B*, no date).

3.2.2. Dodotronic UltraMic UM192K

As discussed in the literature review, a high-quality microphone is required to detect bats as bat calls are extremely high in frequency, starting at 20 kHz and ranging to 108 kHz (*Frequencies | Staffordshire Bat Group*, no date). Whilst a cheaper, off the shelf microphone could have been used, like those found in mobile phones, it would have required amplification circuitry to be built for the bats to be heard accurately.

The Dodotronic Ultramic UM192K is an ultrasonic USB microphone that works across Windows, Linux, and Apple devices. It has a sampling rate of 192 kHz, almost 90 kHz above the highest bat call frequency in the United Kingdom (Pelicella, no date). This fits with Nyquist's Theorem that states, "for an accurate digital representation of a sound wave, the sample rate must be, at least, two times bigger than the highest frequency going to be

recorded (*Sample rate : What is it? Which to use? What is the best? 44,1k vs 48k*, 2019)." The microphone attaches simply to any device using a mini-USB connection into the microphone and then a USB-A connection into a device like a RPi or laptop.

The Dodotronic Ultramic UM192K can be purchased from the Dodotronic website for €200.00 or £169.52 based on the exchange rate on 8 August 2021 (Pelicella, no date). The microphone is shown below in figure 3.4

Figure 3.4: Dodotronic Ultramic UM192K, image provided by the Dodotronic website (Pelicella, no date).

3.2.3. Cost

The software used is open-source and free, so the only cost of running the bat detection device is the hardware. The cost of the total package is calculated below:

Device	Price
Raspberry Pi Starter Kit	£78.50

Dodotronic Ultramic UM192K	£169.52
Total Cost	**£248.02**

Table 3.1: Total cost breakdown of the bat detection device.

This price compares quite favourably to the bat detectors mentioned in the literature review, which at the cheaper end were priced around £82.50 for a handheld device. However, for a passive listening device, the price jumped up to £755.00 (*Bat Survey & Monitoring | Wildlife Survey & Monitoring | NHBS*, no date). While not inexpensive at £248.02, over 68% of the total price is included in the microphone. The RPi itself is cheaper than one of the handheld bat detection devices and offers much more flexibility than only being a simple bat detector if a citizen scientist's interests changed in the future.

3.3. Software

Three pieces of software were used to create the bat detection device: Edge Impulse, SoX and, Python. While they each will be discussed in greater detail further down, overall, Edge Impulse handles the machine learning processes of the device while SoX and Python handle audio and data processing as well as the automation of the device.

3.3.1. Sound eXchange (SoX)

Sound eXchange (SoX) describes itself as the "Swiss Army knife of sound processing programs (*SoX - Sound eXchange | HomePage*, no date)." However, for the bat detection device, SoX will render spectrograms from the recorded audio files. SoX is primarily run through the command line and, as such, can be accessed through Python to perform the same activities. The code to create the spectrograms was based on examples in the Connected Environments GitHub repository with helper scripts to process the audio files (*echopibox/ultramic2spectrogram.py at main · djdunc/echopibox*, no date).

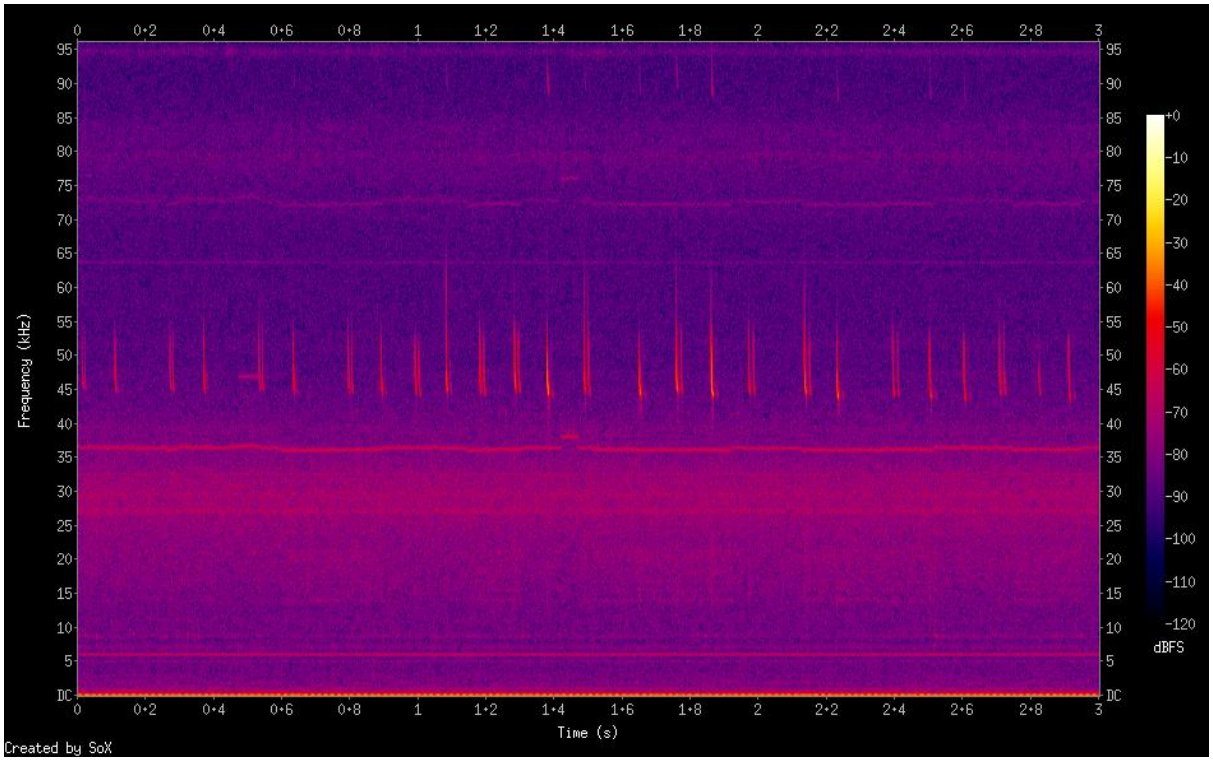


Figure 3.5: Sound eXchange (SoX) spectrogram rendering of an audio file, including a bat call.

Figure 3.5 shows a SoX rendering like those seen in figures 3.1 and 3.2. Again, breaking the spectrogram down further though the x-axis shows time, the y-axis shows frequency (in kHz in this spectrogram). At the same time, the z-axis is represented by the spectrogram's colour or intensity (*SoX Main Manual*, no date).

In figure 3.5, the bat calls are very apparent in their unique hockey-stick shape. Along the y-axis, it is evident that the calls take place between approximately 45 kHz and 65 kHz matching the frequencies expected by bats in the United Kingdom (*Frequencies | Staffordshire Bat Group*, no date). Based on SoX's description of the z-axis, the bat calls are especially apparent by their colour and intensity, as well as they come through quite clearly against the purple background noise with their much more vibrant pink to red colour. The absence of a sharp intensity of noise is quite evident in figure 3.6 below, where there is simply the more monotonous purple without the intense pink to red colours seen in figure 3.5.

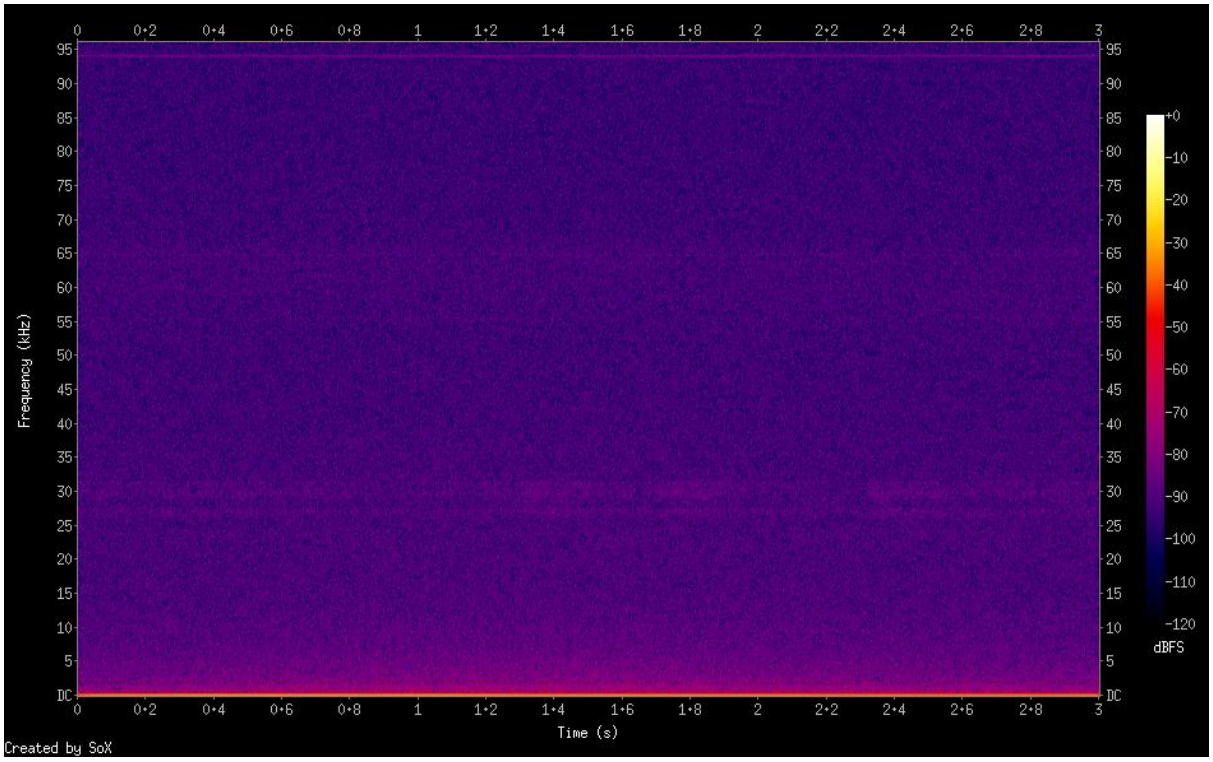


Figure 3.6: Sound eXchange (SoX) spectrogram rendering of an audio file, not including a bat call.

The spectrograms created by SoX will then be the images used by Edge Impulse for both training and testing purposes and the basis of the images fed into the model when it detects bats on new and unknown audio.

3.3.2. Edge Impulse

As mentioned in the literature review, Edge Impulse aims to simplify the machine learning user for its users. It simplifies the process by allowing users to perform three main activities in one simple to use user interface:

1. Data acquisition, tagging and splitting between training and test data
2. Machine learning model building, training, and testing
3. Simplified deployment to edge devices

3.3.2.1. Data acquisition, tagging and splitting between training and test data

Regardless of the type of model being built in Edge Impulse, one of the first steps to be completed is adding data so that the model can learn. Edge Impulse can receive data in six different ways, shown below in figure 3.7.

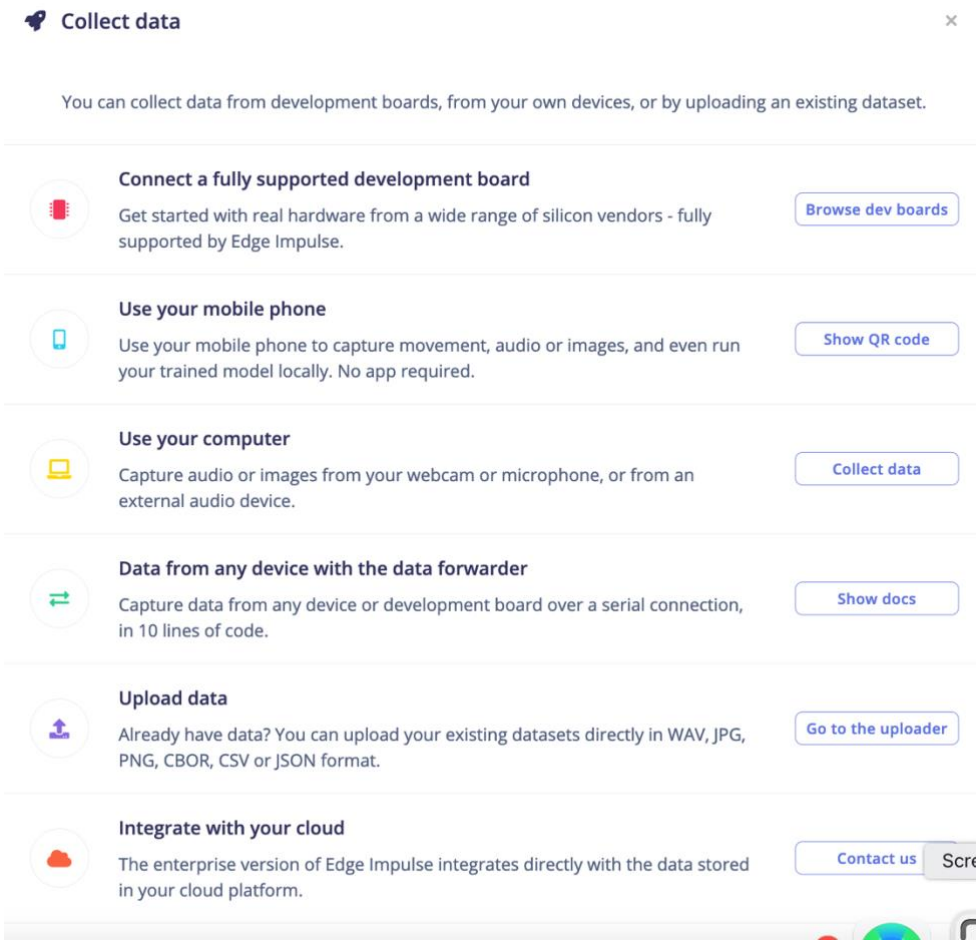


Figure 3.7: Screenshot of Edge Impulse's data acquisition options (Edge Impulse, no date).

Data can be added directly from a development board, like a RPi, a mobile device, a computer through a webcam, microphone, or other devices, captured through a data forwarder, uploaded directly through Edge Impulse's uploader, or integrated with a cloud platform.

