ChirpAlert: A Low-Power Acoustic Sensor Platform with On-Device Signature Recognition and LoRa Alerts

Joris Gravesteijn

Delft University of Technology

TU Delft

ChirpAlert: A Low-Power Acoustic Sensor Platform with On-Device Signature Recognition and LoRa Alerts

by

Joris Gravesteijn

Supervisor:	Dr. H.J.C. Kroep
Graduation Committee:	
Dr. R.R. Venkatesha Prasad	Delft University of Technology
Dr. C. Lofi	Delft University of Technology
Dr. H.J.C. Kroep	Delft University of Technology
Faculty:	Electrical Engineering, Mathematics and Computer Science
Research Group:	Networked Systems

Cover: Generated with ChatGPT 5 and modified
Style: TU Delft Report Style, with modifications by Daan Zwaneveld

TUDelft

Abstract

Modern ecological monitoring needs long-term, non-invasive observation of wildlife in remote areas, but traditional methods either store raw audio for offline analysis or depend on power-hungry, networked infrastructure. This creates three problems: high energy cost, high data volume and no real-time awareness in the field.

This thesis presents ChirpAlert, a fully autonomous low-power acoustic sensor platform that records and classifies environmental sounds directly on embedded hardware. The system is built around a custom low-cost embedded platform node with an analog, ultrasonic capable microphone front-end, on-device machine learning inference, environmental sensing, GPS and long-range LoRa connectivity for transmitting live alerts. ChirpAlert runs an optimized audio event detection pipeline that converts raw audio into Mel-based time-frequency features and applies a quantized neural network for bird vocalization detection on the sensor itself.

A training and deployment framework is introduced that automatically assembles a task-specific dataset for a target species and region, performs hyperparameter and preprocessing optimization and converts the trained model to be capable of running on embedded hardware. The final platform operates at approximately 56 mW active power while running on-device inference, enabling long-term deployment.

Field tests confirm that ChirpAlert can recognize target bird sounds with 88.8% accuracy during deployment while correctly labeling 98.1% of the other sounds as not the target species, demonstrating the effectiveness of the proposed ChirpAlert hardware and audio event detection pipeline.

Acknowledgements

These past months have been one of the most exciting and educational periods of my academic career. Not only do I look back with pride, I'm also very happy with the final result and will continue working on ChirpAlert in the future. However, I could not have reached this point on my own. I would like to sincerely thank my supervisor Dr. Kees Kroep for taking me on last-minute after my first supervisor left the TU Delft and for his guidance during my thesis. During our weekly meetings, he supplied me with essential feedback that elevated this work to the quality that it currently has. Without his questions, help and experience in academic writing, this work would not have reached the quality it currently has. I would also like to thank Dr. Ranga Rao Venkatesha Prasad for giving me the opportunity and guidance to do this research.

Apart from my professor and supervisor, I would like to thank my girlfriend, family and friends for supporting me throughout my thesis.

Joris Gravesteijn
Delft, October 2025

Contents

Abstract **i**

Acknowledgements **ii**

1 Introduction **1**

2 Related Work **4**
2.1 Audio Event Detection 4
2.2 Related Hardware Solutions 5

3 Theory **7**
3.1 Audio Signals 8
3.2 Audio Pre-Processing 8
3.3 Audio Event Detection 11
3.4 Machine Learning for AED 13
3.5 On-Device AED 17

4 Methodology **20**
4.1 Breadboard Prototype 20
4.2 Training Datasets 22
4.3 Key Parameters 23
4.4 Generic Pipeline Structure 25
4.5 ModelZoo Experiments 25
4.6 A New Framework 32
4.7 On-Device optimizations 36

5 On-Device Implementation **41**
5.1 Sensor Design 41
5.2 Audio Event Detection Implementation 44
5.3 Spectrogram Verification 45
5.4 Power Characteristics 45

6 Deployment Results **48**

7 Future Work **51**

8 Discussion **52**

9 Conclusion **53**

References **54**

A ModelZoo Experiments Results **59**

B MiniRes Experiment 2 Confusion Matrices **64**

C Hyperparameter Space **66**

D Pica pica Dataset Species List **68**

Abbreviations

ADC Analog to Digital Converter.
AED Audio Event Detection.
AGC Adaptive Gain Control.

BCE Binary Cross Entropy.
BLE Bluetooth Low Energy.

CE Cross Entropy.
CNN Convolutional Neural Network.
CRNN Convolutional Recurrent Neural Network.

DCASE Detection and Classification of Acoustic Scenes and Events.
DRC Dynamic Range Compression.
DROP Delayed Reduce On Plateau.

FCE Focal Cross Entropy.
FFT Fast Fourier Transform.
FLOP Floating Point Operation.
FP32 32-bit Floating Point.
FPU Floating Point Unit.

GMM Gaussian Mixture Model.
GPS Global Positioning System.

HMM Hidden Markov Model.

IMU Inertial Measurement Unit.

LoRa Long-Range.

MAC Multiply-Accumulate.
MCU Micro-Controller Unit.
MFCC Mel-Frequency Cepstral Coefficient.
ML Machine Learning.

PCEN Per-Channel Energy Normalization.
PMIC Power Management Integrated Circuit.
PTQ Post-Training Quantization.

QAT Quantization-Aware Training.

RNN Recurrent Neural Network.
ROP Reduce On Plateau.
RP Recall at Precision.
RP90 Recall at 90% Precision.

SIMD Single Instruction Multiple Data.
STFT Short-Time Fourier Transform.
SVM Support Vector Machine.

WSN Wireless Sensor Network.

1 Introduction

As cities expand, habitats contract and climates shift. Protecting and understanding our ecosystem is one of the defining responsibilities and challenges of modern society. At the same time, technological progress offers us unprecedented possibilities to understand and support nature. We possess the capabilities to capture, process and interpret information from the world around us at a scale once unimaginable. Our new technologies give us the chance to gain a deeper understanding of nature that allows us to make informed and responsible decisions that will not only protect the ecosystem, but actively contribute to improve it and provide long-term stability for our future generations.

A crucial part of this effort is to observe the ecosystem without disturbing it. Imagine a biologist sitting quietly in a forest at dawn, hoping to track down a herd of animals. As the biologist is walking through the forest, he brushes against a bush which results in the animals running away. The act of the biologist trying to observe the ecosystem, alters the behavior being studied. This highlights the main challenge of non-invasive observation: How can we observe something without disturbing the natural flow? Traditional systems often utilize visual sensors such as cameras to observe the world around us. These tools have transformed the way we study wildlife and the environment in general. From camera traps hidden deep in forests to drones surveying remote landscapes, visual systems provide a powerful, unmanned window into nature. Not only do they allow researchers to document animal behavior and estimate population sizes, they can also track the presence of species over long periods of time without the need for human presence. By doing so, they have opened up entirely new possibilities for ecological research and conservation.

While these systems have proven invaluable, they also come with practical challenges. Cameras depend heavily on lighting conditions and can easily be obstructed by vegetation or terrain. In addition, they typically require a large amount of storage and significant computational resources to store and process the images and video, increasing both the energy consumption and infrastructure requirements. An alternative approach to environmental monitoring is to use audio-based methods. These offer a powerful way to observe nature without disturbing the ecosystem. As sound moves in all directions, a single microphone is capable of governing a wide area both during day and nighttime, even in cluttered environments without a direct line of sight, an area where visual systems often struggle. The advantages of acoustic monitoring methods extend beyond improved observational characteristics. Audio data also requires less storage and computation in order to extract the meaningful information, making it particularly suitable for remote and non-invasive deployments. Instead of processing a continuous video stream, an acoustic sensor only needs to process an audio waveform, which is significantly smaller and requires less computation. Not only does this greatly reduce the required infrastructure and energy requirements, it also allows the sensor itself to remain compact, preserving the non-invasive goal of wildlife observation.

One of the key challenges in acoustic wildlife monitoring is dealing with the complex nature of the soundscapes. Overlapping sounds, echoes and environmental noise are all hitting the microphone at the same time, making it difficult to isolate individual sounds and identify their sources. In the past,

this task relied heavily on human expertise. Researchers were required to listen to these recordings and annotate the sound sources. In recent years, advances in digital signal processing and machine learning have made it increasingly feasible to analyze these complex soundscapes autonomously. Algorithms can now distinguish between natural and artificial sounds, detect individual species and even distinguish between different animal behaviors based on sound. One of the key developments in the field of acoustic ecological monitoring is a deep learning model called BirdNET [1], released in 2021. Not only are recent versions of this model capable of detecting and classifying over 3,000 bird species with expert-level accuracy, recent versions are even capable of recognizing other kinds of animals, all while running on a smartphone.

Recent advances in embedded hardware have further increased the feasibility to deploy acoustic sensors fully autonomously. The rise of low-power, high-performance platforms and on-device AI has enabled complex sensing and processing tasks on the device itself. What once required powerful desktop computers or cloud computing, can now be executed locally in real-time. By processing the audio and performing the event detection on-device, only the relevant results, such as the presence of a species, needs to be transmitted. This drastically reduces data traffic and energy consumption while allowing the sensor to operate entirely independent of external infrastructure. These on-device capabilities are further supported by new long-range, low-energy communication technologies such as LoRa and highly efficient hardware designs allowing ultra low-power consumption. Together, these developments make it possible to build compact sensors, capable of running machine learning models directly on the device while operating for extended periods on small batteries or solar panels. During deployment, the detected events can be transmitted over large distances without the need for extensive and invasive infrastructure or frequent maintenance.

The combination of these technological advances paves the way for networks of intelligent acoustic sensors, capable of continuously monitoring wildlife and ecosystems while remaining fully autonomous, low-cost and highly efficient, all without disturbing the ecosystem itself. However, building an embedded system that is capable of reliably and autonomously recording and observing an ecosystem is far from trivial. Detecting and classifying audio events still involves computationally intensive steps starting with extracting useful information from the raw audio up to determining the animal that produced the sound. On small, low-power hardware, these operations must be able to work with the small amount of computational resources that they have available. As such, many of the current state-of-the-art machine learning models, capable of classifying wildlife sounds, are far too large and complex to fit on the limited resources of embedded devices. Furthermore, these systems have to run while only having limited power sources such as batteries or solar panels.

To explore these challenges, this research aims to develop an autonomous, energy-efficient and accurate acoustic monitoring system capable of recording and classifying wildlife and environmental sounds directly on embedded hardware. Bird vocalizations are chosen as the primary case study for this work, as they provide a well established domain for acoustic event detection. Birds produce rich and diverse vocal signals that are widely used as ecological indicators, making them an ideal benchmark for verifying the effectiveness and real-world performance of the proposed system.

The design of such a system revolves around three main goals:

1. **Efficiency** - Every stage should be optimized for the limited computational and memory available on embedded devices.
2. **Accuracy** - Despite the hardware constraints, the system must achieve reliable detection and classification performance.
3. **Autonomy** - The system must be able to operate for long periods without human intervention.

Balancing these three objectives is one of the core challenges of embedded acoustic monitoring and helps ensure a platform capable of listening to nature continuously and non-invasively. However, achieving all three of these goals simultaneously presents a complex design challenge. Improving one of these aspects often comes at the cost of another. Improving efficiency often comes at the cost of accuracy, while improving accuracy can quickly exceed the capabilities of the available resources. Balancing these trade-offs require careful design of the hardware, software and machine learning components of the system. This leads to the central research question of this system:

> "How to perform accurate and efficient bird detection using low-cost, resource-constrained embedded hardware?"

To answer this question, this work proposes the design of both the hardware and software components of an Audio Event Detection (AED) system. A custom sensor platform is designed with an environmental monitoring suite, including a microphone with ultrasonic capabilities and hardware capable of running Machine Learning (ML) models on-device. In order to verify the performance of the sensor platform and the accompanying AED pipeline, bird monitoring is selected as the AED task for the system. This is a well-established AED task and relevant use-case that serves as a benchmark for verifying the performance of the system in terms of accuracy, efficiency and autonomy. The system is designed to recognize one species at a time, reflecting a practical deployment scenario for targeted environmental monitoring. Apart from being able to run AED on-device, the platform also contains an enriched environmental sensor suite to make it a one-stop platform for environmental observation and allow for on-device combination of acoustic and environmental data for new research opportunities. The research also proposes an AED framework, capable of exploring different datasets as well as automatically building a dataset based on the deployment region and target species. The framework is also capable of training and converting a model to function on embedded methods. Within the framework, different preprocessing methods and parameter configurations are evaluated using hyperparameter optimization to determine which offer the best trade-off in terms of classification performance and resource requirements. Finally, a real-world field deployment is conducted to assess the end-to-end performance of the sensor platform and the full AED framework under real-world conditions.

This thesis contributes:

- **The design and implementation of the ChirpAlert platform:** A fully autonomous sensor platform capable of recording, processing and classifying environmental sounds on-device with ultrasonic and low-power capabilities. ChirpAlert contains a rich environmental sensor suite, including gas and light composition sensors, a LoRa radio and GPS.
- **The development of a complete on-device audio event detection pipeline:** A software framework capable of training, optimizing, evaluating and converting machine learning models for embedded deployment, including data and model hyperparameter optimization.
- **A method for automatic dataset generation:** A data collection pipeline that automatically builds a dataset based on the target species and geographic area of the deployment.
- **Foundations for scalable and extensible environmental monitoring:** The framework can be extended to multi-species classification or adapted to detect other types of environmental and ecological sounds, laying the foundations for future, large-scale, low-cost sensor networks.

This thesis is structured as follows: Chapter 2 reviews related works identified during the literature review. Chapter 3 presents the theory that this research builds on. Chapter 4 explains the methodology used to design the final system implementation. Chapter 5 describes the ChirpAlert system design, how the AED pipeline was implemented on-device and describes its resource and energy requirements. Chapter 6 describes the real-world test and the accompanying results. Chapter 7 outlines possible future work that can further improve the system and its usability. Chapter 8 discusses and reflects on the thesis. Lastly, chapter 9 concludes the research and is followed by a list of references and appendices that further support this research.

2

Related Work

2.1. Audio Event Detection

Over the past two decades, a significant amount of research has been done in the field of Audio Event Detection (AED). Early approaches relied on handcrafted features such as Mel-Frequency Cepstral Coefficients (MFCCs) combined with classifiers such as Hidden Markov Models (HMMs), Gaussian Mixture Models (GMMs) and Support Vector Machines (SVMs) in order to classify events inside audio [2, 3, 4, 5, 6, 7]. These systems typically split the audio signal into short frames and calculate features based on those windows. The classes are then modeled based on the characteristic statistical distribution over these features. While these approaches proved effective for controlled environments such as speech and meeting-room monitoring, performance degraded in complex, overlapping and noisy environments.

One of the key driving factors for audio event detection are the Detection and Classification of Acoustic Scenes and Events (DCASE) challenges which have helped standardize AED tasks and have driven most of the recent progress in AED. Not only does DCASE introduce public datasets and common evaluation metrics, it also resulted in numerous papers on the technical solutions of their top contenders, making the insight public for anyone to use [8, 9].

Modern Features

To address the limitations of handcrafted features, most modern AED systems operate on time-frequency representations of the audio signal. The current industry standard is to perform Short-Time Fourier Transforms (STFTs) on the audio, project them on Mel-scaled filter banks and log-compress the result to obtain a log-Mel spectrogram. This representation not only captures the temporal and frequency information of the signal, it also roughly aligns the spectrogram with the human auditory response using logarithmic compression. Large-scale audio classification work on data sources such as YouTube and AudioSet have demonstrated that log-Mel spectrograms are strong features for deep learning models in case of AED [10, 11, 12].

Even though log-Mel spectrograms are effective, they are sensitive to variations in absolute loudness and varying background noise, both of which are common in real-world environments. To improve the robustness of the AED system, Per-Channel Energy Normalization (PCEN) spectrograms were introduced as an adaptive alternative [13]. They aim to minimize the effects of varying loudness and stationary background noise by combining Dynamic Range Compression (DRC) with Adaptive Gain Control (AGC). Research has shown that this can improve the performance of the system significantly when compared with the standard log-Mel spectrogram [14, 15, 16], especially in noisy, far-field wildlife monitoring [17].

Deep Learning

The introduction of deep learning marked a major shift for AED. Model architectures like the Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) enabled systems that directly learned the discriminative patterns of an audio source from the spectrograms without needing

handcrafted intermediary features [7]. Supervised learning is the dominant form of training for modern AED systems. In this setup, models are trained to recognize audio classes by using audio data with accompanying class annotations such as a dog bark. Most current CNN and Convolutional Recurrent Neural Network (CRNN) architectures for audio event detection are trained in this fully supervised way using large labeled datasets such as AudioSet. Later on, recurrent layers were introduced to the CNN architecture to further capture the temporal components within the spectrogram, giving the rise to the now commonly used CRNNs architecture [7, 18, 19, 20].

Audio Based Bird Classification

BirdNET [1] is one of the key works in the AED event detection space. BirdNET is a deep neural network trained on a large amount of annotated bird vocalizations and recent versions are capable of identifying over 6,000 classes, including man made sounds and non-bird species such as insects and frogs. It demonstrates that large-scale supervised learning on expert-labeled bird audio can yield highly accurate species classification at scale.

Another key factor in the rapid developments in the bird AED space is played by the yearly BirdCLEF competitions which focus on automatic identification of bird vocalizations. During these challenges, a (weakly) supervised multi-class classification problem is given where systems must assign species labels to short segments, extracted from real-world long field recordings [21], with the 2025 iteration expanding this to include amphibians, mammals and insects as well [22]. Because BirdCLEF uses real soundscape data, it has pushed research toward robust classifiers that can handle the complex nature of overlapping sources, distant callers and habitat noise.

2.2. Related Hardware Solutions

The following sections denote important hardware solutions that are closely related to this research. Table 2.1 shows a comparison between ChirpAlert and the different related works.

Static Recorders

Currently available devices like AudioMoth [23] are widely used as low-power autonomous recorders. These devices can capture audible and ultrasonic frequencies to a SD-card for analysis after the full deployment. They establish a strong benchmark for the size, cost and endurance of the autonomous recording part of the system. Another example of such a system is AURITA [24] which combines two recording solutions in order to collect and compare ultrasonic and audible audio at the same time [24]. However, these devices don't perform on-device detection and classification beyond simple sound level based triggers. They overlap with this research in terms of low-power acoustic capture of audible and ultrasonic frequencies in the field while retaining a small form factor. ChirpAlert moves the detection and classification to the device, transmitting alerts over LoRa. This means that the occurrence information can be gathered in near real-time instead of after the full deployment has completed. In addition, different parameters of the process, such as the recording threshold and duty cycle can be changed remotely during the deployment thanks to the long-range communication capability of ChirpAlert.

TinyChirp

TinyChirp [25] proposes a full TinyML pipeline for recognition of corn bunting birds' songs using low-power microcontroller platforms. They compare multiple different types of compact neural architectures and compression strategies on a curated dataset. The authors put emphasis on the deployment gap between lab-grade model and embedded constraints. They report that carefully tuned tiny CNNs can achieve strong accuracy at very low compute and memory budgets. TinyChirp overlaps with this research in terms of on-device inference using spectrogram based features. On the contrary to this work, they only perform their experiments using a custom dataset created from combining recordings from various publicly accessible sources and a custom set of recordings, recorded with high-quality, directional parabolic microphones [26]. They do not perform simultaneous recording using low-cost hardware and verify their system in the field. ChirpAlert complements TinyChirp by showcasing a complete embedded sensor platform, including a low-cost analog front-end and communications stack. We also show an automated species and region aware dataset generation process and perform a field

Table 2.1: A feature comparison between the ChirpAlert platform and other closely related hardware solutions. TinyChirp was excluded as this is not a full system designed for deployment, only for evaluation.

Platform	Inference	Connectivity	Sensors	Microphone	Power	Active Consumption
ChirpAlert	On-device	LoRa, BLE, GPS	Enviro, light, motion	Single, ultrasonic	Solar, Battery	56mW
AudioMoth [23]	None	GPS (Add-On)	None	Single, ultrasonic	Battery	33-132mW
AURITA [24]	None	None	None	Dual, ultrasonic	Battery	720-1055mW
TinyBird-ML [27]	On-device	BLE	Motion	Single	Battery	6mW
BirdWeather PUC[28]	Cloud-Based	BLE, Wi-Fi, GPS	Enviro, light, motion	Dual	Solar, Battery	270mW

deployment where inference is done on the recordings from the low-cost audio front-end.

TinyBird-ML

TinyBird-ML [27] is also a noteworthy related work. They showcase a 1.4g wearable node that acquires acoustic signals while attached to an animal. It performs on-device syllable classification while streaming the results over Bluetooth Low Energy (BLE). They contribute a device capable of performing embedded inference with an extremely low weight, small size and small power consumption, all while providing low-latency embedded inference. TinyBird-ML overlaps with this work in the sense that they perform TinyML classification on severely constrained hardware and provide wireless telemetry. ChirpAlert differs since it is designed for stationary, autonomous sensing for a long duration using solar and battery deployments. We integrate long-range communication as well as BLE while focusing on recognizing species rather than syllables.

BirdWeather PUC

The BirdWeather PUC [28] is a portable, self-contained recording device with an environmental sensor suite. It is designed to collect audio recordings and environment data during a deployment and automatically process them using cloud-computing when connected to the internet. The BirdWeather PUC overlaps with this research as it is capable of continuously recording and monitoring wildlife while also collecting environmental data. The key difference is that the BirdWeather PUC requires an active internet connection to upload the recordings and perform inference using a model with substantial computational resources. When the PUC has no internet connection, it records everything to a SD-card and performs post-hoc analysis of the audio, meaning no real-time inference. In contrast, the ChirpAlert platform is capable of executing inference directly on the embedded hardware without requiring access to additional computational resources. While the BirdWeather PUC is capable of detecting over 6,000 different species using the same model, it lacks the capabilities to do this during remote deployments, especially when considering the energy requirements of uploading the audio data.

3 Theory

In order to understand the considerations made for the final system it is important to understand the concepts of an Audio Event Detection (AED) system, especially in the context of embedded systems. This section will explain the concepts required to understand the AED pipeline, starting at the raw audio and ending at embedded Machine Learning (ML) capable of running on embedded hardware.

Figure 3.1 shows a conceptual overview of the stages in an AED pipeline. The input to the system is raw audio, possibly containing segments of audio that we want to recognize, called an audio event. Examples of such audio events are a dog bark or door slamming shut. Based on the use-case, it could be important to recognize and classify specific sounds in order to act on them. First, the raw audio gets converted from an analog signal into a digital signal such that it can be further processed. Following the digitization, the important aspects of the audio get extracted and converted into so-called features. These features are often chosen with the use-case of the system in mind and can have a significant impact on the system performance. During this stage, optional augmentations can be performed on the data in order to make it more robust against noise or hardware inaccuracies. The last step is processing the features through a machine learning model, pretrained to detect and classify the audio events, producing a prediction for that segment of audio.

The following sections investigate and explain the different stages of this conceptual pipeline, with a focus on recognizing bird species using low-cost embedded hardware.

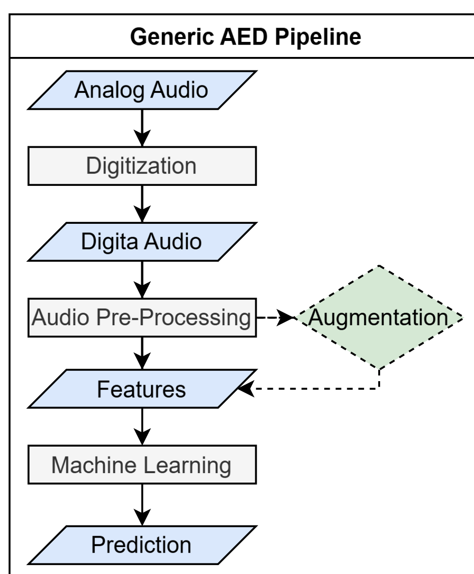


Figure 3.1: Conceptual overview of an Audio Event Detection pipeline. The dashed augmentation step is optional.

3.1. Audio Signals

The starting point of any audio-based system is the actual audio. Understanding how this audio works is essential for the development of a system that can successfully detect and classify different sounds in real-world environments. While the fundamental concepts around audio are easy to understand, the nature of audio is very complex. Not only does it continuously vary over time, it is also very susceptible to noise and often consists of multiple overlapping sources that are producing sounds at the same time. In its most basic form, audio is a mechanical wave where the amplitude of the signal translates to the loudness and the frequency indicates the pitch of the sound. Here, a higher amplitude results in a higher perceived volume and a higher frequency in a higher pitch.

Digitization

Before any meaningful analysis can be performed, the analog audio signal first has to be collected from a microphone source and converted into a format that a computer can process. Figure 3.1 shows that this is done by a process called digitization. The analog audio is converted by sampling the amplitude of the audio signal at a fixed sampling rate. In order to preserve the information contained in the audio, the sampling rate must adhere to the Nyquist-Shannon theorem [29]. This means that in order to accurately capture a bird call with frequency components up to 8 kHz, the signal must be sampled at a minimum rate of 16 kHz. When a lower sampling rate is used, this results in a phenomenon called aliasing where high frequencies are wrongly translated into lower ones. This creates distortion in the audio and can cause important features, such as specific bird calls or environmental sounds to be misrepresented or unusable [30].

Even with a valid sampling rate, a second challenge remains: each sampled value is still a real number, which cannot be stored in a digital computer. The signal first has to be quantized into a discrete value which is a process known as quantization. Each sampled amplitude gets approximated into the nearest value in a fixed set of available values. The number of available values gets dictated by the bit depth, where a 16-bit audio depth can represent $2^{16} = 65,535$ values. Since the quantization process results in an approximation of the actual signal, the error between the real-valued sample and the approximated sample introduces noise called quantization noise. While digitized audio with a lower bit depth requires less memory and processing power, it does increase the quantization error which shows a trade-off between audio fidelity and resource constraints [30]. An example of a digitized audio waveform is shown in Figure 3.4a.

3.2. Audio Pre-Processing

In order to extract more meaningful information from the audio, it can be further processed to better highlight patterns that are present such as the frequencies or temporal changes. After the digitization process, the audio is in its time-domain form, meaning that it is stored as a sequence of amplitude values plotted over time. While this does accurately capture the waveform, it doesn't explicitly showcase any of the underlying frequency components that make up the sound. Many of the features of interest are better characterized by their distribution of energy across frequencies, also called their spectral content, rather than their raw amplitudes. To reveal the spectral content, the sound has to be converted from the time-domain into the frequency-domain.