As the user adds data, it must be tagged with the appropriate label. For the bat detection device created for this model, two labels were used bat or no bat indicating the presence of a bat all or the absence of a bat call.

Data was added into Edge Impulse through their uploader as either a .png or .wav file for this project. How these two types of files were processed and experimented on will be explained in a subsequent part of the methodology section. As the data is being uploaded, it is also split between training and test data. Training data is the data that the model learns from and allows

it to make, hopefully, accurate predictions. Test data is data that the model is not trained on and is unseen until the model is tested to validate its accuracy once it has been built.

3.3.2.2. Machine learning model building, training, and testing

The next step in generating a model on Edge Impulse is to create an impulse. In figure 3.8 below, a screenshot shows an impulse designed to classify an image. However, regardless of the processed data type, audio, image, or other, impulse creation follows the same format. The similarities are apparent compared to figure 3.9, which shows an impulse created to analyse audio input.

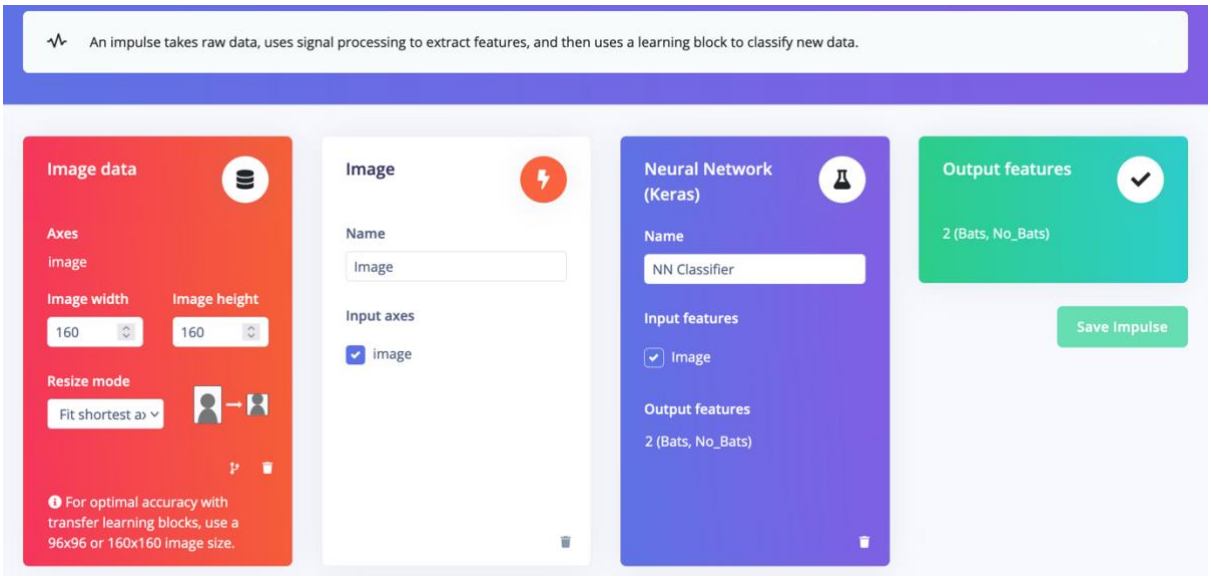


Figure 3.8: Screenshot of an impulse created in Edge Impulse to classify images (Edge Impulse, no date).

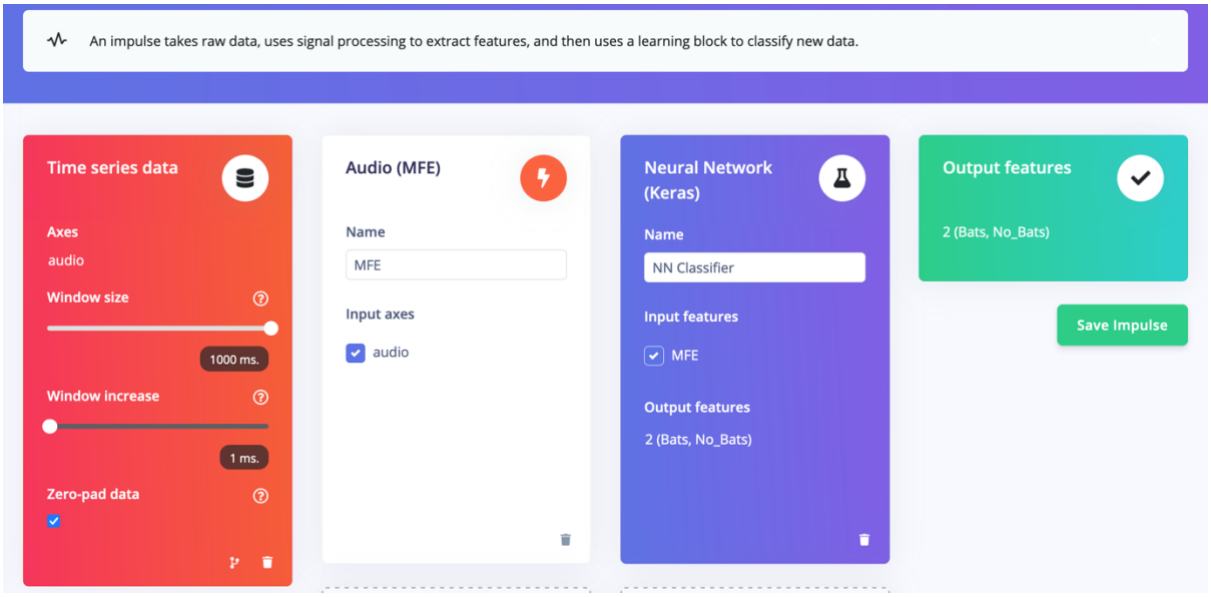


Figure 3.9: Screenshot of an impulse created in Edge Impulse to process audio time-series data (Edge Impulse, no date).

Both impulses create a standard shape or size of the input data, as seen on the left-hand side. However, how the data input is shaped to develop the model's features depends on the type of analysed data. For image classification, this begins with shaping all images to a specific size, and for audio or time-series data, this means creating equal window sizes for analysis.

Following creating an impulse, the next step is to set the analysed data's parameters further. For images, this is a simple choice as the size was established during the impulse creation, 160 x 160 in figure 3.8, so the final thing to do is choose whether the image should be classified as RGB or grayscale. However, for audio data, more variables can be tweaked. The differences can be seen in figures 3.10 and 3.11, respectively.

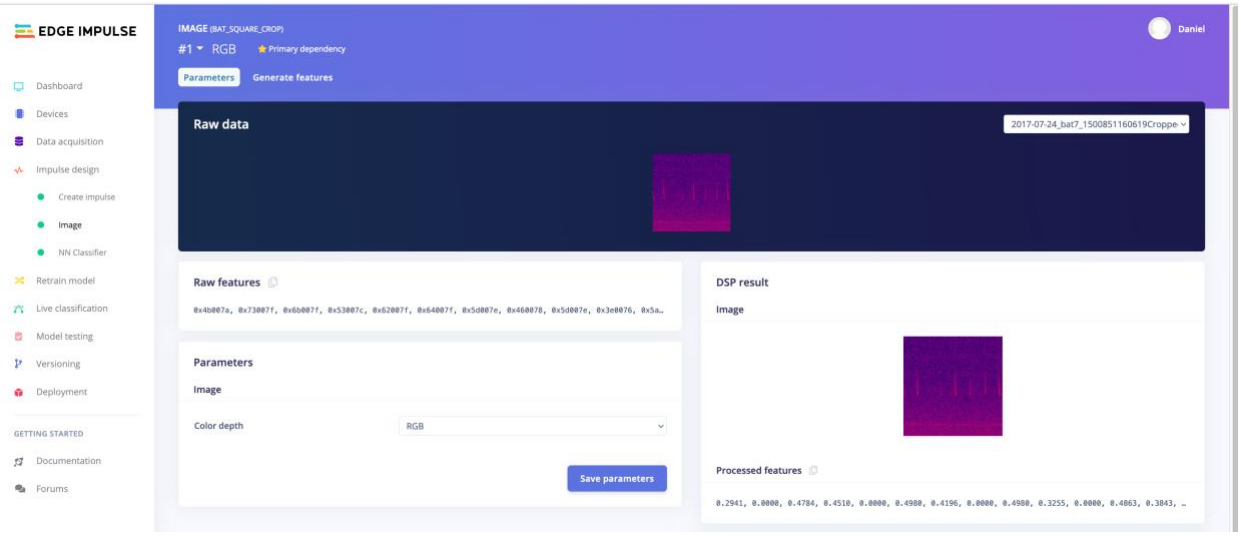


Figure 3.10: Screenshot of the parameter selection page for an image classification model on Edge Impulse (Edge Impulse, no date).

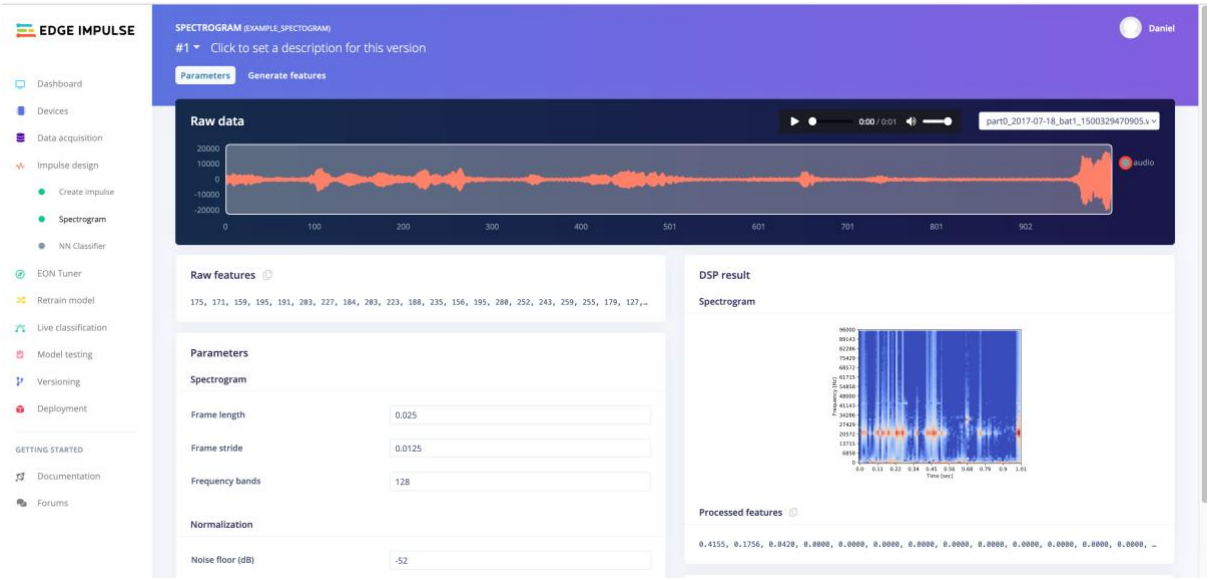
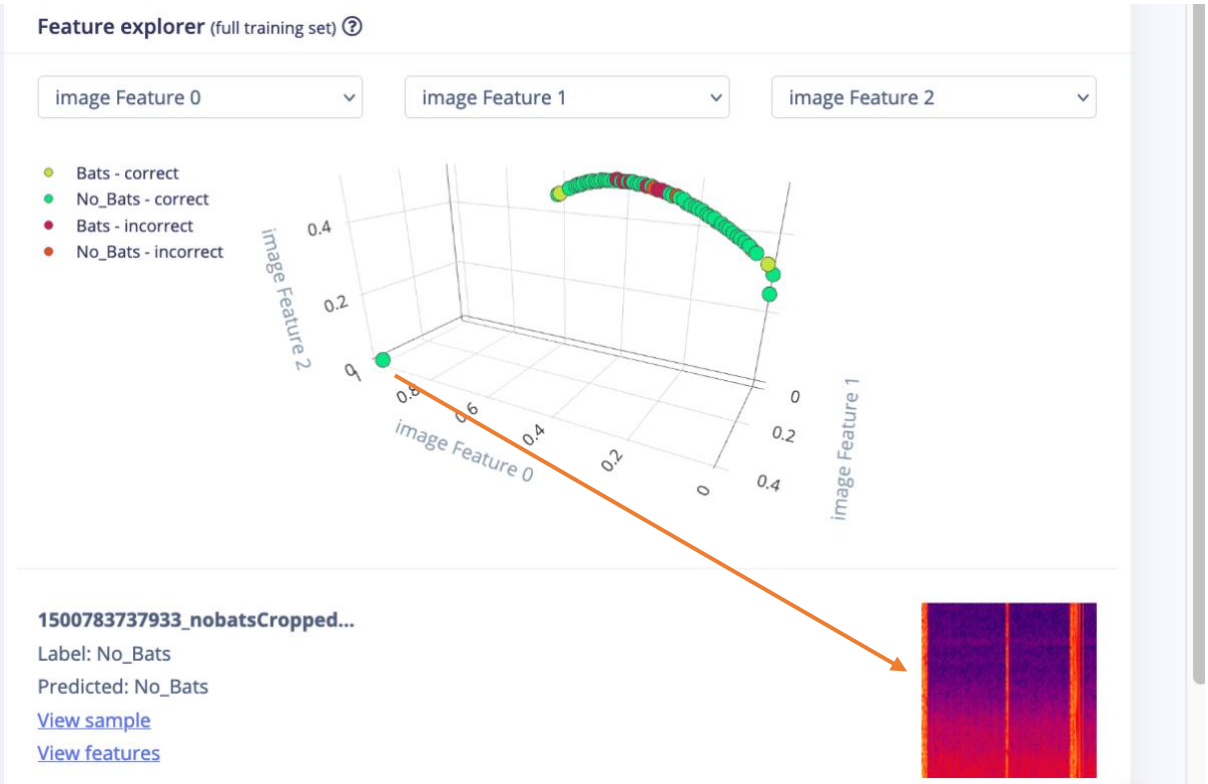


Figure 3.11: Screenshot of the parameter selection page for an audio classification model on Edge Impulse (Edge Impulse, no date).