Short-Time Fourier Transform (STFT)

One of the most widely used techniques for the preprocessing step in Figure 3.1 is the Short-Time Fourier Transform (STFT) [31, 32], which allows the spectral content of an audio signal to be analyzed over short, overlapping frames. First, the signal gets divided into short, overlapping frames. Each of these frames undergoes a discrete Fourier transform to convert it into the frequency domain which results in a set of complex numbers which can be further processed in order to create the spectrogram. The process up to this point is shown in Equation 3.1 [31]:

$$X(n, k) = \sum_{m=0}^{M-1} x[m]w[m - nR]e^{-j2\pi km/M}, \tag{3.1}$$

where

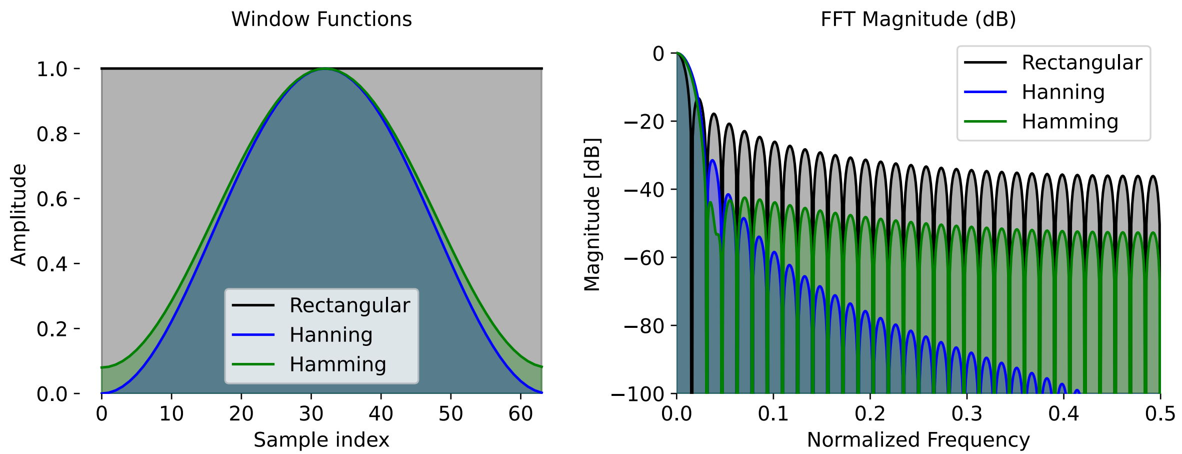


Figure 3.2: Comparison between the rectangular, Hanning and Hamming windows in both the time (left) and frequency (right) domains.

- $x[m]$: The original signal,
- $w[m]$: The window function such as Hanning or Hamming,
- R: The hop size,
- n: The frame index,
- $X(n, k)$: The spectrum of frame n for frequency k,
- M: The window size,
- $e^{-j2\pi km/M}$: The discrete Fourier transform basis function.

The magnitude of the complex numbers showcases the amount of energy present at a given frequency and time and can be computed using Equation 3.2.

$$S(n, k) = |X(n, k)|^p, \tag{3.2}$$

where $S(n, k)$ denotes the magnitude ($p = 1$) or power ($p = 2$) of the spectrum for frame n and frequency k. A higher magnitude or power indicates that the corresponding frequency component is strongly present at that moment of the signal [33]. By placing the results after each other, some time information is preserved and a spectrogram is created. Figure 3.4b and Figure 3.4c show two spectrograms in magnitude and power form, respectively. In contrast to the standard Fourier transform, which showcases a global overview of the frequencies in a signal, STFTs allow for time-localized frequency information to be highlighted, making them more suitable for signals such as bird calls or other audio events. They have shown to be very efficient for encoding audio for further analysis and recognition [25, 34].

Windowing Functions

An important part of the STFT process is the windowing function that is chosen, denoted as $w(m)$ in Equation 3.1. Before the STFT is performed, the continuous audio signal is divided into short, overlapping frames. To facilitate this, a windowing function is used. Figure 3.2 showcases three types of windows: Rectangular, Hanning and Hamming. The trivial way of windowing the signal is by simply dividing it into smaller frames. This has the side effect that this introduces discontinuities at the frame boundaries due to the sharp transitions. This process is called spectral leakage and causes the frequency content to get distorted by other, nearby frequencies. To mitigate this as much as possible, a windowing function is applied to each frame before the Fourier transform gets performed. This function smoothly tapers off the edges of the frame, reducing the influence of the sharp edges of the frame. While this does reduce the spectral leakage and helps preserve the frequency content, it is a trade-off with respect to the time information in the signal. The windowing function most commonly used for STFT based audio processing is the Hanning, or Hann, windowing function as this strikes a good balance between time and frequency resolution [35]. It is a cosine-shaped function and smoothly tapers to zero at both

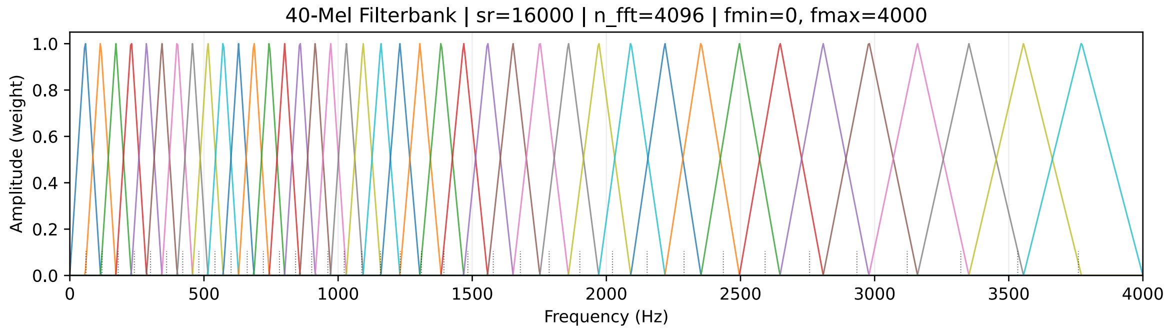


Figure 3.3: Example Mel-filterbank with 40 Mel bins

ends, as shown in Figure 3.2. Another commonly used window is the Hamming window. This is a slight variation on the Hann window which aims to further reduce the amplitude of the sides of the signal. This can help increase the influence of low amplitude frequency components inside the signal.

Mel Spectrograms

While the spectrograms produced by the STFT provide a detailed view of the spectral content of the signal, it still has a limitation in that the frequency axis is linear. This means that all frequency intervals are treated equally but humans perceive sound in a more logarithmic nature. It is easier to tell the difference between 500 Hz and 1000 Hz than it is to differentiate between 7000 Hz and 7500 Hz. As a result, linear spectrograms can overrepresent high-frequency details that contribute little to the signature of an audio signal [32, 36]. To mitigate this, a variation of spectrograms called Mel spectrograms are used, visible in Figure 3.4e. Instead of visualizing the magnitudes of the STFT as with a normal spectrogram, the magnitudes are passed through Mel-filter banks that use the Mel-scale to compress the higher frequencies [32, 36]. Mel-filter banks first convert Hertz to Mel using:

$$m = 2595 \log_{10}(1 + \frac{f}{700}), \tag{3.3}$$

where f denotes Hertz and m denotes Mel. The Mel-scaled frequencies are then passed through a triangular Mel-filter bank, shown in Figure 3.3. The center of each triangle filter has a response of 1, which linearly decreases to 0 on both sides. For each filter bank, the magnitudes from the STFT that fall in that frequency range are multiplied with the response and accumulated, resulting in the spectral magnitude of that filter bank. This process is performed using equation [31]:

$$S_{\text{Mel}}(n, m) = \sum_{k=0}^{k-1} H_m(k) S(n, k), \tag{3.4}$$

where

- $S(n, k)$ is the power spectrogram at frame n and frequency bin k,
- $H_m(k)$ is the mth triangular Mel-filter,
- k is the number of frequency bins,
- $S_{\text{Mel}}(n, k)$ is the Mel-spectrogram at frame n and Mel-frequency bin m.

This process results in a new kind of spectrogram which still represents the spectral content of the audio but is scaled to better match how humans perceive sound. Another benefit is the dimensionality reduction that is inherent to the Mel-spectrogram. Due to the combination of multiple, higher frequency bands into a single Mel-bin, the dimensionality of the data is reduced, thereby reducing the required memory for further processing. Previous research has shown that Mel-spectrograms are very effective for audio classification and detection tasks and are seen as the defacto standard [10, 11, 12].

Log-Mel Spectrograms

However, Mel-spectrograms are not a perfect solution. While they more closely represent how humans perceive the sound, they still represent raw power values which can differ across the different frequency bands or most importantly, different audio recordings. These differences can result in poor generalization of the AED system, especially in noisy or near silent recordings where the high frequencies can dominate the spectrogram. This can be solved by converting the linear power scale to a logarithmic one. Not only does this reduce the influence of large power values but it also increases the visibility of subtle low-power patterns in the spectrogram [37]. This is clearly visible when comparing Figure 3.4d and Figure 3.4f. Further processing these power log-Mel spectrograms and converting to a decibel scale results in a power log-Mel spectrogram, shown in Figure 3.4h. Up to this point, the y-axis is scaled relative to the audio segment that was used to generate the spectrogram. By converting to a decibel scale, everything gets scaled relative to a reference. This means that when analyzing multiple audio segments, the same scale is used across of all of them. Not only does this increase the interpretability of the spectrogram by compressing the dynamic range, it also helps ensure consistent spectrograms across the whole dataset.

Per-Channel Energy Normalization Spectrograms

An alternative modification to log-Mel spectrograms is called Per-Channel Energy Normalization (PCEN) which aims to remove the effects of background noise. While PCEN is capable of using both magnitude and power Mel-spectrograms, it is commonly used with the magnitudes from the STFT. PCEN combines Dynamic Range Compression (DRC), which is also done during the in the process of log-Mel spectrograms, and combines it with Adaptive Gain Control (AGC). DRC is responsible for reducing the variance of foreground loudness while AGC is intended to suppress stationary background noise. Previous research has shown that the resulting spectrogram has improved performance when compared to the traditional log-Mel spectrogram [14, 15, 16]. Figure 3.4g shows a Mel-spectrogram that has been generated with PCEN as a post-processing step. When comparing Figure 3.4g and Figure 3.4h, a clear difference is noticeable with respect to the background noise. Figure 3.4g shows almost no background noise and clearly shows the elements of the bird call. Meanwhile, the bird call is also clearly visible in Figure 3.4h, meaning that the extra computation for PCEN might not be worth it, especially in an embedded context. This shows that there is not a clear winner and both have their benefits.

In short, by converting the raw audio into a log-Mel spectrogram or PCEN Mel-spectrogram, the resulting data is compact, less susceptible to noise and more perceptually meaningful for audio classification and detection tasks. The entire conversion process for both types of spectrogram is shown in Figure 3.4.

3.3. Audio Event Detection

The last step in the Figure 3.1 is the part when the actual detection is done. There are many real-world applications that do not only want to capture audio from the environment, but also what to recognize what and when specific sounds occur. This process is known as Audio Event Detection (AED) [20]. A big challenge in this field is that, unlike in controlled environments, the audio recordings are often noisy and unstructured with multiple audio sources producing sound at the same time. On top of this, different audio snippets can vary significantly in loudness, duration and spectral content making it difficult to run an analysis that works on all of them without issue. More specifically, AED focuses on recognizing a set of predefined sound events inside a longer audio recording, often including their start and end times. The events are often short and stand out from background noise. A typical use-case is speech recognition where spoken words are seen as events. The goal is to be able to understand what the user is saying by translating the audio into a format that the computer can understand. The difficulties of AED immediately become apparent: every human has a different pitch, tone, speaking speed and speaking volume. Another example is environmental surveillance where AED tries to recognize sounds such as a specific bird call or a gunshot. Challenges here are in distinguishing between the different birds or between a door and a gunshot. There are two main types of tasks in AED: detection and classification. The goal of both classification and detection is to be able to determine if one or more known events are present in a given audio segment. The difference however is that with classification, the only interest is to detect if an event is present in the audio, regardless of when. With the detection task, there is also an interest into the exact moment that the event starts and stops. Both tasks can be performed for single or multi-labeled environments, the latter trying to identify multiple, possibly

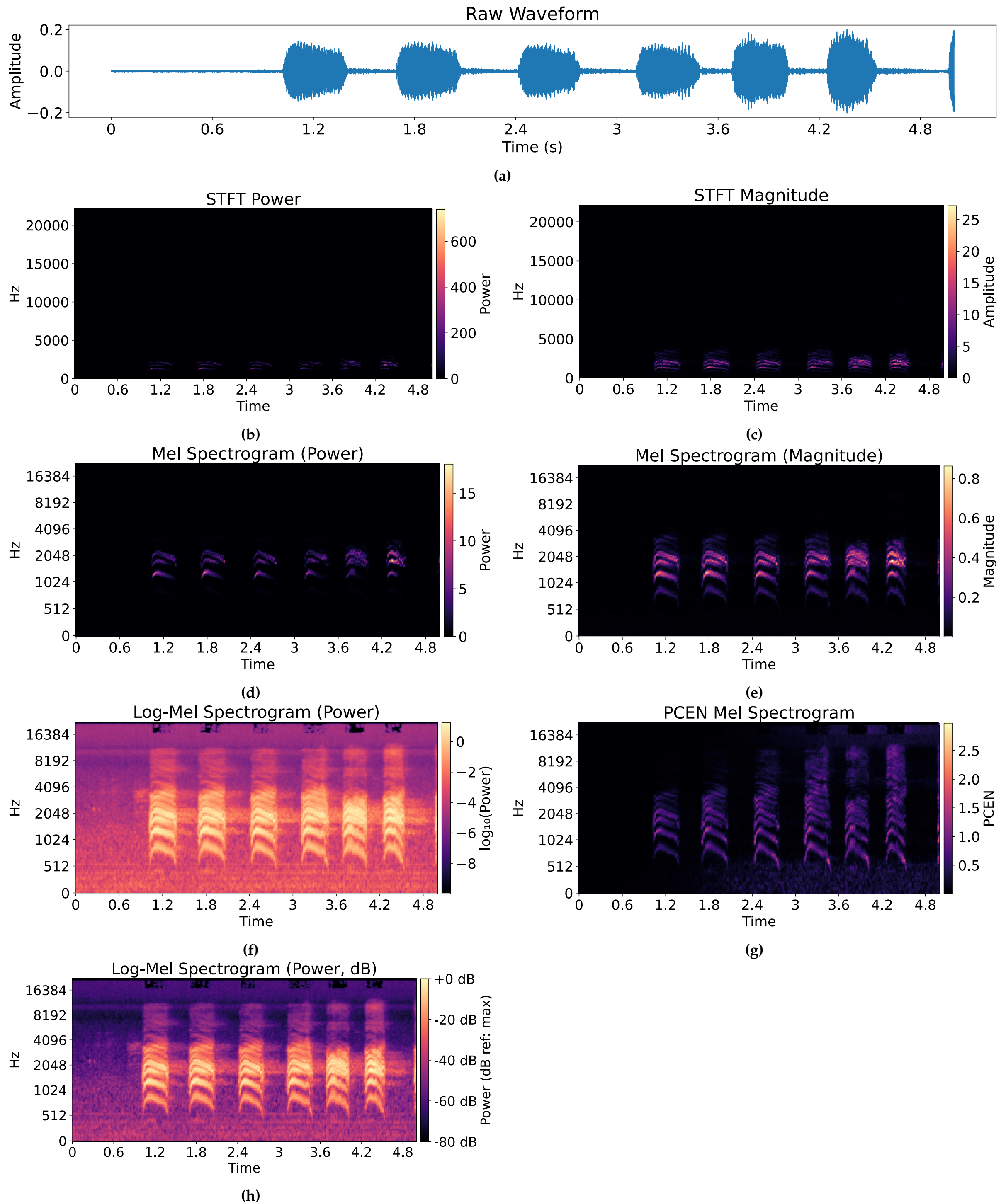


Figure 3.4: Transformation of the audio signal from waveform to a decibel scaled log Mel-spectrogram and Per-Channel Energy Normalization (PCEN) Mel Spectrogram. The left column showcases the required steps to go from the raw audio to a log-Mel spectrogram. The right column shows the required steps for a PCEN spectrogram. (a) Raw waveform before any pre-processing is performed. (b) Power spectrogram resulting from Equation 3.2. (c) Magnitude spectrogram resulting from a Equation 3.2. (d) Power Mel-spectrogram resulting from Mel-scaling a power spectrogram. (e) Amplitude Mel-spectrogram resulting from Mel-scaling a magnitude spectrogram. (f) Log-Mel power spectrogram resulting from logarithmically scaling the power Mel-Spectrogram. (g) PCEN spectrogram resulting from processing the amplitude Mel-spectrogram with PCEN. (g) Log-Mel power spectrogram resulting from converting the power to decibel scale. Images generated using librosa [38] on Xeno-Canto recording XC741985 [39].

overlapping, audio events at the same time.

Despite the challenges of AED, there are also some significant advantages over other methods of environmental surveillance and event detection. Another method that is used for these purposes is computer vision which leverages cameras combined with image processing to perform classification and detection. However, such systems are less robust to environmental conditions such as time of day, lighting or occlusions. For example, a computer vision system might not be able to work at night while an audio based approach is unaffected. In addition to performance in the dark, an audio based system also doesn't require line of sight. The sound can still reach the microphone by bouncing off objects and buildings [24].

In short, AED focuses on identifying one or multiple sound events in a longer audio fragment. The classification task tries to recognize if the event occurred at all while detection also tries to determine the exact timestamps. The main challenges are the variability in the audio such as loudness or pitch, the susceptibility to environmental noise and the possibility of overlapping audio sources.

3.4. Machine Learning for AED

Due to the complex nature of AED, traditional rule-based techniques are often insufficient in terms of robustness due to overlapping events and environmental noise. Hence, the main solution for AED is supervised Machine Learning [40]. A machine learning model is trained on a dataset containing audio recordings and the corresponding annotations. These annotations contain the class of the audio event such as a dog bark and the temporal information such as the start and end time within the recording. The model learns to associate the patterns in the input audio with the classes from the training data, even if they are overlapping. After training, the model can generalize to new, unseen recordings where it can still recognize the classes it trained on.

Accuracy Metrics

An important aspect of AED is how to determine the accuracy of the system. In contrast to audio event classification, AED is also interested in the timing and event overlap. This means that the accuracy metrics need to account for this imbalance in order to correctly showcase the accuracy of the system. The problem with basic accuracy is that many of the frames don't contain any events at all. A model that simply predicts a single class for every frame can exploit this and pretend to be highly accurate [41]. This issue is not only inherent to AED but many ML tasks, thus standard metrics like precision, recall and F1-score are used. Precision measures the proportion of the predicted events that were actually correctly labeled, showcasing how many of the predicted events were actually true.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{3.5}$$

where TP are true positives and FP are false positives. On the other hand, recall denotes the proportion of the actual events that were actually recognized by the model. This showcases how good the model is in actually detecting the events.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{3.6}$$

where TP is true positive and FN is false negative. The F1-score combines both the precision and recall and provides a single measure that rewards models when they are both accurate in their predictions and detection of the actual events.

$$\text{F1-Score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{3.7}$$

Combining precision, recall and the F1-score gives a much better insight into model performance than basic accuracy and mitigates the issues coming from the high amount of frames with no events at all [42].

Lastly, the Recall at Precision (RP) metric is used. This metric captures the recall at a set precision level, prioritizing the recall over the precision. By setting the precision threshold to a high confidence level,

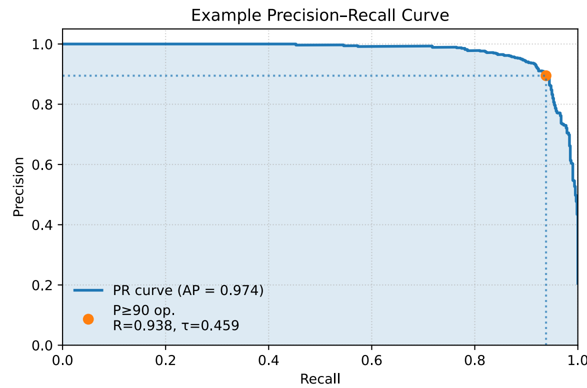


Figure 3.5: Example PR-curve at a precision threshold of 90%. The decision point τ denotes when a class is labeled as class 1 or 2. The recall is 0.938 at a precision of 0.9.

such as 90%, the training process focuses on optimizing the recall while ensuring that the model doesn't simply predict everything as a single class. The RP metric is more difficult to compute than the other metrics. In order to find the highest score, it needs to calculate the precision and recall at different classification thresholds. The model gives a single probability in the form of a fraction between 0 and 1. The higher the score, the higher the probability that the segment contains the target sound. The decision point classifies the score into one of the classes based on the threshold. For example, when the threshold is set to 0.5, every probability lower than 0.5 is labeled as class 1 and everything equal or above as class 2. During the training process, a set of thresholds is used and for each threshold, the precision and recall get plotted. An example of such a curve is shown in Figure 3.5. In the case of the RP metric, the highest recall gets selected at the set precision threshold. This is the RP metric for that evaluation step. It is important to note that this metric optimizes for the recall value, meaning that if there is a higher precision value at an identical recall, this is not always found. It does however offer two tunable parameters that can help with the performance of the model. The threshold value can be set higher to force a higher confidence level, requiring a higher probability to get classified as class 2. The precision value can be set higher to force a lower rate of false positives at the risk of missing some target sounds. Both can be decided based on the deployment parameters.

However, better accuracy metrics do not solve all issues in determining AED performance. Since audio is a continuous signal, the accuracy can differ on different levels over granularity. The two most common levels are frame-level and clip-level. During frame-level detection, a prediction is made at every time step of the recording. This level of granularity offers the highest temporal resolution as a single frame can range anywhere from milliseconds to a couple of seconds in duration. For each frame, the system predicts which (possibly overlapping) classes are active. This level also offers the fastest response time as it requires the least amount of audio to detect if a class of interest is active. However, if the patterns of the class are only unique over longer sections of audio, this method might also result in a high misclassification rate and be less robust to noise overall [43]. Another big challenge of frame-level detection is when the dataset contains weakly labeled data. This means that the exact timestamps are not present but only general intervals. For example, between second 3 and 6 of the recording, a blackbird chirps.

The next level of granularity mitigates these issues by making a single prediction for an audio clip, often in the range of 1 and 10 seconds, instead of per frame. The frame-level predictions are aggregated together such that a single, more robust prediction is made. While this improves the accuracy and robustness to noise, it increases the coarseness of the timing information which might be a deal-breaker depending on the use-case [44]. An important note is that this type of AED is often referred to as audio tagging instead of detection [20].

Data Augmentation

As mentioned in section 3.3, a key challenge in AED is that the sound events are often noisy and unstructured. If this was not accounted for during training, the model might suffer when predicting on

unseen recordings. To address this, a process called data augmentation is performed on the training dataset. This is an optional stage in the pipeline shown in Figure 3.1 and can be performed on the audio or the spectrograms. This section will explain the ones used in this research but is by no means an exhaustive list.

Gaussian Noise

Gaussian noise is added to the audio by adding random values, sampled from a normal distribution, to the audio waveform. This process simulates background noise or imperfections in the microphone that may be present in the real-world recordings. The purpose of this method is to increase the model's robustness in situations where the signal-to-noise ratio is low. The amount of noise added to the waveform is typically scaled to the amplitude of the signal to prevent the noise becoming the dominant factor in near silent recordings.

Loudness Augmentation

Loudness augmentation randomly scales the amplitude of the audio. It's important to note that the entire waveform is scaled by the same factor to preserve the patterns in the original audio. This simulates variations due to distance, signal bouncing or varying microphone gain. It encourages the model to focus on learning the frequency patterns instead of focusing on the amplitude patterns.

SpecAugment

SpecAugment [45] aims to help the model become robust to deformations in the time direction, partial loss of frequency information and partial loss of small segments of data. It is mainly used in speech recognition tasks but has also been used in different tasks like bird recognition [46]. SpecAugment performs three operations on the spectrograms: Time warping, frequency masking and time masking. The time warping operation involves slightly shifting parts of the data along the time axis to mimic temporal distortion. The frequency masking operation randomly selects one or more frequency bands inside the audio and sets all of their values to zero, simulating their loss in a real-world audio recording. The final operation of time masking randomly masks out sections of the audio, simulating the audio dropping out for a short amount of time or interference. The combination of these three operations help ensure that the model is more robust in real-world environments.

Mixup

Mixup is an augmentation strategy that generates new training samples by linearly interpolating between pairs of existing samples and their respective labels. The technique was originally developed for image classification [47] but has since been proven useful for audio based recognition, given that they use spectrograms as input [48]. The main advantages of Mixup for AED are that it encourages linear behaviors between classes, which helps reduce overfitting and improves generalization. It can also simulate overlapping sound events due to the linear interpolation, allowing the model to learn more about overlapping events.

Convolutional Neural Networks

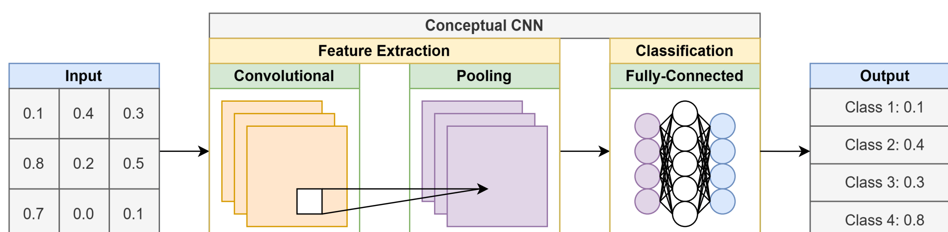


Figure 3.6: Conceptual overview of a Convolutional Neural Network (CNN) showcasing the three types of layers and how they interact.

Convolutional Neural Networks (CNNs) have become the standard approach for spectrogram-based AED tasks. They provide a trainable, data-driven solution that can work directly on the spectrograms. A CNN is built with neurons, each containing trainable weights that can be altered changed through training. Each neuron takes in an input, and based on the neuron's weight, it produces an output.

The neurons are stacked in order to form 2-dimensional and 3-dimensional layers. A basic CNN has three types of layers: convolutional, pooling and fully-connected layers. These layers are placed after each other in order to create the full neural network. The last layer of the network also contains a loss function. During training, this function compares the predictions of the model with the ground-truth labels of the input data in order to determine the accuracy, or loss of the network. Based on the loss, the optimizer updates the weights of the network in order to improve the loss during the next training step [49, 50]. Figure 3.6 shows a conceptual overview of a CNN.

Convolutional Layer

Convolutional layers are responsible for processing the input by using small, trainable filters. Each filter is applied across all possible positions of the input, producing a response that indicates how strongly a particular pattern is present at that position. Because the same filter is reused across the input, the network can detect the same signature, regardless of when it happens in the input. In the case of spectrograms, this means that these filters can find an audio signature regardless of when it happens within the spectrogram. The outputs of all filters are collected into feature maps which are passed on to the next layer.

Pooling Layer

The pooling layers are responsible for dealing with small imperfections in the patterns, resulting from noise or deformations in the audio. The audio event might be slightly stretched or partially masked by background noise. The pooling layers account for this by summarizing small regions of the feature maps and summarizing them using simple operations such as averaging. By keeping only the average response of each region, pooling produces a representation that is more robust to small changes in the audio pattern. At the same time, this means that the pooling layers reduce the dimensionality of the data as multiple regions of the feature maps are compressed into a single representation.

Fully-Connected Layer

After multiple layers of convolution and pooling, the network has reduced the dimensionality of the data significantly but it still has to be converted into the dimensionality of the output. The fully-connected layers are responsible for this mapping and achieve this by connecting every neuron of the last layer to every neuron of the output layer. Each of these neurons has a trainable weight that converts the inputs to a single output. This means that the actual classification into one of the output classes is done by the fully-connected layers. Because these layers can combine the information from all learned features simultaneously, complex decision boundaries can be formed using the weights of the neurons.