For images, only two variables exist, RGB or grayscale, but seven variables are available for audio classification ranging from frame length on the top of the left-hand side of figure 3.11 to the noise floor (dB) variable in the normalisation section. These different options will be explained more when the models deployed for the bat detection device are described. Furthermore, in both figures 3.10 and 3.11, Edge Impulse also creates the processed version of the input that will be processed. In figure 3.10 this is the processed square image of the spectrogram on the right-hand side of the screenshot, wherein figure 3.11 is the spectrogram that Edge Impulse creates based on the audio input using, in this example, spectrogram features. This spectrogram differs in appearance created in SoX, but the high frequency of bat calls can be seen just above 20,000 Hz.

Once the features have been generated, Edge Impulse enables the features to be explored using an interactive graph seen below in figure 3.12. This graph allows the user to understand how the data being modelled is grouped using the features generated by Edge Impulse and to supplement or remove potentially noisy data if needed. This graph extrapolates the features it pulls from the spectrogram or audio file and plots them along three axes compared to the other samples. The feature explorer may help notice specific microphones with high background noise levels that make them different from the others and could influence the

results as they stand out once plotted on this graph. For example, looking at the screenshots below, the differences appear in the bottom right-hand side of the screenshot, where the top one shows the image from the circle far to the left with high levels of noise compared to the one to the right, which offers a distinct lack of background noise. The orange arrows highlight both examples.



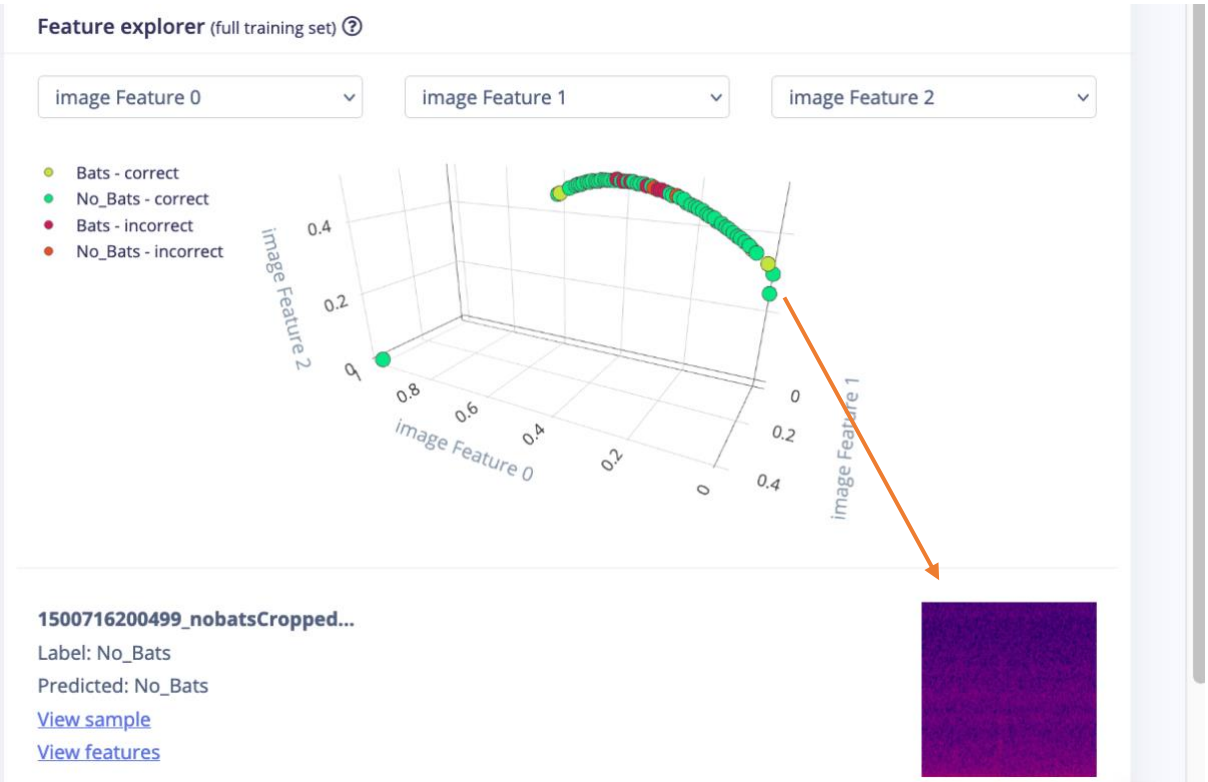


Figure 3.12: Screenshots of the feature explorer generated by Edge Impulse with added orange arrows highlighting the examples (Edge Impulse, no date).

The next step in Edge Impulse is to use the application to build a machine learning model. Again, like making the Impulse and the image or the parameters and features, this is done through a simple user interface. However, for more advanced users, the models can be customised directly online in a Keras notebook or taken offline, edited and reuploaded as an iPython notebook (*Keras: the Python deep learning API*, no date). In both the advanced examples, they can work directly with TensorFlow and Keras (*TensorFlow*, no date). The user interface for the image classification model is shown below in figure 3.13.

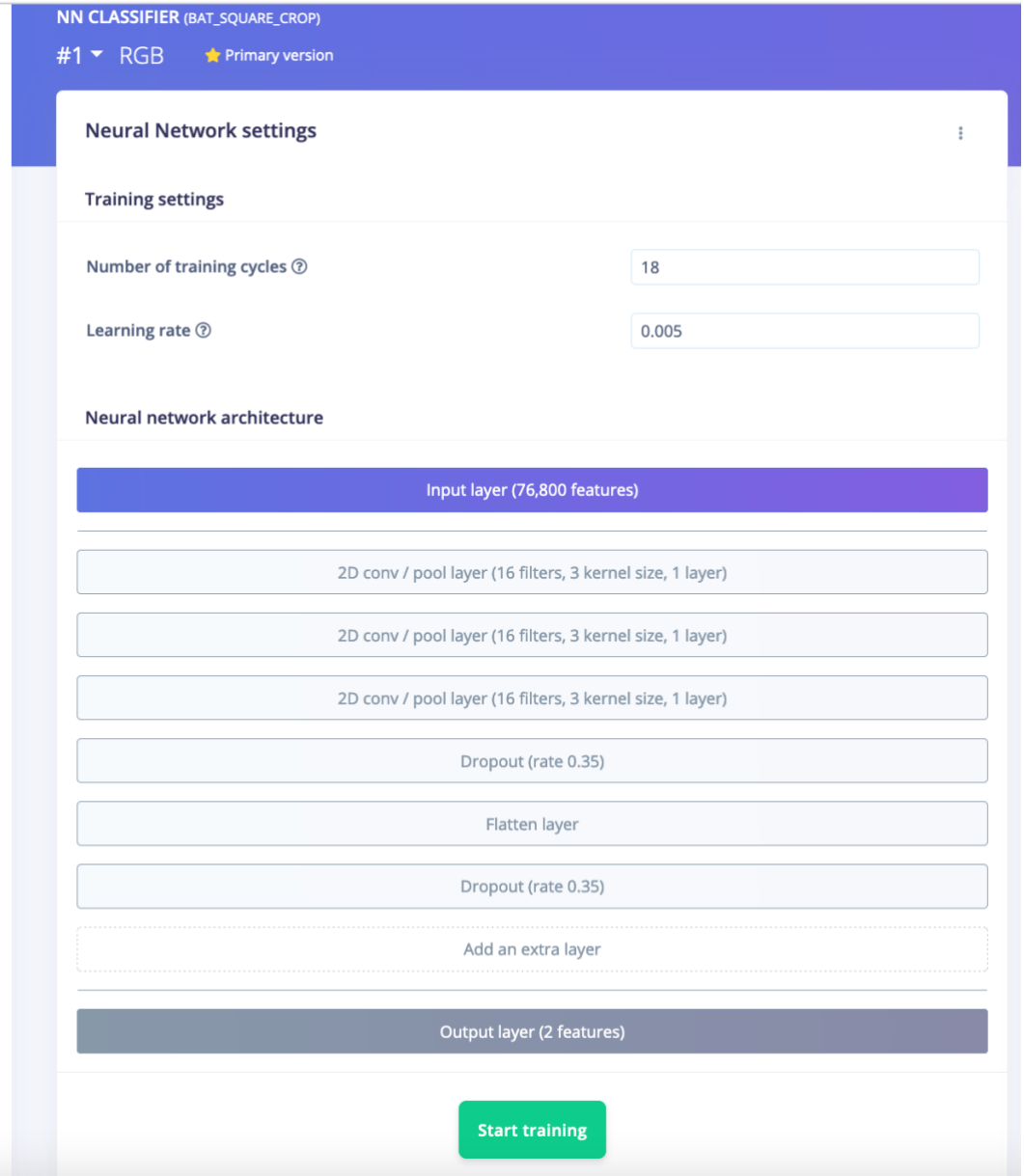


Figure 3.13: Screenshot of the configurable parameters for building a machine learning classifier using the Edge Impulse interface (Edge Impulse, no date).

Starting from the top of figure 3.13, the number of training cycles and the learning rate can be adjusted to match the model's needs. Then, moving down the figure, the number of features is defined by the image created in figure 3.10. In this example, the 76,800 features are 160 by 160 pixels multiplied by the three colours: red, green, and blue (RGB). The layers below the features are all freely adjustable or removable as benefits the model being made. For example, in the 2D conv / pool layers, the number of filters, the kernel size, and internal layers are also adjustable.

Figure 3.14 below is a screenshot of the configurable Keras interface within Edge Impulse. The Keras interface is much closer to a more traditional computer science approach as it is a piece of Python code held within Edge Impulse. This Keras notebook may put someone off looking to solve a simple problem in their domain with a machine learning approach because of its intimidating nature. In contrast, the Edge Impulse interface is much more user friendly, understandable, and approachable.

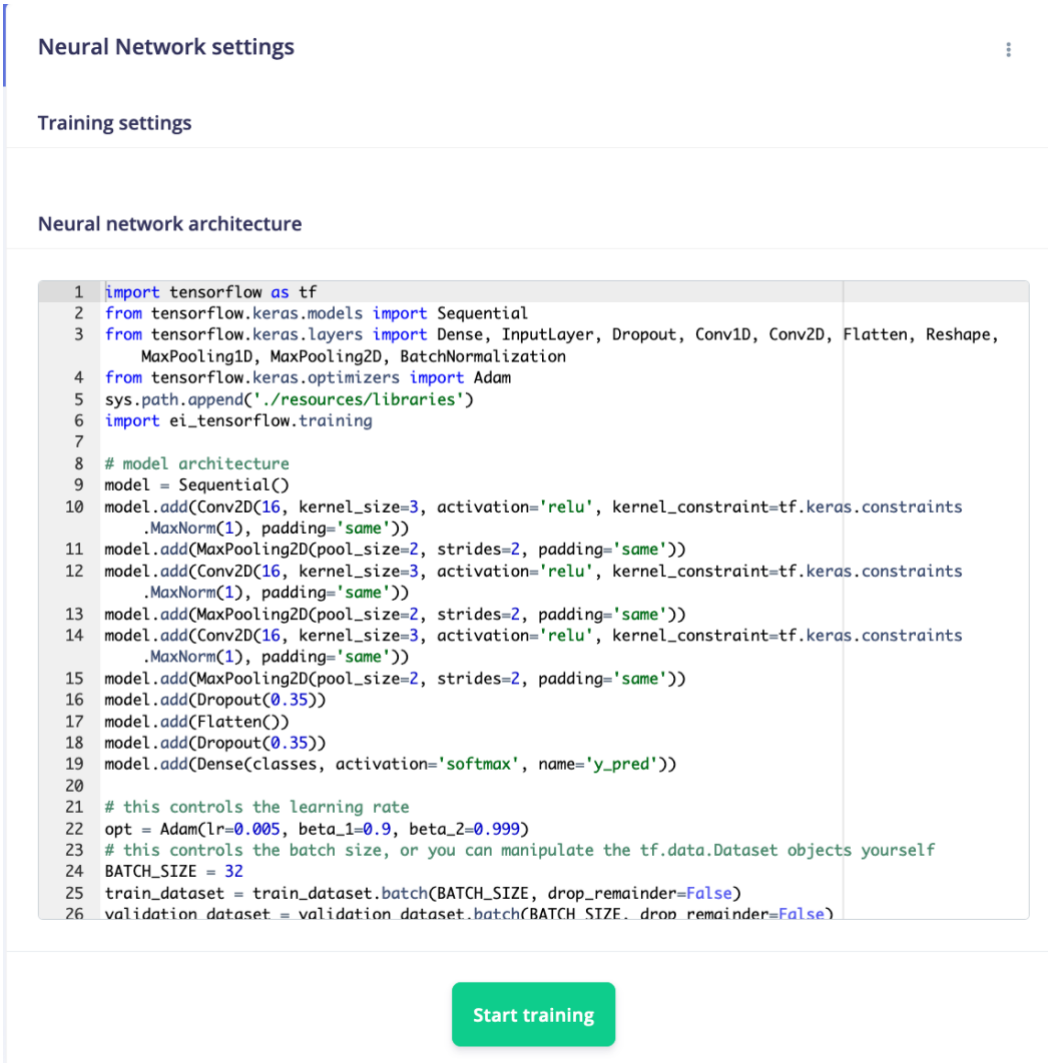


Figure 3.14: Screenshot of the Keras interface in Edge Impulse (Edge Impulse, no date).

Lastly, as Edge Impulse separates the submitted data into training and test data, training and testing the model becomes very straightforward. Through the click of a button, Edge Impulse first trains the model on the training and validation data. Once the training is complete, the model can then be run on the test data. Figure 3.15 shows a screenshot of Edge Impulse's

output of test data classification, which shows the summarised test results and a new feature explorer highlighting the correct and incorrect test data classifications on one page.