Loss Functions and Optimizer

A fundamental component of training a CNN or machine learning model in general is the loss function. This function quantifies how well the model's predictions align with the truth and help guide the learning process. Not only does it help estimate the performance of the model, it also helps guide the model into the right direction during training. The type of loss function can affect the model's behavior such as penalizing a type of error more than another while it also helps prevent overfitting and improves generalization. An important distinction to make is between the loss function and the other metrics. The loss function only indicates the error of the model when comparing it with the truth, while metrics are more tailored to indicate the performance of the model with a certain goal in mind. The loss function is often more stable, meaning that it is better suited to help the optimizer tune the model. The metrics themselves have nothing to do with the actual optimization process but can indicate when sufficient performance has been reached.

The loss function works by calculating the loss after each step of training and using an optimization algorithm to update the model's parameters in order to minimize the loss. This process is repeated until the model either reaches a sufficient amount of performance or the training time has been used up. In the case of neural networks, this means that the optimizer tweaks weights and biases of the layers. A commonly used optimizer is the Adam optimizer. This is a stochastic gradient descent-based optimizer that is both computationally efficient, requires a low amount of memory and is well suited to deal with large amounts of data and parameters. The loss is fed into the Adam optimizer after each step, after which it uses stochastic gradient descent to determine the magnitude and direction of the weight updates. The Adam optimizer has been shown to work well in practice and is thus widely used [51].

For this research, two types of commonly used loss functions were used: Cross Entropy (CE) loss and Focal Cross Entropy (FCE) loss. Both utilize cross entropy [52] as their fundamental building block but FCE expands CE by putting more emphasis on hard/rare cases. In the case of AED there are often much more negative examples than positive meaning that CE spends more of its budget on easy examples. For simplicity, this section will explain further with Binary Cross Entropy (BCE) which is easily expanded for multi-class classification problems. BCE is calculated by:

$$\mathrm{BCE}(p, y) = \begin{cases} -\log(p) & \text{if } y = 1 \\ -\log(1-p) & \text{otherwise,} \end{cases} \tag{3.8}$$

where $y \in [0, 1]$ denotes the ground-truth of the sample and $p \in [0, 1]$ defines the model's estimated probability for the class with label 1 [53]. In order to expand this to the formula for FCE, p is substituted by p_t following:

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1-p & \text{otherwise,} \end{cases} \tag{3.9}$$

resulting in:

$$\mathrm{BCE}(p, y) = \mathrm{BCE}(p_t) = -\log(p_t). \tag{3.10}$$

Finally, this gives the equation for focal loss:

$$\mathrm{FL}(p_t) = -\alpha(1-p_t)^{\gamma}\log(p_t), \tag{3.11}$$

where α is a class weighing factor and γ a focusing parameter. α helps balance the importance of positive and negative samples while γ helps differentiate between easy/hard examples [53]. As this research deals with highly imbalanced datasets, both of these loss functions will be evaluated.

3.5. On-Device AED

While the usage of machine learning models has greatly improved the performance and possibilities in AED, most of these systems have access to vast amounts of resources such as memory and CPU/GPU computing power. In contrast, the goal of this research focuses on on-device deployment without any access to such infrastructure, resulting in new constraints for the AED system. Not only must the system be powerful enough to run an AED model with reasonable accuracy, it must also be able to simultaneously and continuously listen to the environment, perform real-time audio processing and do all this while using as little energy as possible. In order to facilitate these constraints, the full scale model needs to be converted to be able to run on low-cost embedded hardware. Not only does this entail reducing the computational and resource requirements, it also means converting the preprocessing methods to be function on-device. Figure 3.7 shows a conceptual overview of the stages required to go from a desktop model to an on-device solution.

TinyML

One of the key challenges of TinyML lies in the memory requirements of the models. Compared to mobile machine learning, which runs on mobile phones, TinyML has orders of magnitude less memory available making a lot of explored architectures simply not fit on embedded systems. These systems often only have a couple of megabytes of flash storage and RAM sizes up to a megabyte. A very simple ML model can easily exceed these memory constraints if it's not designed with memory efficiency in mind [54]. The memory usage of a ML model is measured in model size and peak runtime memory. The model size dictates how much flash is required to store the trained model on the embedded system. The peak runtime memory measures the maximum amount of RAM needed during inference. This includes the memory required by the model and the intermediary buffers such as for the input and output.

In addition to the limited memory, the embedded systems also have limited processing speeds. They often only have a single processing core running up to a couple of hundred megahertz. They also

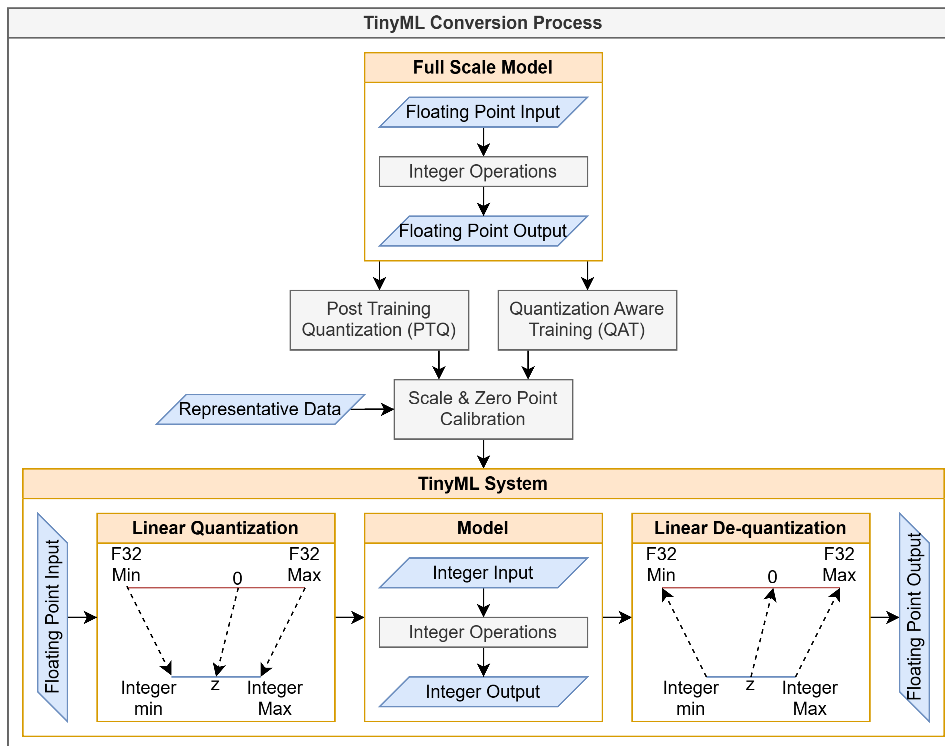


Figure 3.7: Conceptual overview of converting a desktop AED model to run on embedded hardware. The model gets quantized to use integer operations using PTQ or QAT and the scale and zero-point values are calculated based on representative data. For the linear quantization, z denotes the zero-point value which maps the F32 zero to the integer range.

rarely have any form of hardware acceleration such as GPUs or TPUs. As a result, the inference time for the TinyML model can become a bottleneck, especially when the use-case requires real-time predictions. In recent years, new microcontrollers have become commercially available that do support hardware acceleration, opening up a lot of possibilities [55]. The computational requirements of ML models commonly gets measured in three different metrics: Floating Point Operations (FLOPs), Multiply-Accumulates (MACs) and inference latency. The amount of FLOPs showcases the amount of floating point operations required for a single inference. A big advantage is that this metric is platform independent, but this metric doesn't work for integer models and doesn't account for memory access costs. This means that the actual performance on the target system is difficult to estimate. The MACs metric is a subset of the FLOPs metric and simply counts the multiplications and additions required for a single inference, regardless of the data type. One of the main benefits is that this works for all data types but it has similar downsides to the FLOPs metric. The MAC metric has been shown to be very effective at predicting model complexity in TinyML [56].

In contrast to FLOPs and MACs, the inference time metric is very system specific and can only be obtained by measuring the time for a single inference. This metric can then be used to calculate if the system meets real-time constraints. A downside of this metric is that it can only be obtained through experimentation or simulation, making it difficult to estimate before any hardware is selected.

The final main challenge of TinyML lies in the energy consumption of the system. Many embedded systems rely on battery-powered or energy harvesting based systems which impose strict constraints on the power consumption of the entire system. This means that the computationally expensive and thus energy intensive TinyML component of the system has to share its power budget with other aspects of the system such as the microphone or any form of wireless connectivity. In order to be feasible for long-term deployment, such a system is expected to use power in the range of a couple of milliwatts [57]. The main metric used to measure the energy consumption of the system is the energy usage per inference, calculated by measuring the system power consumption.

All in all, this means that in order to be able to run AED on the embedded hardware, a suitable system

must have a small memory footprint in the range of a couple of hundred kilobytes. Additionally, it must use as little processing power as possible to reach real-time inference constraints and energy constraints in the range of a couple milliwatts.

TFLite Micro

One of the big driving factors about the rapid expansion of the TinyML space is a framework called TensorFlow Lite (TFLite) Micro [58]. TFLite Micro was purpose built to address the challenges mentioned in Figure 3.5 and help improve the deployment of machine learning on embedded systems, including microcontrollers. It is an inference engine designed to run models that were pretrained on less constrained hardware on ultra-low power and memory constrained devices. In addition to providing the actual operations that run on the embedded device, the TFLite micro framework also supplies the necessary tools to convert and deploy the ML models.

Quantization and Optimization

In order to mitigate the three main challenges of TinyML, model compression and optimization techniques have been developed to help reduce memory and computational footprints at the cost of accuracy. The first step is model quantization which reduces the precision of the weights and activations in the model, as shown in Figure 3.7. Models that run on large infrastructure often use 32-bit Floating Point (FP32) while TinyML often uses 8-bit or 16-bit integers. Not only does this save memory, it also helps reduce computation as integer math is often less expensive than floating point math in terms of computation [59], even when a dedicated Floating Point Unit (FPU) is available. There are two methods of quantization that are commonly used: Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT). With PTQ, the model is fully trained using floating point and only converted after the entire training process is done. With QAT, the training process tries to compensate for the quantization-related errors during training by simulating the quantization noise during the model evaluation at each training step. While QAT has shown to reduce the accuracy impact of quantization, it does suffer from a longer training process and less flexibility during deployment [60].

Calibration

Both types of quantization have to deal with converting the floating point range into integer range in order to correctly convert the input and outputs of the quantized model. This operation is done by using the scale and zero point values. To correctly set these values, a calibration process is performed that requires data representative of the deployment. The calibration process runs a set of samples through the model and records the minimum and maximum values for the input and output. Based on these values, the scale and zero-point values are calculated using

$$\text{scale} = \frac{\max - \min}{q_{\max} - q_{\min}}, \tag{3.12}$$

where $q_{\max}$ and $q_{\min}$ denote the maximum and minimum values of the quantized range, in the case of an 8-bit integer, this is [-128, 127]. The zero-point is calculated using

$$\text{zero_point} = q_{\min} - \frac{\min}{\text{scale}}. \tag{3.13}$$

These values are then combined into

$$X_{\text{quant}} = \text{round}(\text{scale} \cdot X + \text{zero_point}), \tag{3.14}$$

in order to quantize a value X and

$$X = \frac{X_{\text{quant}} - \text{zero_point}}{\text{scale}}, \tag{3.15}$$

to dequantize a value X. This allows the floating point data to be converted into integer values while having the side effect of losing precision. The loss is called the quantization error which has a similar effect as the digitization error explained in section 3.1.

4 Methodology

This section walks through the different steps taken from the start to the finished sensor and Audio Event Detection (AED) pipeline. It starts at the breadboard prototype phase and goes through the entire process which resulted in a sensor capable of running AED on-device and a pipeline to go from raw audio to developed features.

4.1. Breadboard Prototype

To test the feasibility of a system that can continuously record sound as well as simultaneously run AED, a breadboard prototype was made. Based on literature research and previous experience, it was decided to go with an STM32 MCU. Not only because of the familiarity with the platform but also because they offer peripherals that can be used to help reduce the computational complexity of the entire system. The STM32U5 line-up of chips was chosen since it offers up to 4 MB of flash and up to 2.5 MB of RAM. In addition to a large amount of memory, it also offers advanced peripherals that can help optimize the system in the future. STMicroelectronics also offers two state-of-the-art platforms that can help optimize the model deployment, performance and memory footprint: STM32 Edge AI and the STM32 model zoo. For the breadboard prototype, a readily available development board for the STM32U5 lineup was combined with an I2C environmental sensor, analog MEMS microphone and amplification circuit and SD card receptacle. The focus of this design was to get a basic understanding of the most important parts of the system and attempt to identify important considerations before moving into the PCB design. Another big benefit of creating a breadboard prototype using the same type of components is being able to write software while the PCB design was being manufactured, limiting the amount of time spend waiting. This section describes the different aspects of the final system that were investigated.

Analog Front-end

Before working on this thesis, I had limited knowledge regarding analog circuit design. Based on the performed literature research, an analog microphone front-end was chosen as this showed the best power characteristics, gives better flexibility regarding the gain of the circuit and since there were no readily available digital microphones that supported ultrasonic frequencies. The circuit was first simulated using a program called LTSpice [61]. This allowed easy testing of different types of amplification circuits such as inverting and non-inverting amplifiers without the risk of damaging any components. After multiple tests, a non-inverting amplification circuit was chosen due to the preferable characteristics for audio which was implemented on the breadboard prototype as shown in Figure 4.2b. The simulation results can be seen in Figure 4.2a.

Micro SD-Card

During the initial research for this project, it became apparent that using external storage in the form of a Micro-SD card was the most common way for long-term storage. As I was unfamiliar with a Micro-SD card slot or designed my own PCB with it, the functionality was tested as well as the STM32 ecosystem. Using a readily available Micro-SD card adapter which supported the full, 4-bit SDIO interface of the STM32U5, a basic system was developed to store the recordings retrieved from the analog front-end.