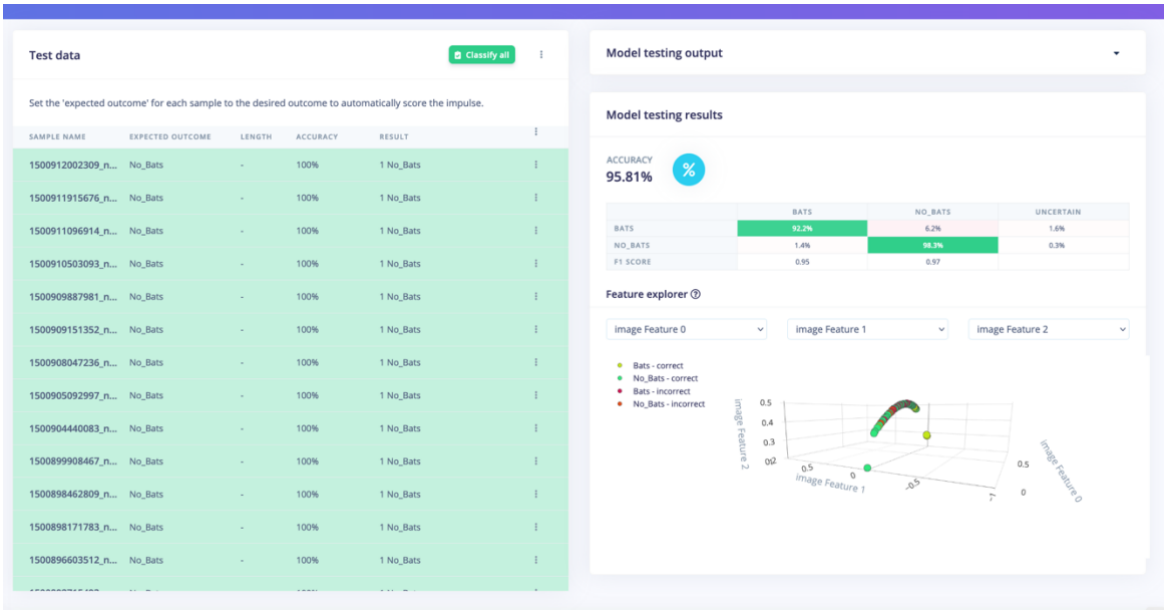


Figure 3.15: Screenshot of the test data and results page on Edge Impulse (Edge Impulse, no date).

3.2.2.3. Simplified deployment to edge devices

Edge Impulse simplifies the deployment of the machine learning models into libraries that edge devices can use. Models can be deployed through C++ libraries, Arduino libraries, WebAssembly and directly onto specific firmware options like OpenMV cameras. Each deployment option has precise and easy to follow instructions on the Edge Impulse website (*Running your Impulse locally*, no date). Figure 3.16 shows all the deployment options below.

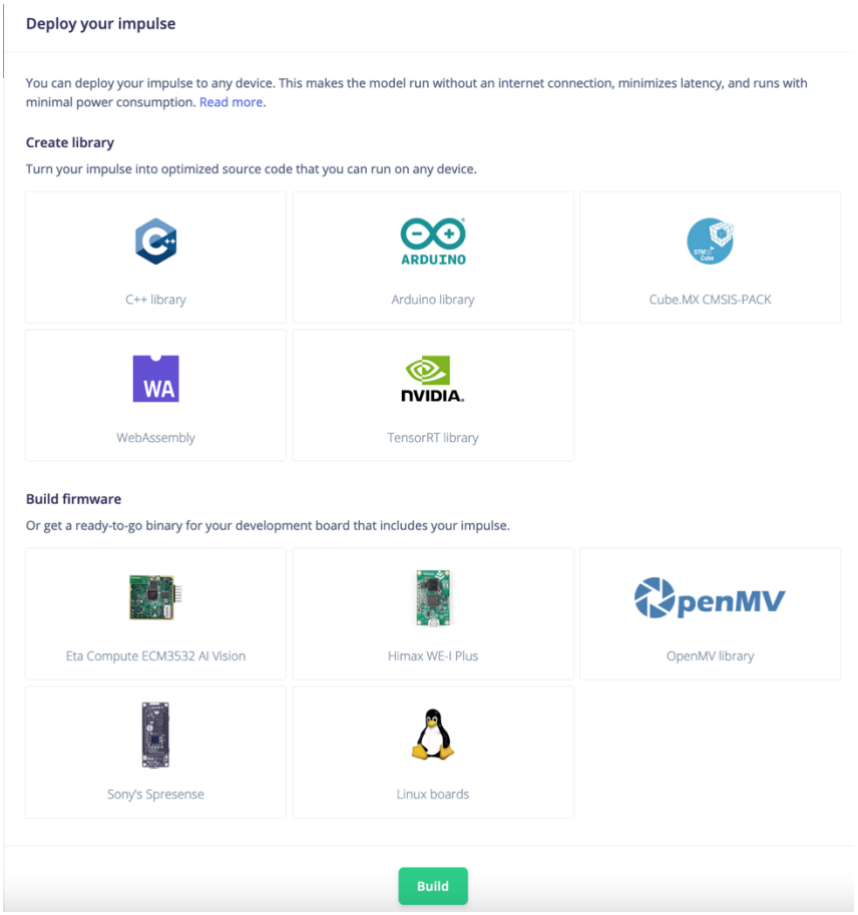


Figure 3.16: Screenshot of the Edge Impulse deployment options (Edge Impulse, no date)

3.3.3. Python

Python was used in two main ways to build the bat detection device. Firstly, it is used heavily in the processing of data for analysis. Python combines all the tasks that take raw audio and transform it into something that the machine learning model can analyse (*echopibox/ultramic2spectrogram.py at main · djdunc/echopibox*, no date). Secondly, it is used to automate the process between recording and the bat call classification. How Python performs these functions will be described next in the implementation journey and deployment section.

3.4. Implementation journey and deployment

3.4.1. Audio input options

As stated in the research question, the bat detector journey and deployment goal were to create a bat detector that could compare to the bat detectors developed in the Shazam for Bats

project but using more accessible, citizen scientist friendly means. After the decision was made to use Edge Impulse, two different classification options became apparent:

1. SoX generated spectrograms and an Edge Impulse image classification model
2. Edge Impulse audio classification model

Bat detection models were made for both options and deployed to RPi devices as prototypes where possible. How the two options were developed and deployed differed and will be explained in separate sections.

The explanations will follow the following three steps:

1. Audio input and data preparation
2. Edge Impulse model building
3. Prototype deployment

3.4.2. Option 1: SoX generated spectrograms and an Edge Impulse image classification model

The first option explored was using SoX to generate spectrograms and then utilising Edge Impulse to create an image classifier.

3.4.2.1. Audio input and data preparation

The first step in preparing the audio files to train the model was to create the spectrograms using SoX. To make the spectrograms, SoX must be installed on the computer. Once installed, SoX can be accessed through the command line, terminal or within a Python script. The images that SoX creates can be seen in figures 3.1, 3.2, 3.5 and 3.6.

Initial experiments were ran using SoX spectrograms as created without editing. However, using these unedited spectrograms did not achieve results comparable to the Shazam for Bats results. Furthermore, the accuracy never reached 86% on test data when classifying these unedited spectrograms after attempting over twenty-four model parameter combinations.

To improve model accuracy, the images were cropped using Python. First, the spectrograms were cut from their original shape to a 160x160 pixel square to reflect the optimum size for

Edge Impulse image classification. After a manual review of the spectrograms with bat calls, the optimum space to take a cut began just below 35 kHz and extended to just below 65 kHz in the centre of the spectrogram. The reason for cropping at this location is that almost all bat calls either extend up from 20 kHz to above 35 kHz or, on the other side, begin below 65 kHz and extend upwards.

Figure 3.17 shows an original spectrogram on the left and then the cropped image on the right. While the image is significantly smaller, the bat calls remain distinct and prominent. Below the original image and the cropped image is the original image with a white space representing where the crop was taken.

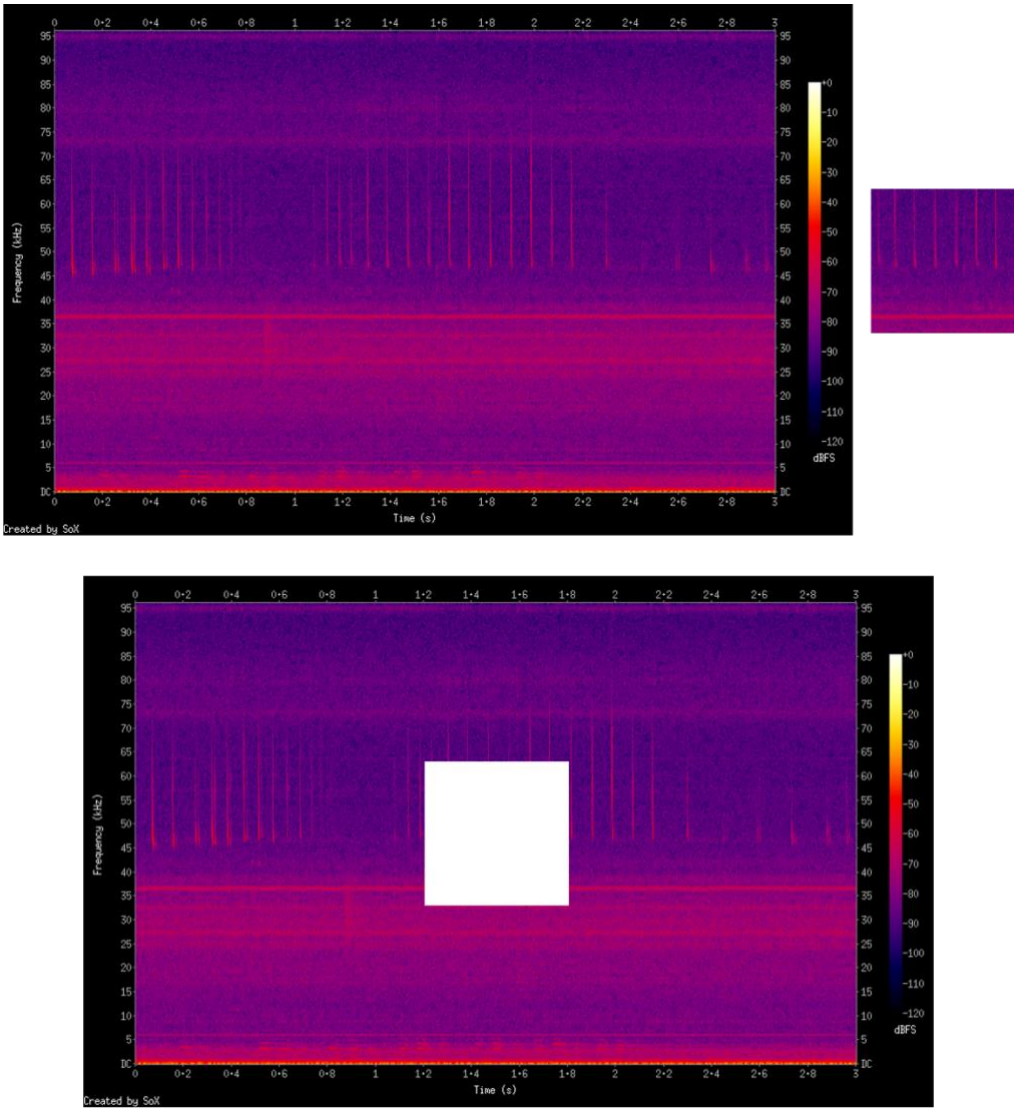


Figure 3.17: Image showing an original SoX spectrogram, a square crop used to generate the Edge Impulse model and an original SoX spectrogram highlighting the location where the cropped image was taken.

Once each spectrogram was cropped, a manual check was done to ensure that a bat call was present in each new image. A review was undertaken because, in some original spectrograms, the bat calls were left or right of where the crop was taken and therefore did not include a bat call once cropped. The images without bat calls were then removed from the dataset before being uploaded into Edge Impulse to build the model. Figure 3.18 highlights three examples of the type of cropped images that were removed. After the review, 556 images were removed and not added to the dataset because they did not include bat calls.

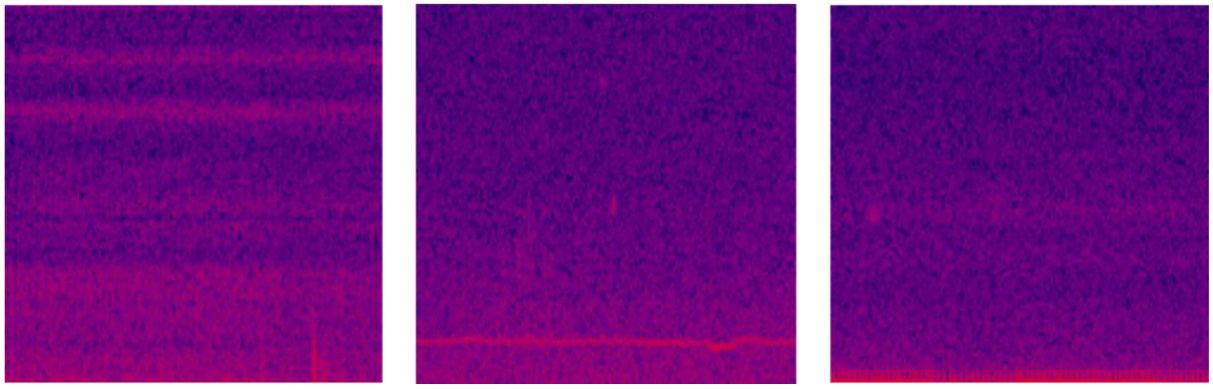


Figure 3.18: Three examples of cropped images removed from the data used to build the Edge Impulse because they do not contain bat calls.

Table 3.2 below outlines the makeup of the dataset used to create the model in Edge Impulse.

	Images with bat calls	**Images without bat calls**	**Total images**
Training data images	1,132	1,583	**2,715**
Test data images	238	356	**594**
Total number of images	**1,370**	**1,939**	**3,309**

Table 3.2: Breakdown of the bat spectrogram dataset used to build the bat detection model.

3.4.2.2. Edge Impulse model building

Dozens of model parameters were tested for image classification, especially when testing the original uncropped images. The learnings from these tests were then continued onto testing the new cropped images. Over twenty tracked trials were completed trying different

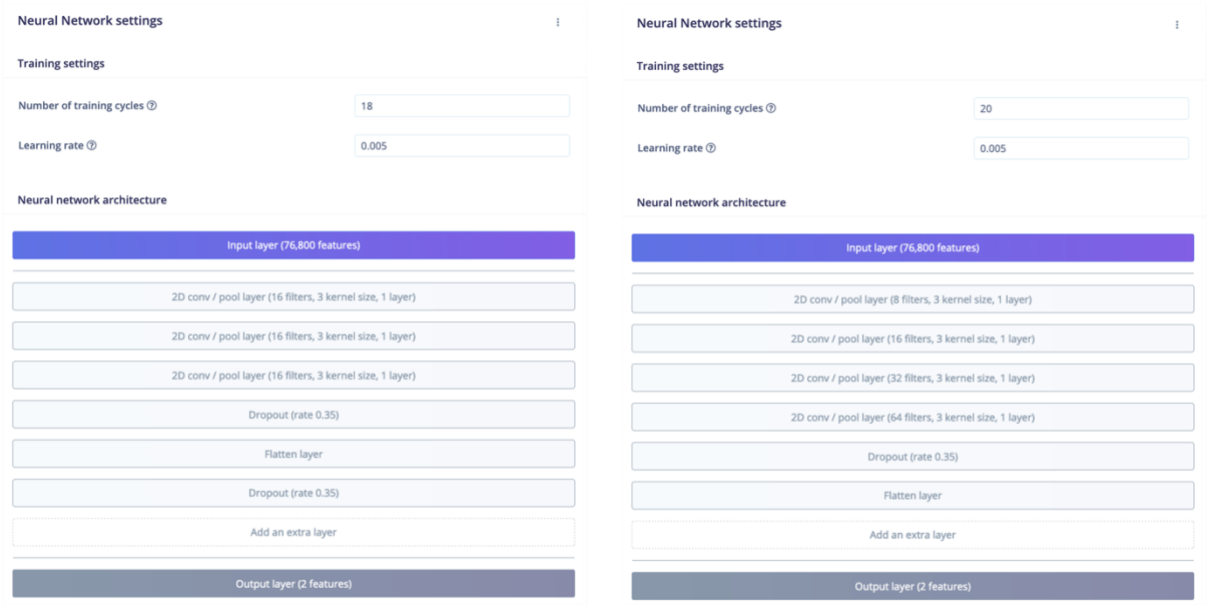
combinations of RGB or grayscale, varying the number of layers, and tweaking learning rates and kernel sizes. A table of the testing results can be seen in Appendix A as a link to GitHub. Please note that this list is not exhaustive and that not every test done was recorded and that tests were done on different SoX outputs, including the square crops and the entire rectangular outputs. In the end, however, manual testing in Edge Impulse proved outdated at the end of July 2021 when Edge Impulse introduced EON Tuner.

EON Tuner is an Edge Impulse tool that automates the search for the most accurate model parameters. While as of August 2021, EON Tuner does not directly work on image classification, it does work on audio classification models. This is because both models, audio and image classification, use neural networks to classify their respective data inputs. This means that the optimised architecture found by EON Tuner for the audio data could then be brought over to the image classifier and tested against the best manually found model.

In the end, the audio classification architecture was 3.86% more accurate on the image test data than the best performing manually found model. The new model found by the EON tuner, which was deployed to the prototype, was 97.6% accurate on validation data and 99.49% correct on test data. Figure 3.19 below breaks down the model architecture for both the best manually found performing model and the best performing model found by EON Tuner. Both models use RGB images and not grayscale. Table 3.3 below breaks down their accuracy on training and test data.

	Training Data Accuracy	**Test Data Accuracy**
Manually found best performing model	96.9%	95.81%
EON Tuner found best performing model	97.6%	99.49%

Table 3.3: Comparison of the accuracy of the best performing models found manually and by the EON Tuner.



Manually found best performing model

EON Tuner found best performing model

Figure 3.19: Comparison of the model architecture of the best performing models found manually and by the EON Tuner.

3.4.2.3. Prototype deployment and testing

Deploying the image recognition was less straightforward than deploying the audio detection model. It was less straightforward because Edge Impulse image detection models expect a camera sensor to provide the visual input, but in this deployment, there is no camera used. So, to run the image classifier model without a camera on new audio several steps had to be taken to allow it to understand the input.

For the image classifier model, a feature is represented by a colour hex code. A colour hex code is a hexadecimal way to represent a colour in RGB format (*What is a Color Hex Code? - Definition from Techopedia*, no date). The Edge Impulse model classifies an image by reading a string of 76,800 hex codes separated by commas left to right and top to bottom, representing the 160x160 RGB image.

The goal for the prototype was to take the raw audio input and have a script that classifies the entire audio piece and identifies bat calls with their time stamps. To start building the prototype, the first step was exporting the model from Edge Impulse as a C++ library to be used on both a laptop and a RPi. Then, after following the deployment instructions, a working model was installed on both the computer and RPi (*Running your Impulse locally*, no date).

With the installed model, the next step was to create a Python script, which can be seen in a link to GitHub in Appendix B, that automates taking a recording and transforming it into features subsequently run and classified by the newly built model. Listed below are the steps that the Python script went through to complete this process:

1. **Clipping audio to 1-second .wav files** – the first step was to use Python to convert a long audio file into several 1-second .wav files. The package Pydub was used to complete this task.
2. **Convert the 1-second .wav files to spectrograms and crop to the appropriate size –** this was done using SoX to create the spectrogram and then the Python package Image to crop the images.
3. **Convert cropped spectrograms to colour hex code features –** this was done using the Python packages NumPy and the matplotlib package imread. While imread extracts the colour hex codes, the remainder of the code prepares the hex code string in a format that the Edge Impulse model understands.
4. **Run the model on the feature outputs and identify bat calls –** The remainder of the code runs the model on the hex code feature strings that have been created and classifies each one as having a bat call present or not. The classifications and confidence in the bat call and the timestamp are then exported into two .csv files. One of the .csv files contains only the timestamps with identified bat calls, whereas the second includes all time stamps.

A screenshot of the positive call .csv file can be seen below in figure 3.20. The results of this model will be discussed in greater detail in the experiments and results section.

	A	B	C	D	E	F
1		time	filename	bats_present	confidence	
2	400	400	bat2_24june	TRUE	0.922164	
3	401	401	bat2_24june	TRUE	1	
4	402	402	bat2_24june	TRUE	1	
5	403	403	bat2_24june	TRUE	0.908316	
6	405	405	bat2_24june	TRUE	0.895165	
7	408	408	bat2_24june	TRUE	0.999899	
8	409	409	bat2_24june	TRUE	0.969117	
9	411	411	bat2_24june	TRUE	0.999997	
10	412	412	bat2_24june	TRUE	1	
11	413	413	bat2_24june	TRUE	1	
12	414	414	bat2_24june	TRUE	1	
13	415	415	bat2_24june	TRUE	1	
14	417	417	bat2_24june	TRUE	0.999258	
15	418	418	bat2_24june	TRUE	1	
16	419	419	bat2_24june	TRUE	0.99936	
17	422	422	bat2_24june	TRUE	1	
18	423	423	bat2_24june	TRUE	0.999991	
19	424	424	bat2_24iune	TRUE	1	

Figure 3.20: Screenshot of the positive bat calls .csv file generated by the bat detection script.

3.4.3. Option 2: Edge Impulse audio classification model

The second option explored was using Edge Impulse's built-in audio classifier models.

3.4.3.1. Audio input and data preparation

Preparing the audio clips to be used in Edge Impulse's audio classification model was much more straightforward than the image classification model. The first step taken was to split all the three-second clips into three separate one-second clips. The audio splitting was done using the Python package Pydub. The code can be found in Appendix B in a link to the GitHub repository.

Once split and before adding to Edge Impulse to create the model, a review was done of the one-second clips to be added to ensure that they did include bat calls. Like in the image classifier prototype, this was done because the bat calls are spread through the three seconds, so each new clip may not contain a bat call.

After this review of the audio clips, they were added to Edge Impulse. Table 3.4 outlines the split of the data used to create the model.

	Time with bat calls	Time without bat calls	Total time
Training audio time	6m 43s	7min 12s	**13m 55s**
Test audio time	1m 38s	2m 9s	**3m 47s**
Total audio time	**8m 21s**	**9m 21s**	**17m 42s**

Table 3.4: Breakdown of the bat audio dataset used to build the bat detection model.

3.4.3.2. Edge Impulse model building

As the image classifier section mentioned, Edge Impulse introduced the EON Tuner feature at the end of July 2021 for audio classification models. To recap EON Tuner automates the search for finding the optimum parameters for the machine learning architecture.

Before running EON Tuner, the user must choose three options:

1. **Dataset category** – The dropdown list has three choices: keyword spotting (e.g., yes/no), audible events (e.g., breaking glass), or continuous audio (e.g., water running). For the bat detection model, the audible events variable was chosen.
2. **Target device –** This dropdown list provides thirteen choices for different devices for device deployment. They range from Arduino Nanos to MacBook Pros. As a RPi was not a list option, the MacBook Pro variable was selected for this model.
3. **Time per inference (ms) –** This selection box allows the user to select the goal time they would like the model to make an inference. For the bat detection model, the standard choice of 100ms was chosen.

Once these variables are selected, the user clicks the "Start EON tuner" button, and the tuner begins to search for the optimum architecture. Edge Impulse warns that this process can take up to six hours but running it for this model took approximately thirty minutes. Figure 3.21 below shows the EON Tuner page after the tuner has been run.

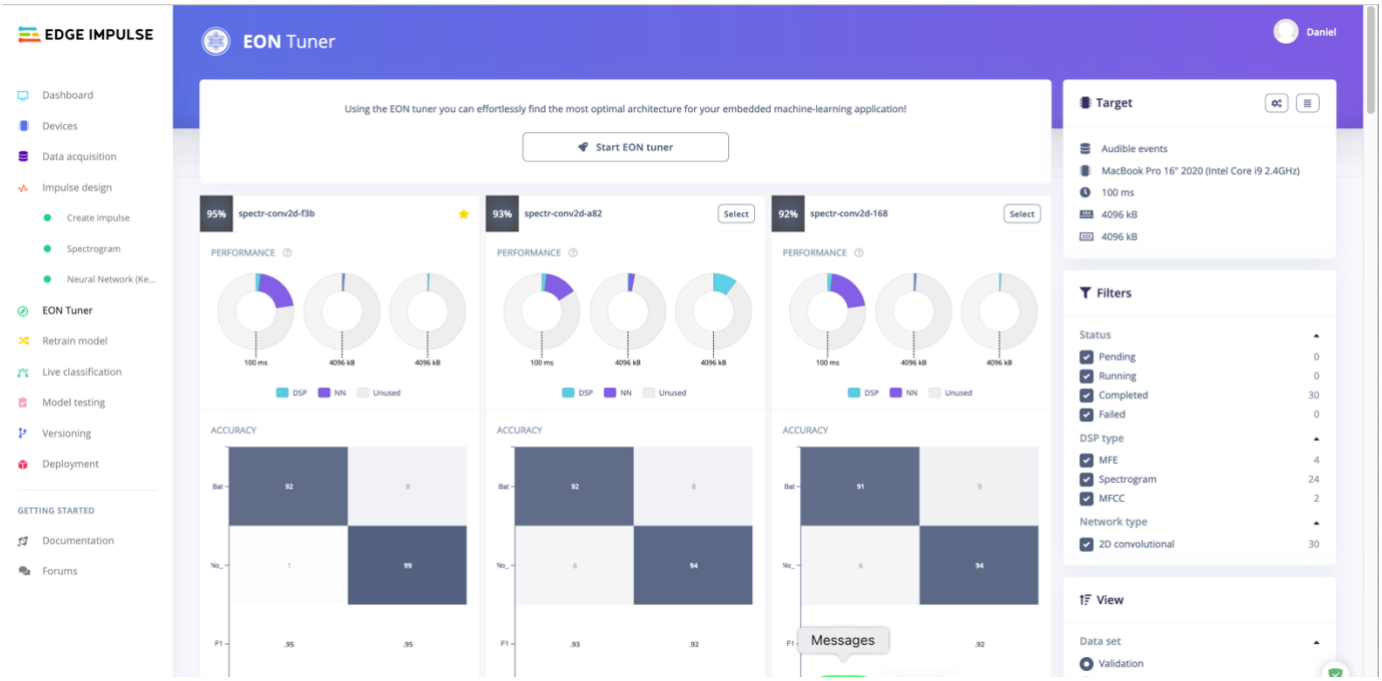


Figure 3.21: Screenshot of the Edge Impulse EON Tuner output

Thirty different model architectures were tested for the audio classification model, and then the models were then sorted by their accuracy on the validation dataset. The top three are shown above in figure 3.21. Figure 3.22 shows the optimum model architecture chosen for the bat detection device.

Figure 3.22: Screenshot of the optimum model architecture as found by Edge Impulse's EON Tuner.

The image on the left of figure 3.22 highlights three performance indicators for the model's speed as per the target device chosen. The leftmost circle highlights the time per inference, broken down into time spent on the digital signal processing (DSP), neural network (NN) and then the time left unused. The inner circle shows the target RAM used, and the right-hand

circle shows the target ROM. In this example, the model falls well within the goals for all three indicators.

Meanwhile, the image in the middle of figure 3.22 is a confusion matrix for the model architecture. This model has an F1-score of 0.95 for both detecting bats and the absence of bats with the validation data. On test data, the model had an F1-score of 0.94 for detecting bats and an F1-score of 0.96 for detecting the absence of bats. Thus, it was 95% accurate overall on both the validation and test datasets.