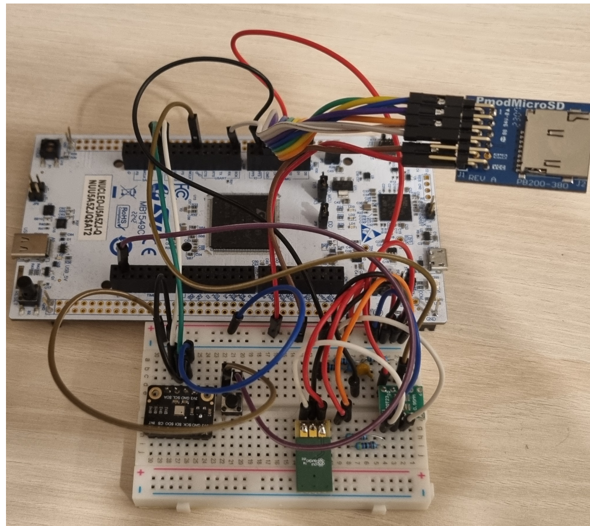
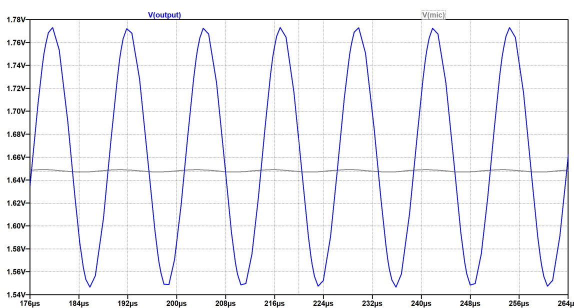
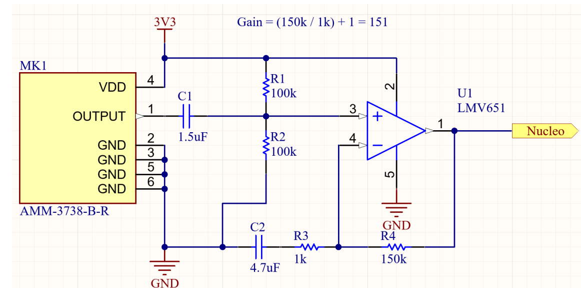


Figure 4.1: Breadboard prototype using an STM32U5A5AJ-Q Nucleo Board, an AMM-3738-B-R microphone combined with an LMV651 non-inverting op-amp circuit, a BME688 environmental circuit and a PMOD micro-sdcard adapter.



(a) Simulation results from the analog front-end with LTSpice at 80 kHz with a bias of 1.1V and an amplitude of 0.001V. This shows 151x amplification of the input signal (mic).



(b) The circuit used for the analog front-end on the breadboard using the AMM-3738-B-R microphone combined with an LMV651 non-inverting op-amp circuit (151× amplification).

Figure 4.2: Analog front-end: (a) LTSpice simulation; (b) corresponding breadboard circuit.

The FileX file-system was used as that was natively supported by the MCU. At first, the 4-bit mode of the SDIO interface would not work. Researching the issue showed that the design considerations for the clock, data and command lines of the SD-Card interface are very specific. The data lines have to be very short and have to be shielded from noisy components such as high speed communication interfaces such as I2C. In addition, the length of the command, clock and data lines has to be closely matched in length to prevent read or write errors. All of these considerations were important to note before moving to the actual PCB design.

I2C

As mentioned before, my level of knowledge regarding the analogue part of PCB design was very limited. Initial research for this project showed that I2C can be a mayor disruptor of an analog signal such as the analogue front-end. In order to get familiar with such disruptions and learn how to prevent them, an I2C environmental sensor was added to the prototype to investigate the effects of different I2C speeds. Figure 4.3 shows that there is crosstalk on the microphone line while there is communication with the I2C sensor, meaning this is something to consider while designing the sensor.

Feature Extraction

Another big part of testing the feasibility of the system is to ensure that the system is capable of simultaneously gathering data and processing the data. A significant part of this is extracting the log-Mel spectrogram features from the audio in a reasonable amount of time. In order to ensure that this was possible, the feature extraction was implemented for the prototype. One of the key factors in making this possible in real-time is the usage of the CMSIS-NN library, which is developed for the

Figure 4.3: Crosstalk on the Microphone line when communication is happening with the BME688 I2C sensor. During this image, the microphone is recording and the temperature and humidity is read from the BME688. The blue and yellow lines are the I2C lines and the purple line the microphone line. Before the I2C communication, the image shows that the analog line contains significantly less noise than during the communication.

ARM Cortex-M core contained in the STM32U5. This library leverages precomputed coefficients and ARM specific Single Instruction Multiple Data (SIMD) instructions to speed up the computational process. Combining these software optimizations with the STM32AI Pre-processing library developed by STMicroelectronics helps create the spectrograms as efficient as possible. A big constraint of the feature extraction was to be able to generate these features quick enough so that the AED model can still run its inference before the data buffer filled up again. The results from this test show that it took around 500 - 800 ms to create a log-Mel spectrogram of a 3-second audio snippet, depending on the parameters used, which is well within the real-time constraint.

TFLite Micro

One of the final parts before moving on to the actual sensor design involved testing the TFLite Micro and STM32 Edge AI components. In order to ensure that the STM32U5 platform was able to run TinyML models on the hardware and get more familiar with the software, a couple of small models were created and ran on the MCU. This all went without issues except for a specific TensorFlow model which contained an operation that is not supported by TFLite Micro. This is an important consideration to keep in mind while selecting the model architecture.

Bluetooth

One of the important requirements of the system is that it has to be configurable and reachable using Bluetooth Low Energy (BLE). Not only does this help with the ease-of-use, it also allows the researcher to change parameters on the fly or read the system status. In order to get familiar with the STM32 BLE stack, a STM32WB55-based board was programmed to perform simple functions over BLE and interface with the main MCU. This allowed for testing of the interactions as well as estimate the power characteristics.

Design Considerations

Based on the results of the breadboard prototype, the following considerations have to be kept in mind for the PCB design:

1. Analog design sensitivity: Analog front-end circuits are highly sensitive to inference. They will need to be shielded from the I2C and SDIO lines.
2. SDIO Trace Design: The SD-Card interface requires matched and short trace lengths for the CMD, CLK and DAT lines.

4.2. Training Datasets

For this research, the AED task is limited to recognizing a specific bird species at a time. In order to help verify the system performance, three main datasets are used: BirdCLEF2021, BirdSet, and ESC-50. In

order to accurately verify the system performance, the datasets are divided into training, validation and test sets based on an 80%/10%/10% ratio. During training, the model doesn't see the validation or test sets to help ensure that the results are representative. If this was not the case, the model would be evaluated on data it was trained on, meaning that it would not give a good representative view of the performance on unseen data.

BirdCLEF2021

The BirdCLEF2021 [21] dataset was selected since it contained a combination of short, annotated training recordings and long, annotated soundscape recordings. This combination is particularly well-suited for developing and evaluating the AED system as it more closely mimics the real-world deployment. The system is trained on the short recordings which contain limited background noise and almost no overlapping sources. After training, the system is evaluated on the soundscape recordings to determine the performance of the model in a more real-world scenario where the sound sources can overlap and disrupt each other. The soundscapes are also much longer and also contain segments without bird calls. This allows for the evaluation of silence/background noise detection. The BirdCLEF2021 datasets contains 60,000 short, annotated audio recordings divided into 397 bird species, all sampled from the Xeno-Canto community library. In addition to the short recordings, the dataset also contains 100 10-minute soundscapes, containing a variety of birds.

BirdSet

The Birdset [62] dataset is a large scale benchmark dataset which focuses on avian bio-acoustics. It contains over 6800 hours of training recordings spanning over 10,000 classes. Additionally, it also has more than 400 hours of recordings across 8 labeled evaluation datasets. This dataset was included as an alternative to the BirdCLEF2021 dataset after the soundscape recordings were found to be of such low quality that they were unusable. The Birdset dataset is divided into smaller subsets, of which the High Sierras Nevada subset was used for this research. This contains 5,460 training recordings across 21 classes. In addition, it includes 12,000 5-second test recordings for evaluation. While this dataset was not used for any formal experiments, it was used in developing the AED pipeline.

ESC-50:

The ESC-50 [63] dataset contains 50 classes of environmental sounds such as rain and animals. It is widely used as a benchmark for environmental sound classification and is selected to support the training and evaluation of the no-call recognition part of the system. It contains 2000 short audio recordings, sampled at 44.1 kHz, that will help train the model to distinguish between environmental background noise or noises of interest.

4.3. Key Parameters

Even though the available parameters can differ greatly between the experiments, there are some consistent parameters that are import to understand. Mainly the class definitions, number of Mel-bins and number of frames are important.

Class definitions

Across the experiments, there are four possible classes. The `target` class is present in all experiments and denotes the sound signature of the bird species of interest. The `other` class defines anything other than the target sound. During experiments 1 through 2, this does not include background noise, as this is classed as `background`. After the results of experiments 1 and 2, experiment 3 defines a new class called `other_and_back` which groups the `background` and `other` class into one in order to go from a multi-class classification problem to a binary one. From experiment 4 and onwards, only the target class remains for simplicity. This was not possible before due to the constraints of the STM32 AI ModelZoo framework.

Mel-bins

Different counts of Mel-bins have been evaluated. As mentioned in section 3.2, the output of the Short-Time Fourier Transform (STFT) is passed through a Mel-filter bank to group the linear frequency scale into Mel frequency bins, reducing the dimensionality while preserving the frequency information

that is more relevant to human perception. The number of Mel-bins has a direct effect on the resolution of the resulting spectrogram such that a lower amount of Mel-bins results in a more aggressive compression. While this does result in a smaller memory and processing footprint, it also reduces the detail of the frequency components. In contrast, a higher amount of Mel bands capture finer frequency details, possibly resulting in more information at the increased cost of memory and slower inference. Four different values are evaluated: 64, 80, 96 and 128 Mel-bins. These configurations are commonly used on embedded hardware and are fully supported by the ARM CMSIS-NN library which is used for the embedded implementation.

Number of Frames

The number of frames in a spectrogram determines the temporal resolution and duration of the segment passed into the model. As described in section 3.2, once a waveform is converted to a Mel-spectrogram, the time information is discretized into overlapping frames. The number of frames controls how much time the model sees at once and how fine-grained the time resolution is across the input. Choosing this value means balancing between accuracy and resource constraints as a higher number of frames increases the memory and computational requirements for the model. While a lower value means faster inference, it does introduce a risk of missing longer patterns in the audio. During the first experiments, four values are tested: 32, 64 and 96. In order to calculate the amount of seconds S in a single window, Equation 4.1 is used.

$$S = \frac{(F-1)R}{f_s}, \tag{4.1}$$

where S denotes the frame duration, F denotes the number of frames, R hop length in samples and f_s the sample rate in Hz. At 16 kHz with a hop length of 512, 32 frames equates to

$$S = \frac{(32-1)*512}{16000} \approx 0.992 \tag{4.2}$$

seconds and 96 frames equates to

$$S = \frac{(96-1)*512}{16000} \approx 3.040 \tag{4.3}$$

meaning that the chosen variables range from approximately 1 to 3 seconds of audio as input to the model. 128 frames was also attempted during the ModelZoo experiments but a lot of recordings were not long enough to support this, resulting in a lot of errors.

With the new framework, this is changed to always encompass 3 seconds of audio. How to calculate the amount of frames based on the other experiment parameters is further explained in section 4.6.

Sample Rate

The sample rate is a very important parameter as it does not only directly impact the resource requirements of the model, it also has a big effect of the amount of information in the data. As mentioned in section 3.1, the usable frequencies are determined by the Nyquist-Shannon theorem. In other words, if the sound signature has frequency components at 6 kHz, the sample rate must at least be 12 kHz. However, a higher sample rate means more data to process which can introduce longer pre-processing and inference times. The sample rate also has a big impact on the other parameters of this section, as shown in Equation 4.1. For the first three experiments, the sample rate is fixed to 16 kHz based on the frequencies of the three target classes. This value was reached by analyzing the sound files for all the target birds in the BirdCLEF2021 dataset using a Python script that scans for the lowest, highest and peak frequency. The peak frequency is defined as the Fast Fourier Transform (FFT) bin with the highest mean energy across all the files, while the minimum and maximum are defined as the lowest and highest bin with at least 5% of the energy of the peak bin, respectively. The results are shown in Table 4.1. An important note is that this will also include microphone and background noise which can skew the results. Thus based on the peak energies all well below 8 kHz and other research [64, 65] showing that two of the target species sing below 8 kHz, a sample rate of 16 kHz was chosen for the first experiments. In later experiments, this is expanded to 32 kHz in order to make the system more generic across species.

Table 4.1: Sample rate exploration for experiments 1 through 3. The peak frequency is defined as the FFT bin with the maximum mean energy across all the input files. The minimum and maximum frequencies are defined by finding lowest and highest FFT bins what have at least 5% of the energy of the peak value, respectively.

Species	Min(Hz)	Max(Hz)	Peak(Hz)
`rucwar`	348.5	9995.6	3097.6
`bobfly1`	253.0	10208.0	2580.9
`reevir1`	412.4	8227.5	2169.1

4.4. Generic Pipeline Structure

Even though the exact pipeline differs between the STM32 AI ModelZoo and the custom framework, there is a general structure that they both follow, shown in Figure 4.4. First, the raw audio divided into smaller chunks, the size of which is determined by number of frames in the experiment configuration. Each chunk is then passed through a band pass filter and converted into a log-Mel or Per-Channel Energy Normalization (PCEN) spectrogram as explained in section 3.2. This spectrogram is the input to a Convolutional Neural Network (CNN), which predicts if the target signature is in the audio chunk. After model inference, different metrics are used to evaluate the performance of the system.