The image on the right-hand side provides the model architecture information. The input section shows how the audio clips will be broken down into inspection windows. For this architecture, it will take a 1000ms window and break it down into four 250ms windows.

The spectrogram section highlights how Edge Impulse will generate the spectrogram to be analysed. It shows that this model will utilise a frame length, the length of time between frames, of 0.025 seconds, a frame stride, the step between successive frames, of 0.0124 seconds, and a noise floor, everything quieter than this will be dropped, of -52dB (*Edge Impulse*, no date).

Lastly, the Keras section breaks down the model architecture into its layers. This model has four 2D convolutional layers and finishes with one dropout layer. The specific build of the layers can be seen in figure 3.22.

Once the EON Tuner finishes, an architecture can be chosen by simply clicking the select button. After the model has been selected, it transfers to the Impulse and can be deployed to a device.

3.4.3.3. Prototype deployment and testing

Deploying the audio classification model to the RPi is much simpler than deploying than image classifier. Edge Impulse provides specific instructions on how to connect to a RPi device.

Edge Impulse provides a specific software development kit (SDK) for Linux and Python, which allows the model to be easily run on the RPi (*Linux Python SDK*, no date). In addition, Edge Impulse also provides SDKs for Node.js, Go and C++. The code for the Python SDK can be found at the Edge Impulse GitHub (*Edge Impulse Linux SDK for Python*, 2021). When using the Python SDK, Edge Impulse generates a file called modelfile.eim, which can then be called upon using a Python script that Edge Impulse provides (*Edge Impulse Linux SDK for Python*, 2021).

Figure 3.23 below shows the audio detection model running on a RPi device using the Python SDK using a Dodotronic Ultramic UM192K.

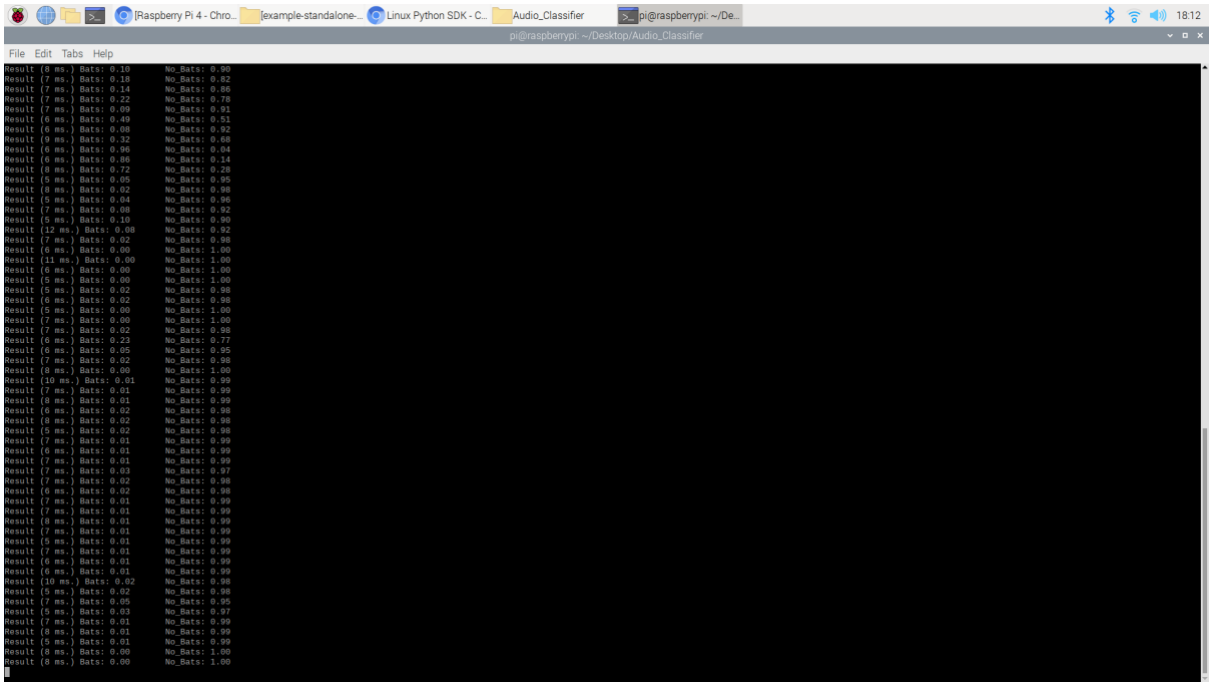


Figure 3.23: Screenshot of the audio detection model running on a Raspberry Pi 4 using a Dodotronic Ultramic UM192K.

3.5. Statement of Ethics

Data captured during this project cannot identify individuals. Nor does the data used to build the machine learning models identify individuals. For this reason, there was no ethics review required, and the project was deemed to be of minimal risk.

4. Experiments and results

Due to how the models were created, they had to be experimented on in slightly different ways. Therefore, this section will be split between the two bat detection options like the previous section.

4.1. Option 1: SoX generated spectrograms and an Edge Impulse image classification model

To test the model, an audio file was recorded in Wanstead Flats on 24 June 2021 at approximately 22:00. The length of the file is 23 minutes and 40 seconds. The audio file was recorded using the Dodotronic Ultramic UM192K while connected to MacBook Air using the open-source audio software Audacity (Audacity, no date). Audacity allows its users to create and watch spectrograms in real-time, and through the recording process, it was clear that bats were in the area and being recorded, as seen in figure 4.1 below to the right of the black line.

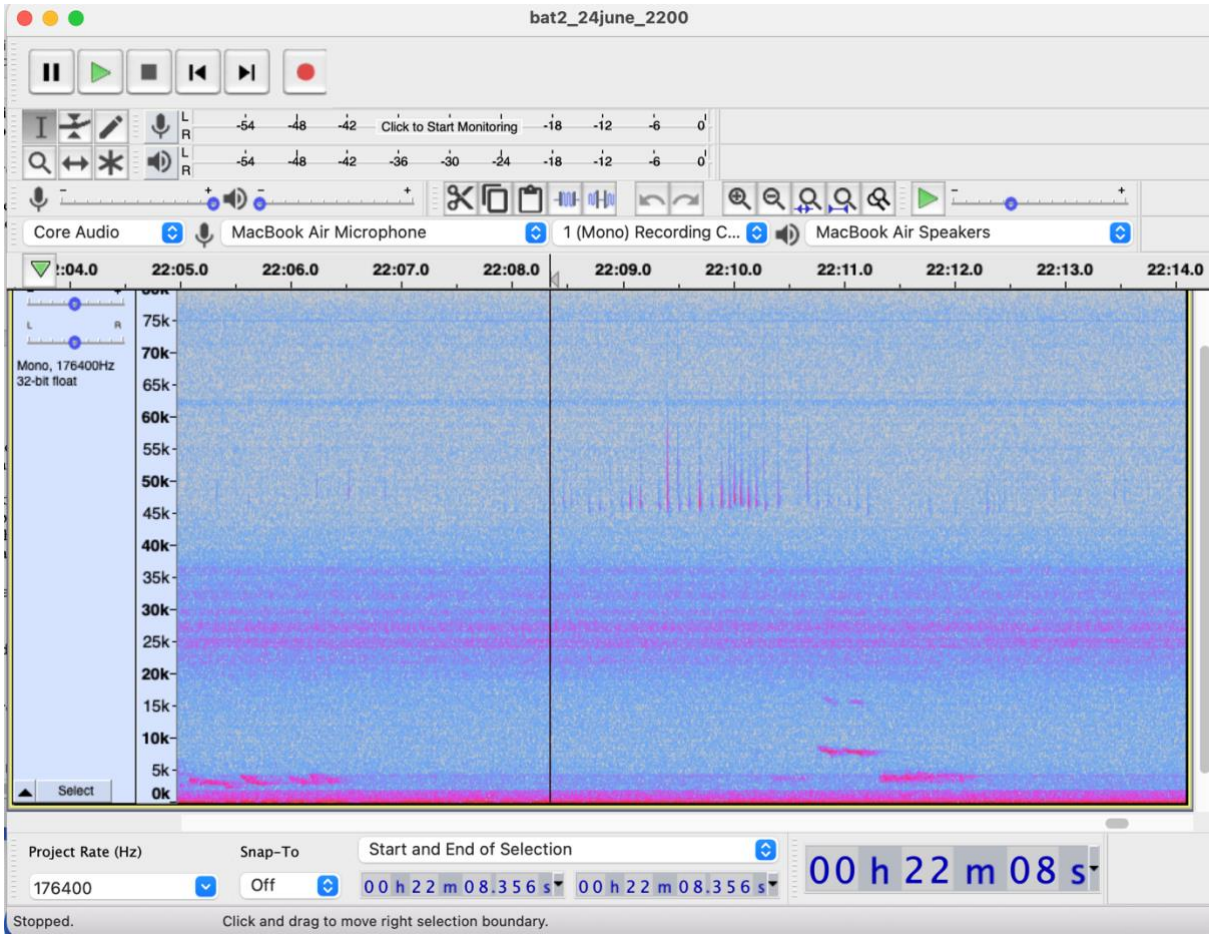
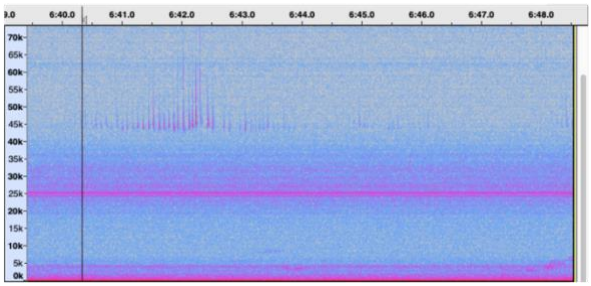


Figure 4.1: Screenshot of the audio recording taken on 24 June using the Dodotronic Ultramic and Audacity highlighting the evidence of bat calls (Audacity, no date).

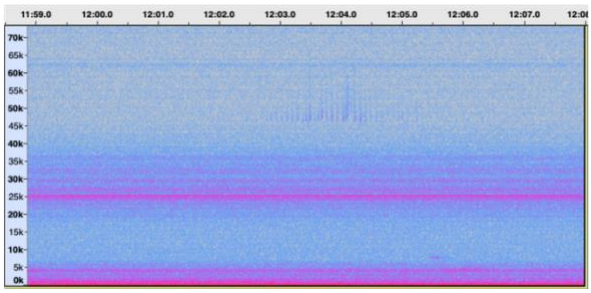
Once recorded, the audio clip was then processed using the image classification model, and Python script explained in the methodology. Processing the 24-minute audio file took approximately nine and a half minutes when done on a MacBook Air with a 1.2 GHz Quad-core Intel core i7 processor.

The outputted result files can be seen in Appendix C, and the beginning of the positive bat calls file can be seen in figure 4.2. Comparing the positive bat detection timestamps with the spectrograms created in Audacity reveals that the bat detection model works well against the new and unseen recorded data. Figure 4.2 below highlights the positive bat detection calls with screenshots in Audacity.

	time	time	filename	bats_present	confidence
400	400	06:40	bat2_24june	TRUE	0.922164
401	401	06:41	bat2_24june	TRUE	1
402	402	06:42	bat2_24june	TRUE	1
403	403	06:43	bat2_24june	TRUE	0.908316
405	405	06:45	bat2_24june	TRUE	0.895165
408	408	06:48	bat2_24june	TRUE	0.999899



	time	time	filename	bats_present	confidence
565	565	09:25	bat2_24june	TRUE	0.993615
723	723	12:03	bat2_24june	TRUE	0.615933
766	766	12:46	bat2_24june	TRUE	0.758321



	time	time	filename	bats_present	confidence
1326	1326	22:06	bat2_24june	TRUE	0.971068
1328	1328	22:08	bat2_24june	TRUE	0.959925
1329	1329	22:09	bat2_24june	TRUE	1
1330	1330	22:10	bat2_24june	TRUE	0.648567
1334	1334	22:14	bat2_24june	TRUE	0.705198
1335	1335	22:15	bat2_24june	TRUE	0.98976
1336	1336	22:16	bat2_24june	TRUE	0.983293

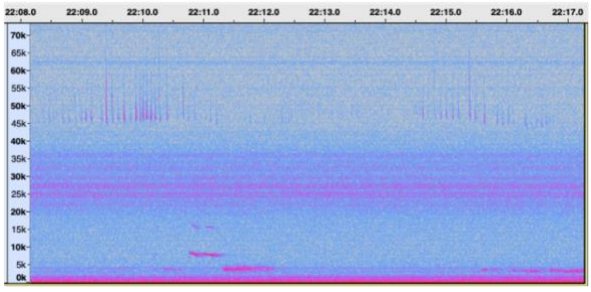
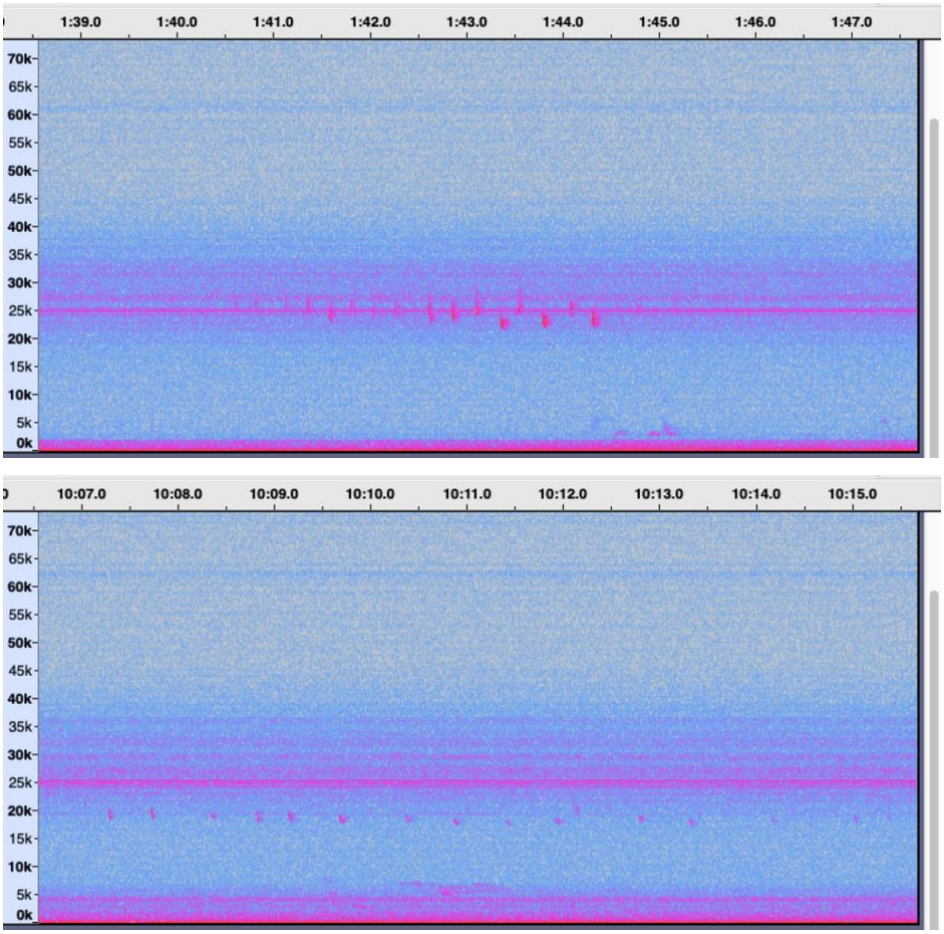


Figure 4.2: Three screenshots of the positive bat detection .csv file with time stamps and the corresponding spectrograms from Audacity.

Figure 4.2 shows that the model is very accurate at recognising the bat calls on the new data at frequencies above 40 kHz. The bottom image also highlights the absence of bat calls as it recognises that there are no bat calls between 22:10 and 22:14. The model's accuracy

continues throughout the twenty-four minutes of audio, and the complete .csv files can be found in Appendix C with a link to GitHub.

However, one issue to note is that despite a plan to recognise bat calls at lower frequencies, it appears that some potential bat calls may be missed when they do not rise above 25 kHz. There are three occurrences of what might be bat calls below 25 kHz in the recording. The three occurrences are highlighted below in figure 4.3. There was no detection of bats at any time in the first two images in figure 4.3. However, bat calls were detected in the third image, between 8:07 and 8:15, likely because of the bat calls at the higher frequency. However, it is also possible that the model correctly ignored these calls as they could potentially be bird calls and not bat calls. In future research, this distinction would have to be made by an ecologist. Again, the complete .csv files can be found in Appendix C in a link to GitHub.



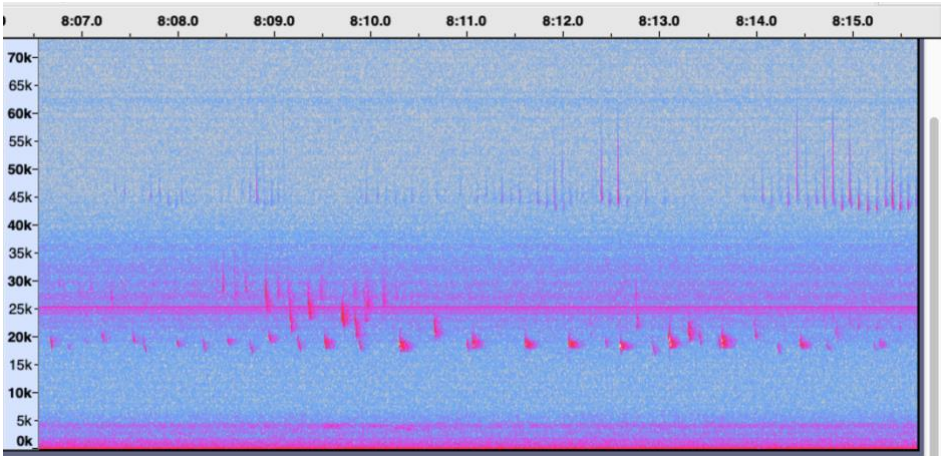


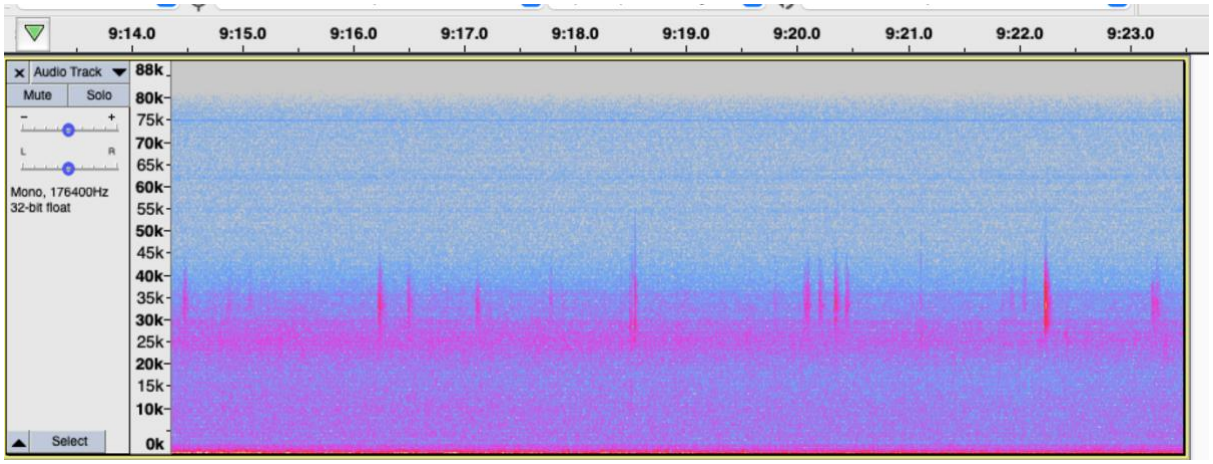
Figure 4.3: Screenshots of the potential bat calls missed or bird calls rightfully ignored by the image detection model at frequencies below 25 kHz.

4.2. Option 2: Edge Impulse audio classification model

The audio classification model proved more challenging to test with live data. While it was quickly deployed to the RPi device and Dodotronic Ultramic and the model registered predictions, the presence of bat calls could not be easily verified because spectrograms were not created automatically. A workaround was done by simultaneously recording audio with Audacity and through the Edge Impulse runner in the terminal as well. Both outputs were then compared to see if bat calls apparent in Audacity were identified by the model running in parallel in the terminal.

On 15 August at approximately 21:30, the technique deployed above was used, and a 10-minute recording was made in Wanstead Flats. The first step to compare Audacity and the Edge Impulse runner model was to export the model results from the terminal and into Excel. Once this was completed, an approximate comparison of the timing was made as the Edge Impulse runner records time by the time it takes to classify a recording in milliseconds. This comparison was completed by grouping all the Edge Impulse results into four bat detection timestamps that reflect one-second passing on the Audacity recording. These four grouped bat call detection ratings were then averaged, and those results were compared to the Audacity recording spectrogram. The apparent weakness of this method is that the timings will not be exactly like for like between the model and the Audacity recordings, so there will be some flexibility when comparing the two.

The detection model appears to be highly sensitive upon the first review as the predictions change and frequently move from moment to moment. However, this could be because the evening was very windy at the recording time, and there was significant background noise from the trees themselves. This is easily seen at the noise levels in the lower frequencies on the Audacity recording. Figure 4.4 below shows a less noisy recording section and compares the Audacity spectrogram to the model's outputs.



	A	B	C	D
1	Grouped Seconds	Approximate Time in Seconds	Approxmate One Second Average Bats	Approxmate One Second Average No Bats
553	552	09:12	0.75095	0.24905
554	553	09:13	0.6909	0.3091
555	554	09:14	0.8933	0.1067
556	555	09:15	0.793325	0.206675
557	556	09:16	0.999825	0.000175
558	557	09:17	0.76935	0.23065
559	558	09:18	0.834125	0.165875
560	559	09:19	0.3731	0.6269
561	560	09:20	0.82435	0.17565
562	561	09:21	0.80695	0.19305
563	562	09:22	0.9488	0.0512
564	563	09:23	0.444675	0.555325
565	564	09:24	0.9378	0.0622
566	565	09:25	0.4605	0.5395
567	566	09:26	0.9361	0.0639
568	567	09:27	0.64435	0.35565
569	568	09:28	0.866725	0.133275
570	569	09:29	0.850475	0.149525
571	570	09:30	0.6887	0.3113

Figure 4.4: Screenshot comparison of an Audacity spectrogram and the Edge Impulse bat detection audio model.

Unlike the recording on 24 June, it does not appear that bat calls were recorded this day. However, sounds like bat calls were found at a lower frequency than the 24 June recording, between 30 kHz and 45 kHz. These appear in the spectrogram and are registered by the model as bat calls, as shown in the bottom part of figure 4.4.

It was determined that these were not bat calls as they appear too regularly through the recording, and when the audio is slowed down, the sound is significantly different from those of verified bat calls.

The noises most confidently picked up as bat calls by the model appear at approximately 09:16, 09:18, and 09:22, all of which the model detects as reflected by column C. However, as reflected by column C's varying values, the model is sensitive and moves around widely. As a result, this model appears susceptible to background noise and would likely benefit from fine-tuning to help alleviate some of the variances and false positives.

5. Discussion

5.1. Could a citizen scientist or non-computer scientist do this, how could it be used, and how would a vast deployment look?

A non-computer scientist would create a bat detector like this, but there would be many growing pains. Edge Impulse makes the machine learning portion for tiny devices very accessible to someone from a non-computer science background; however, many things around the edges of building a device may cause a more layman type person to struggle.

This was most evident when attempting to repurpose the image detection model to process the features of the spectrogram instead of input coming in through a camera. This automation of audio information to the features being assessed by the model required, at a minimum, comfort with using terminal, bash, and Python. While many people outside of computer science have these skills, not every person interested in detecting bats would use Edge Impulse and create a model and working device. However, using Edge Impulse does demystify some of the more black-box elements of machine learning that can be so intimidating with things like Keras and TensorFlow.

Broad deployment of tiny devices detecting bats would provide researchers with a level of data that they would never have had access to in the past. However, a deployment could not simply be broad; it would also have to be unified to be useful for researchers. For example, if thousands of slightly different models were detecting bats in somewhat different ways, then the research would not be reproducible and could not be easily compared. A uniform deployment gathering data in real-time would allow researchers to monitor the effects of events like wildfires, floods and large construction projects on bat activities and populations across different locations.

However, deployment at a small individual scale would allow citizen scientists to provide anecdotal evidence of changes in their locations. If a critical mass of citizen scientists were finding similar changes, for example, decreasing populations, in their bat research, this would end up being something more professional research bodies could not ignore.

5.2. Ethical considerations and privacy concerns

Ethical considerations and privacy concerns are not unique problems to bat detectors. Concerns that devices like Siri or Alexa are listening and recording when they should not have been around for years (*Apple contractors 'regularly hear confidential details' on Siri recordings*, 2019). However, for bat detectors going up in places like the Queen Elizabeth Olympic Park, as in the case of the Shazam for Bats devices, concerns around what these devices might be hearing and who could potentially access the recordings have to be considered.

In a very extreme example, as the Shazam for Bats network is connected to the cloud, it is potentially vulnerable to being hacked. This means that people could access what is said at different points throughout the park, even though the sensors are simply meant to detect bats. In the wrong hands, a sensor network like this could end up being something more nefarious (Chan and Perrig, 2003; Kröger, 2019). Therefore, researchers must be prudent in how, why, and where they set up sensor networks to avoid incidents from occurring.

5.3. Learnings from the journey, constraints, and future improvements

To be fit for more rigorous research purposes, both the image and audio detection models would require some improvements or tweaks to be suitable. For example, as seen in the results of the image detection model, it is potentially missing bat calls made at lower frequencies. For this to be fixed in the future, a logical step would be to validate the training data with a bat expert to ensure that all bat calls are present. Once this is done, the spectrograms in SoX can be tweaked to reflect all bat calls, ideally and if possible, still fitting the 160x160 optimum size in Edge Impulse. However, if the calls at the lower frequencies are birds and not bats, then the model is already performing as intended.

Meanwhile, the audio detection model may benefit from a decrease in its sensitivity. While this model is very accurate in testing, it appears to include many false positives, as can be seen in the classification of the audio recorded on 15 August. If researchers use the audio detection model, understanding how comfortable they are with over-classification would be required. For some studies, over classifying by 20% might be fine and could simply be adjusted for after the fact; however, other studies might require a human in the loop to validate the results and correct each error on its own to ensure accuracy.

Another clear future improvement would be to utilise a constantly running device, classifying in real-time and translating the model's results into meaningful data in a similar vein to the Nature-Smart Cities original work (*Nature-Smart Cities*, no date). A second improvement would be able to classify separate bat species; however, the data provided for this paper did not include the species tags, so this type of classification could not be done. Nevertheless, this would be an excellent addition and would likely stretch Edge Impulse's abilities as a future use case.

Referring to the research question, the constraints appear to be tied back to research requirements and the data used. Edge Impulse provides a robust, flexible, and powerful tool for machine learning which can provide highly accurate results. However, a researcher must know the good-enough threshold for their work and aim to achieve that. This comfort level or good-enough threshold might be 95% accuracy alongside the knowledge that the model likely overfits by 15% for some researchers. For others, however, slightly lower accuracy and underfitting might be preferred if they are not overstating the presence of bats. In the end, like previous more sophisticated models, Edge Impulse relies on a clear goal for the project so that the correct type of data can be used to train the model and subsequently make predictions. It is clear, though, that Edge Impulse is a tool that can easily rival more sophisticated models on suitable use cases and in the right hands.

6. Conclusion

This research set out to create bat detection prototypes using a combination of cost-effective hardware and simple to use, open-source software. Through prototyping, it was hoped that an understanding of the constraints of these lower-cost types of devices could be learnt. Once built, the accuracy and constraints of these devices could then be compared to the more sophisticated examples created by computer scientists. In the end, two prototypes were developed built on data gathered by the Shazam for Bats and Nature-Smart cities work (Gallacher *et al.*, no date; *Nature-Smart Cities*, no date, p.)