During training, different augmentations, explained in Figure 3.4 can be applied to either the raw audio or the spectrogram to help make the model more robust against background noise and hardware inaccuracies. It is important to note that these modifications are only done during the training and are fully optional.

After the experiments, a final pipeline is selected based on its performance. This pipeline is implemented on the sensor platform and used for the final real-world test. During the conversion stage of this project, the on-device optimization techniques explained in Figure 3.5 are used to allow the pipeline to run on resource constraint hardware.

4.5. ModelZoo Experiments

The following three experiments were performed using the STM32 AI ModelZoo framework. The goal of these experiments was to find the best performing audio pipeline for on-device AED, balancing performance with resource requirements. For every experiment, a new configuration is tested which could take form of a new pipeline architecture, model or parameter configuration. Each configuration is evaluated using the metrics mentioned in section 3.4 and the best performing configuration is selected for the final real-world test.

STM32 AI ModelZoo

The STM32 Model Zoo [66] is a collection of pre-trained and optimized neural networks, supported by a dedicated machine learning framework tailored for STM32 microcontrollers. In the context of this research, the Model Zoo provides useful starting points and reference pipelines for AED, enabling rapid experimentation without the need to design models from scratch. This allows the focus to remain on the development of the sensor hardware and embedded software. A key advantage of the STM32 Model Zoo is its integration with STM32Cube.AI, which streamlines deployment to the target microcontroller and provides built-in tools to profile inference latency, memory usage, and flash consumption directly on the device.

For this framework, the ModelZoo was modified and expanded using Python scripts in order to allow for automated evaluation of different configurations as well as deeper analysis of the results.

Experiment 1: One-Stage Classifier

This experiment uses a singe three-way classifier to distinguish between the three classes: `target`, `other` and `background`. The model is trained on the BirdCLEF2021 dataset for the `target` and `other` classes, while the environmental sounds from the ESC-50 dataset are used to train the `background` class. Multiple types of Machine Learning (ML) models are tested in order to see which is the best in terms of accuracy and speed. The STM32AI Model Zoo comes with a couple models that are already heavily optimized for on-device computing. Two of them are particularly interesting for the AED task: Mini

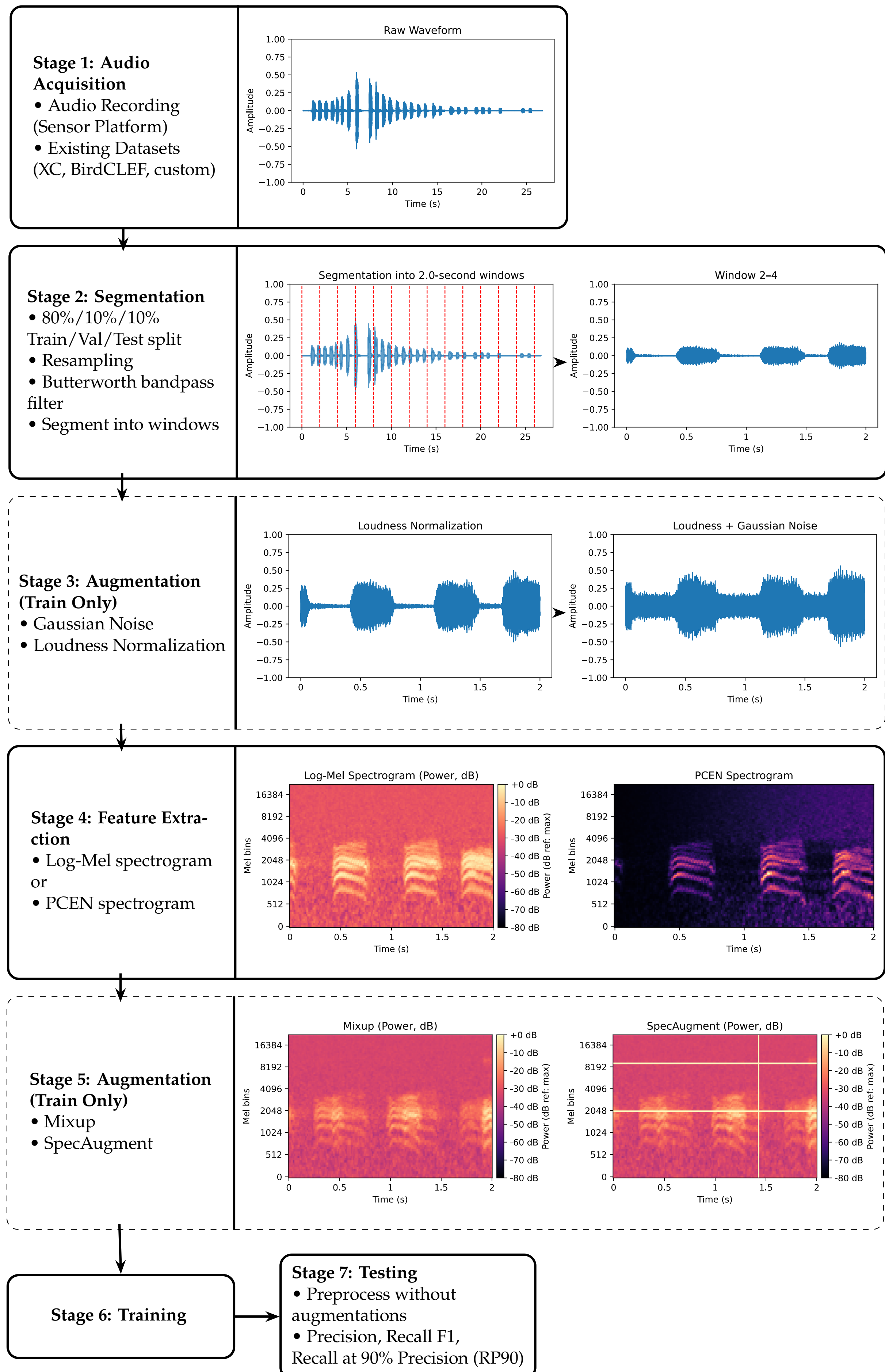


Figure 4.4: Generic pipeline structure and outputs used for experiments 1, 2 and 3. The dashed stages are optional and only done during training. It is important to note that PCEN uses magnitude spectrograms as an input.

Table 4.2: Parameters used for experiment 1.

Parameter	Values
Species	`rucwar`, `bobfly1`, `reevir1`
Mel Size	64, 128
Frames	32, 64, 96

Table 4.3: Experiment 1: Top 5 Generalized Configurations (Short)

Rank	Model	Ver	Stacks	Mels	Frames	$F1_{target}$(std)	$F1_{other}$(std)	$F1_{background}$(std)
1	miniresnet	v2	1	128	64	0.655(0.043)	0.411(0.069)	0.741(0.080)
2	miniresnet	v1	1	128	96	0.645(0.044)	0.419(0.030)	0.846(0.021)
3	miniresnet	v1	1	128	64	0.640(0.012)	0.472(0.066)	0.794(0.074)
4	miniresnet	v1	2	128	64	0.608(0.039)	0.393(0.071)	0.705(0.118)
5	miniresnet	v2	1	128	96	0.590(0.030)	0.419(0.031)	0.780(0.039)

ResNetV1 and V2. It seems only logical to use these as a baseline, especially since previous research showed that ResNet works very well for AED. Both V1 and V2 come in two versions: one stack and two stacks. The Mini ResNet models are based on the ResNet18 model found in TensorFlow but they are optimized for on-device performance. The consist of blocks which are assembled together to from stacks. More stacks result in a larger network which can result to higher performance at the cost of increased model size and longer inference time. The main difference between Mini ResNetV1 and V2 is the TensorFlow model that they are based on. With V1 being based on the ResNet backbone and V2 on the ResNetV2 backbone. For this experiment, four combinations are trained on the three species that are most common in the soundscapes. These models will then be compared based on the metrics in section 3.4. Table 4.2 shows the parameters that are used for this experiment. Each combination was run five times with different train/test splits and the F1 scores were averaged over the runs.

Table 4.2 shows the 5 configurations that performed the best across all the species on the short audio recordings. The 128 Mels versions outperform the 64 Mels versions. The 64 and 96 frames versions both outperform the 32 frames versions and there does not seem to be a clear advantage to one of them. It does seem that the 96 frames version slightly outperforms the 64 frames version in terms of the $F1_{background}$ score. However, the 96 frames version requires more runtime memory and computation, meaning that it might be more beneficial to continue with the 64 frame version on the embedded platform. The MiniResNet V2 with 1 stack seems to outperform the MiniResnet V1 model with 1 stack, however, the difference is not significant. The top three configurations all use 1 stack, showcasing that this works better than 2 stacks. In addition to slightly higher accuracies, the 1 stack models also require less memory at runtime in comparison to the 2 stack models. Lastly, Table 4.3 showcases a very low standard deviation across the board, meaning that the results are very consistent across the different runs.

The soundscape evaluations results differ greatly from the short audio clips. Table 4.4 shows a much lower accuracy score across the board with the best performing model only reaching an F1 score of 0.164 for the target class. Additionally, the other and background class scores have also significantly dropped when compared to the short audio clips. This shows that the model is currently unable to accurately

Table 4.4: Experiment 1: Top 5 Generalized Configurations (Soundscapes)

Rank	Model	Ver	Stacks	Mels	Frames	$F1_{target}$(std)	$F1_{other}$(std)	$F1_{background}$(std)
1	miniresnet	v1	1	64	32	0.164(0.059)	0.229(0.099)	0.617(0.075)
2	miniresnet	v2	1	128	64	0.141(0.011)	0.337(0.035)	0.429(0.138)
3	miniresnet	v1	2	128	96	0.125(0.031)	0.227(0.079)	0.588(0.037)
4	miniresnet	v1	2	128	32	0.124(0.072)	0.232(0.104)	0.591(0.059)
5	miniresnet	v2	1	128	32	0.120(0.060)	0.289(0.041)	0.479(0.035)

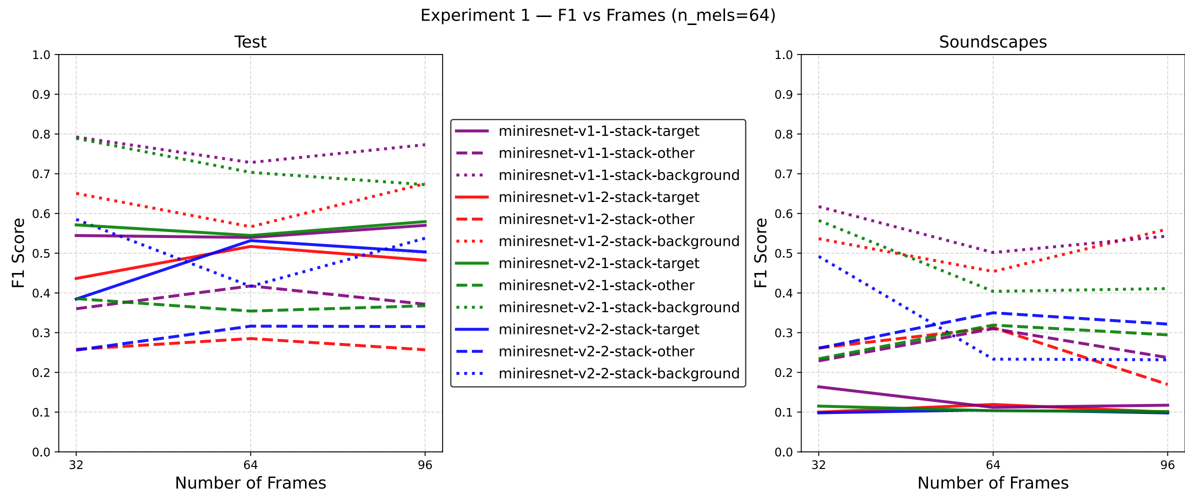


Figure 4.5: F1 scores for the `target` and `other` classes on the test audio recordings from experiment 1. All models used a Mel band count of 64. The models were trained and evaluated on the `rucwar`, `bobfly1` and `reevir` species of the BirdCLEF2021 dataset with a 80%/10%/10% train/test/val split, averaged over 5 repetitions, each having different splits. A higher F1 score means a better accuracy.

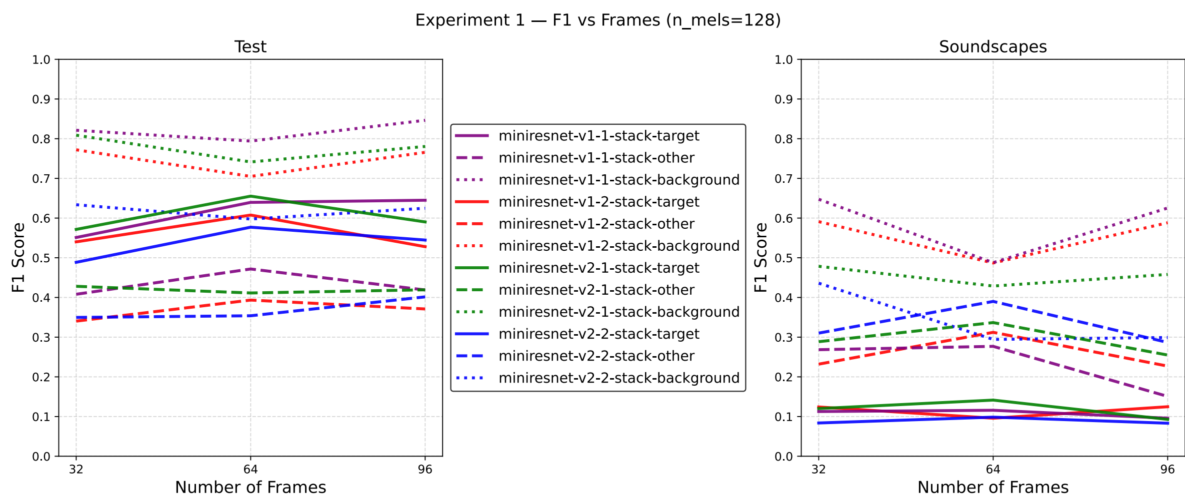


Figure 4.6: F1 scores for the `target` and `other` classes on the test audio recordings from experiment 1. All models used a Mel band count of 128. The models were trained and evaluated on the `rucwar`, `bobfly1` and `reevir` species of the BirdCLEF2021 dataset with a 80%/10%/10% train/test split, averaged over 5 repetitions, each having different splits. A higher F1 score means a better accuracy.

generalize to the soundscapes. The results show that the model version, number of stacks and number of Mels do not seem to impact the accuracy. The top 5 does contain the same model configurations as the short audio clips with the V1 and V2 models with 1 stack again at the top.