The first prototype developed used SoX to create spectrograms of audio files, which then could be analysed by an image detection model made by Edge Impulse. This prototype was 99.49% accurate on the test data surpassing the 95% accuracy seen in the Shazam for Bats research (Gallacher *et al.*, no date). However, when tested on new data, it showed that the model might miss bat calls below 25 kHz.

The second audio detection model also analysed spectrograms, but the spectrograms were directly created in the Edge Impulse built model. Thus, the model did not require external processing software like SoX, decreasing its processing time. This model was 95% accurate on test data matching the Shazam for Bat research's accuracy, but it appeared to generate a high number of false positives when tested on novel data (Gallacher *et al.*, no date).

Both devices utilised relatively inexpensive hardware in a Raspberry Pi and a Dodotronic Ultramic microphone alongside entirely open-source software. This combination of hardware and software proved that less expensive, simpler alternatives to the bat detection devices made by computer scientists could be made by non-computer scientists. However, if either prototype were to be used for ecological research in the future, a clear understanding of the research requirements would have to be understood. This would allow model parameters to be adjusted to reflect the sensitivity and accuracy required and accommodate preferred over or underfitting levels.

Undoubtedly further proliferation of bat detection devices would benefit bat research and the bat conservation effort. With more data being gathered, there is an ever-increasing chance of a breakthrough being made. However, further research on this topic by non-computer science

people should not allow good to become the enemy of great. Future researchers should determine a good-enough threshold for each scenario and just build to surpass that as a starting point. By setting out on the journey, discoveries will be made, and the acceptable threshold may move accordingly. Regardless though, creating a perfect bat detection device will not happen on the first attempt. However, the device created likely will surpass the ones used before, which is the all-important first step.

Bibliography

Aodha, O. M. *et al.* (2018) 'Bat detective—Deep learning tools for bat acoustic signal detection', *PLOS Computational Biology*, 14(3), p. e1005995. doi: 10.1371/journal.pcbi.1005995.

Apple contractors 'regularly hear confidential details' on Siri recordings (2019) *the Guardian*. Available at: http://www.theguardian.com/technology/2019/jul/26/apple-contractors-regularly-hear-confidential-details-on-siri-recordings (Accessed: 16 August 2021).

Audacity (no date) *Home, Audacity ®*. Available at: https://www.audacityteam.org (Accessed: 12 August 2021).

'Audio & Memory Cards: How much record time do I have?' (2011) *Sound & Picture*, 22 June. Available at: https://www.soundandpicture.com/2011/06/audio-memory-cards-how-much-record-time-do-i-have/ (Accessed: 8 August 2021).

Barlow, K. E. *et al.* (2015) 'Citizen science reveals trends in bat populations: The National Bat Monitoring Programme in Great Britain', *Biological Conservation*, 182, pp. 14–26. doi: 10.1016/j.biocon.2014.11.022.

Bat Conservation Trust (no date a) *About us - The Trust, Bat Conservation Trust*. Available at: https://www.bats.org.uk/the-trust/about-us (Accessed: 5 August 2021).

Bat Conservation Trust (no date b) *Why bats matter - About Bats, Bat Conservation Trust*. Available at: https://www.bats.org.uk/about-bats/why-bats-matter (Accessed: 5 August 2021).

Bat Survey & Monitoring | Wildlife Survey & Monitoring | NHBS (no date). Available at: https://www.nhbs.com/1?slug=bat-survey&q=&fR[hide][0]=false&fR[live][0]=true&fR[shops.id][0]=1&fR[subsidiaries][0]=1&hFR[subjects_equipment.lvl1][0]=Bat%20Survey%20%26%20Monitoring (Accessed: 7 August 2021).

Bier, J. (2020) 'EETimes - AI and Vision at the Edge', *EETimes*, 6 September. Available at: https://www.eetimes.com/ai-and-vision-at-the-edge/ (Accessed: 8 August 2021).

Chan, H. and Perrig, A. (2003) 'Security and privacy in sensor networks', *Computer*, 36(10), pp. 103–105. doi: 10.1109/MC.2003.1236475.

Continuous audio sampling (no date) *Edge Impulse Docs*. Available at: https://docs.edgeimpulse.com/docs/continuous-audio-sampling (Accessed: 6 August 2021).

Department for Environment, Food and Rural Affairs, UK (2020) 'UK Biodiversity Indicators 2020', p. 60.

Dickinson, J. L., Zuckerberg, B. and Bonter, D. N. (2010) 'Citizen Science as an Ecological Research Tool: Challenges and Benefits', *Annual Review of Ecology, Evolution, and Systematics*, 41(1), pp. 149–172. doi: 10.1146/annurev-ecolsys-102209-144636.

Dixon, M. D. (2012) ‘Relationship between land cover and insectivorous bat activity in an urban landscape’, *Urban Ecosystems*, 15(3), pp. 683–695. doi: 10.1007/s11252-011-0219-y.

echopibox/ultramic2spectrogram.py at main · djdunc/echopibox (no date) *GitHub*. Available at: https://github.com/djdunc/echopibox (Accessed: 8 August 2021).

Edge Impulse (no date). Available at: https://www.edgeimpulse.com (Accessed: 5 April 2021).

Edge Impulse Linux SDK for Python (2021). Edge Impulse. Available at: https://github.com/edgeimpulse/linux-sdk-python (Accessed: 12 August 2021).

Frequencies | Staffordshire Bat Group (no date) *Staffsbats*. Available at: https://www.staffsbats.co.uk/frequencies (Accessed: 7 August 2021).

Gallacher, S. *et al.* (no date) ‘Shazam For Bats: Internet of Things for Continuous Real- Time Biodiversity Monitoring’, p. 17.

Gonfalonieri, A. (2018) *How Amazon Alexa works? Your guide to Natural Language Processing (AI), Medium*. Available at: https://towardsdatascience.com/how-amazon-alexa-works-your-guide-to-natural-language-processing-ai-7506004709d3 (Accessed: 6 August 2021).

Jones, G. *et al.* (2009) ‘Carpe noctem: the importance of bats as bioindicators’, *Endangered Species Research*, 8, pp. 93–115. doi: 10.3354/esr00182.

Keras: the Python deep learning API (no date). Available at: https://keras.io/ (Accessed: 20 August 2021).

Kröger, J. (2019) ‘Unexpected Inferences from Sensor Data: A Hidden Privacy Threat in the Internet of Things’, in Strous, L. and Cerf, V. G. (eds) *Internet of Things. Information Processing in an Increasingly Connected World*. Cham: Springer International Publishing (IFIP Advances in Information and Communication Technology), pp. 147–159. doi: 10.1007/978-3-030-15651-0_13.

Linux Python SDK (no date) *Edge Impulse Docs*. Available at: https://docs.edgeimpulse.com/docs/linux-python-sdk (Accessed: 12 August 2021).

Miller, P. (2018) *What is edge computing?, The Verge*. Available at: https://www.theverge.com/circuitbreaker/2018/5/7/17327584/edge-computing-cloud-google-microsoft-apple-amazon (Accessed: 6 August 2021).

Muda, L., Begam, M. and Elamvazuthi, I. (2010) ‘Voice Recognition Algorithms using Mel Frequency Cepstral Coefficient (MFCC) and Dynamic Time Warping (DTW) Techniques’, 2(3), p. 6.

Nature-Smart Cities (no date) *Nature-Smart Cities*. Available at: https://naturesmartcities.com/ (Accessed: 7 August 2021).

Park, K. J. (2015) ‘Mitigating the impacts of agriculture on biodiversity: bats and their potential role as bioindicators’, *Mammalian Biology*, 80(3), pp. 191–204. doi: 10.1016/j.mambio.2014.10.004.

Parker, K. A. *et al.* (2019) ‘Rapid Increases in Bat Activity and Diversity after Wetland Construction in an Urban Ecosystem’, *Wetlands*, 39(4), pp. 717–727. doi: 10.1007/s13157-018-1115-5.

Pelicella, I. (no date) *Ultramic UM192K, Dodotronic*. Available at: https://www.dodotronic.com/product/ultramic-um192k/ (Accessed: 8 August 2021).

‘Purchase – SonoBat’ (no date). Available at: https://sonobat.com/purchase/ (Accessed: 6 August 2021).

Raspberry Pi 4 Model B (no date) *The Pi Hut*. Available at: https://thepihut.com/products/raspberry-pi-4-model-b (Accessed: 8 August 2021).

Raspberry Pi 4 Model B Starter Kit (no date) *The Pi Hut*. Available at: https://thepihut.com/products/raspberry-pi-starter-kit (Accessed: 8 August 2021).

Raspberry Pi Bat Project (no date). Available at: http://www.bat-pi.eu/EN/index-EN.html (Accessed: 14 June 2021).

Running your impulse locally (no date) *Edge Impulse Docs*. Available at: https://docs.edgeimpulse.com/docs/running-your-impulse-locally-1 (Accessed: 4 August 2021).

Russo, D. and Ancillotto, L. (2015) ‘Sensitivity of bats to urbanization: a review’, *Mammalian Biology*, 80(3), pp. 205–212. doi: 10.1016/j.mambio.2014.10.003.

SanDisk Ultra 64 GB microSDXC Memory Card + SD Adapter with A1 App Performance Up to 100 MB/s, Class 10, U1 - SanDisk (no date). Available at: https://www.amazon.co.uk/SanDisk-microSDXC-Memory-Adapter-Performance/dp/B073JYVKNX (Accessed: 8 August 2021).

Satyanarayanan, M. (2017) ‘The Emergence of Edge Computing’, *Computer*, 50(1), pp. 30–39. doi: 10.1109/MC.2017.9.

SonoBat – Software for Bat Call Analysis (no date). Available at: https://sonobat.com/ (Accessed: 21 July 2021).

SoX - Sound eXchange | HomePage (no date). Available at: http://sox.sourceforge.net/Main/HomePage (Accessed: 8 August 2021).

SoX Main Manual (no date). Available at: http://sox.sourceforge.net/sox.html (Accessed: 8 August 2021).

TensorFlow (no date). Available at: https://www.tensorflow.org/ (Accessed: 20 August 2021).

The Raspberry Pi Foundation (no date) *Raspberry Pi 4 Model B specifications*, *Raspberry Pi*. Available at: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/ (Accessed: 8 August 2021).

Trust, B. C. (no date a) *Bat groups - Support Bats, Bat Conservation Trust*. Available at: https://www.bats.org.uk/support-bats/bat-groups (Accessed: 18 August 2021).

Trust, B. C. (no date b) *National Bat Monitoring Programme - Our Work, Bat Conservation Trust*. Available at: https://www.bats.org.uk/our-work/national-bat-monitoring-programme (Accessed: 18 August 2021).

Twmffat, T. (no date) *Intelligent Bat Detector*, *Instructables*. Available at: https://www.instructables.com/Intelligent-Bat-Detector/ (Accessed: 18 August 2021).

Walters, C. L. *et al.* (2012) 'A continental-scale tool for acoustic identification of European bats', *Journal of Applied Ecology*. Edited by J. Minderman, 49(5), pp. 1064–1074. doi: 10.1111/j.1365-2664.2012.02182.x.

Warden, P. and Situnayake, D. (2019) *TinyML*. Available at: https://learning.oreilly.com/library/view/tinyml/9781492052036/ (Accessed: 6 August 2021).

What is a Color Hex Code? - Definition from Techopedia (no date) *Techopedia.com*. Available at: http://www.techopedia.com/definition/29788/color-hex-code (Accessed: 12 August 2021).

'What is a Raspberry Pi?' (no date) *Raspberry Pi*. Available at: https://www.raspberrypi.org/help/what-is-a-raspberry-pi/ (Accessed: 8 August 2021).

Appendix A – Test Results

GitHub link - https://github.com/darennie/dissertation/tree/main/Appendix%20A%20-%20Testing%20Results

Appendix B – Bat Detection Script

GitHub link - https://github.com/darennie/dissertation/tree/main/Appendix%20B%20-%20Image%20Detection%20Model%20Script

Appendix C – Outputted Image Detection Results

GitHub link - https://github.com/darennie/dissertation/tree/main/Appendix%20C%20-%20Outputted%20Image%20Detection%20Files

Appendix D – Edge Impulse Model Links

Appendix D.1 – SoX generated spectrograms and an Edge Impulse image classification model

Edge Impulse link - https://studio.edgeimpulse.com/public/43716/latest

Appendix D.2 – Edge Impulse audio classification model

Edge Impulse link - https://studio.edgeimpulse.com/public/39732/latest

Appendix E – Full Audio Recordings and Processed Output Files

Appendix E.1 – 24 June Raw Recording

Google Drive link - https://drive.google.com/file/d/1Wj-OXxrH3q3n4xbWQt3OApuqIP2CVCC_/view?usp=sharing

Appendix E.2 – 24 June Recording Broken Down by Python Script

Google Drive link - https://drive.google.com/drive/folders/1tkHlMnbiK8br0bdHn6shHFt2xG6-8yQf?usp=sharing

Appendix E.3 – 24 June Broken Down .png files

Google Drive link - https://drive.google.com/drive/folders/1TIJrnmveR_6ly5QsJlH0wfttJmwxIMqY?usp=sharing

Appendix E.4 – 15 August Raw Recording

Google Drive link - https://drive.google.com/file/d/1aWkwarE1YKxunX08Gpu5wKRMGSRxanuP/view?usp=sharing

Appendix E.5 – 15 August Raw Output File

GitHub Link – https://github.com/darennie/dissertation/tree/main/Appendix%20E%20%E2%80%93%20Full%20Audio%20Recordings%20and%20Processed%20Output%20Files

Appendix E.6 – 15 August Processed Output File

GitHub Link – https://github.com/darennie/dissertation/tree/main/Appendix%20E%20%E2%80%93%20Full%20Audio%20Recordings%20and%20Processed%20Output%20Files