An important difference to note is that the system performs way better on the test samples than the soundscapes across all three classes. One possibility that comes to mind is the way that the soundscapes are processed before entering the model. The soundscapes are 600 seconds long and are reduced into segments of 5 seconds before entering the zoo. They are split up into segments of 3 seconds without looking at the contents, meaning that bird calls might be cut off. In order to evaluate if this is an issue, a possibility is to split the soundscapes up with an overlap, which is investigated in the next experiment.

To summarize, the MiniResNet V1 and V2 models with 1 stack and 128 Mels perform the best across all species for the short audio clips. The number of frames currently does not seem to impact the accuracy on the soundscapes which is likely due to the low data quality of the soundscapes. The next experiment attempts to improve the model performance on soundscapes by improving the way that the soundscapes are processed before entering the model.

Experiment 2: Better Soundscape Processing

This experiment investigates a different way to process the soundscapes before entering the model. The first experiment showed that the model performs significantly worse on the soundscapes than on the short audio clips. In the current pipeline, the soundscapes are cut into 5 second segments without looking at the recording. These 5 second segments are further divided into small chunks based on the frame length before entering the model. For each chunk, a prediction is made and these predictions are combined into a prediction for the 5-second segment in the form of a majority vote. Since the segments are created without any meaningful windowing function, important temporal information might be lost. If a bird call is exactly on the boundary between two segments, it might not be recognized. In an attempt to mitigate this, the soundscapes is divided into segments with a 50% overlap. The first segment ranges from 0 to 5 seconds, the second from 2.5 to 5.0 and so on. This will add new soundscapes, which ensures that every call is fully present for at least one of the segments, increasing the accuracy of the model. The main challenge was how to determine the labels of the new soundscapes. These labels are generated by selecting the overlapping labels from the original segments. If there is no overlap in the labels, the soundscape is labeled as background noise. This crude solution is more of a heuristic and has the risk of falsely labeling a soundscape but it is the best solution that can be used without having to manually label all the soundscapes. Experiment 1 showed that the 128 Mels configurations with either 64 or 96 frames performed the best on the short audio clips while also being present in the top 5 for the soundscapes. In order to save time, the 32 frames configurations are dropped for this experiment, resulting in the parameter table shown in Table 4.5.

Table 4.5: Parameters used for experiment 2.

Parameter	Values
Stacks	1, 2
Species	`rucwar`, `bobfly1`, `reevir1`
Mel Size	128
Frames	64, 96

Figure 4.7 and Table 4.6 show that the results have decreased significantly with this method. The $F1_{target}$ score dropped across the board. Figure 4.8b shows the confusion matrix of one of the configurations.

Table 4.6: Experiment 2: Top 5 Generalized Configurations (Soundscapes)

Rank	Model	Ver	Stacks	Mels	Frames	$F1_{target}$(std)	$F1_{other}$(std)	$F1_{background}$(std)
1	miniresnet	v2	2	128	64	0.135(0.018)	0.300(0.056)	0.379(0.104)
2	miniresnet	v1	2	128	96	0.123(0.010)	0.241(0.083)	0.542(0.109)
3	miniresnet	v1	2	128	64	0.114(0.047)	0.277(0.036)	0.543(0.093)
4	miniresnet	v1	1	128	64	0.113(0.043)	0.278(0.053)	0.538(0.082)
5	miniresnet	v1	1	128	96	0.096(0.021)	0.244(0.062)	0.561(0.081)

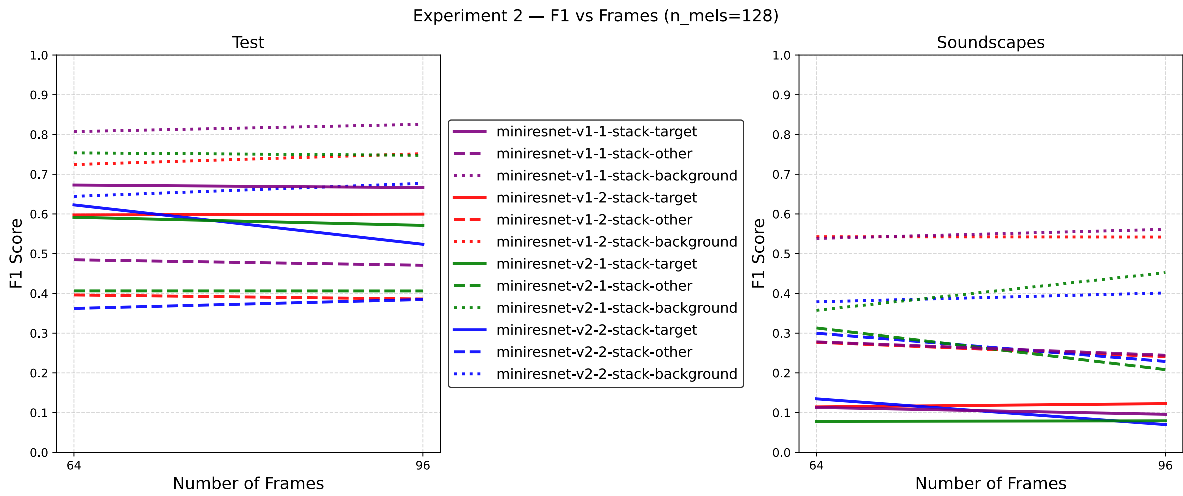
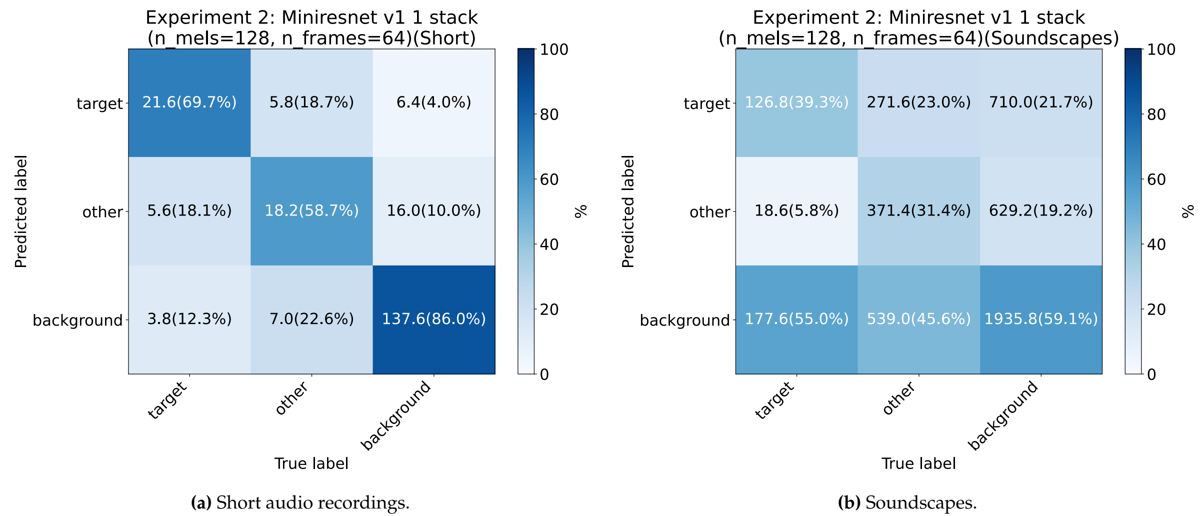


Figure 4.7: F1 scores for the target and other classes on the soundscape audio recordings from experiment 2. All models used a Mel band count of 128. The models were trained and evaluated on the `rucwar`, `bobfly1` and `reevir` species of the BirdCLEF2021 dataset with a 20% train/test split, averaged over 5 repetitions with different splits. A higher F1 score means a better accuracy.



(a) Short audio recordings.

(b) Soundscapes.

Figure 4.8: Confusion matrices for experiment 2, configuration n_mels=128, n_frames=96.

It shows that the model predominately labels the windows as `background` and has a difficult time correctly recognizing the `target` class. When comparing this with Figure 4.8a, it shows that the model is significantly better at classification on the short audio with a higher accuracy for every class, with the main bottleneck being accurately distinguishing between the `other` and `background` classes. The cross validation matrices for the other experiments are shown in Appendix B.

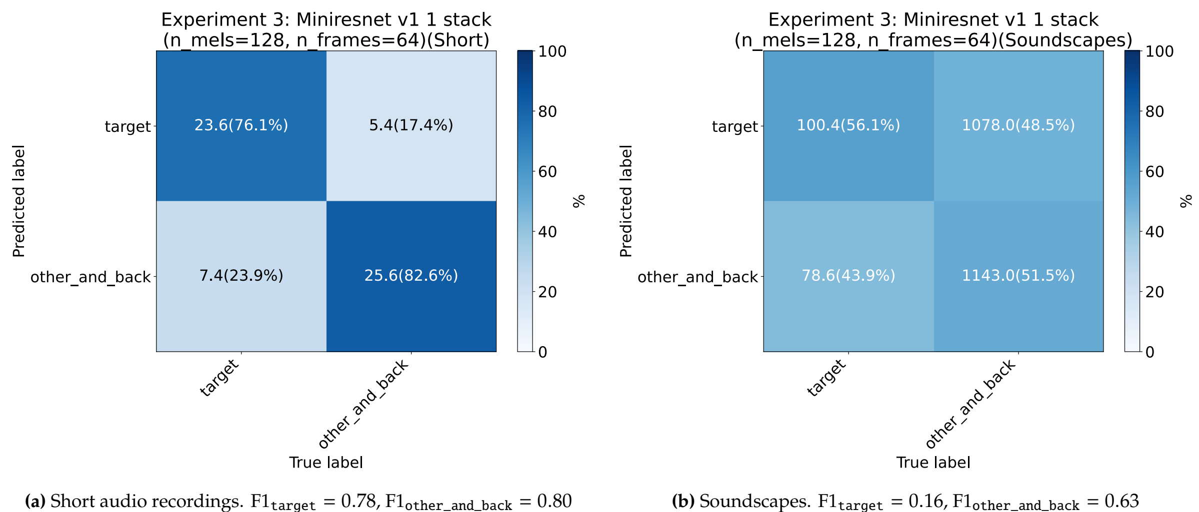
The results from this experiment show that segmenting the soundscapes with overlap is not the solution and instead makes the problem worse. The results of the short audio recordings show that the model has a difficult time distinguishing between the `other` and `background` and classes, which might propagate to the soundscape results. One approach to mitigate this is to group the `other` and `background` classes into a single class. This could help the model focus on recognizing the specific characteristics of the `target` class' sound signature and classify the rest as other sounds, regardless if it is a different species or background noise. As this is relatively easy to achieve, the next experiment investigates if this helps the model's performance.

Experiment 3: Binary Classification

The goal of this experiment is to increase the accuracy on the soundscapes by changing from a multi-label classification problem into a binary classification problem. A new class `other_and_back` is created by merging the `other` and `background` classes into a single class. The expected benefit is that the model is more focussed on recognizing the specific temporal and frequency components of the target sound signature and therefore makes less false `background` predictions. This third class was achieved by simply replacing all the `other` and `background` labels with the `other_and_back` label and rerunning the experiments described in experiment 2 for the `rucwar` class.

Table 4.7: Used parameters for experiment 3.

Parameter	Values
Stacks	1
Species	`rucwar`
Mel Size	128
Frames	64



(a) Short audio recordings. $F1_{\texttt{target}} = 0.78$, $F1_{\texttt{other_and_back}} = 0.80$

(b) Soundscapes. $F1_{\texttt{target}} = 0.16$, $F1_{\texttt{other_and_back}} = 0.63$

Figure 4.9: Confusion matrices for experiment 3, configuration n_mels=128, n_frames=64.

Figure 4.9a and Figure 4.9b show the results from changing from three classes to two classes. The accuracy of the model improves slightly, predicting 56.1% of the `target` class correctly. However, the increase in accuracy is not sufficient and the overall F1 scores are still too low.

Based on these results, a manual investigation on the soundscapes was performed. This showed that many of the 5-second segments only contained a single, short bird call which is not enough for the

window to be labeled as a bird call by the majority vote. This investigation also showed that the soundscapes in the BirdCLEF2021 dataset are amplified to such an extent that noises such as microphone static are louder than the actual bird calls. In addition, many of the labeled 5-second segments did not contain an audible bird call.

All in all, the first three experiments show that different configurations or introducing overlap in the segmentation are not the solution. In hindsight, the conclusions taken from experiment 1 with regard to the issues with the soundscapes were wrong. In reality, the weakly labeled nature and bad quality of the soundscapes require a different approach in order to better train and evaluate the model. To achieve this, a completely new approach was needed, building on the knowledge gained from the first experiments and especially the manual soundscape investigation. This approach focuses on improving the quality of soundscapes by ensuring that the labels are actually true and improves the quality of the audio by filtering out the background noise. The experiments did show that a binary classifier works better than the three class model, which also benefits the final deployment in terms of simplicity and resource requirements.

4.6. A New Framework

Up to this point, Python scripts were used as a wrapper around the STM32 AI ModelZoo which allowed for automatic training, testing and evaluation. It was not possible to easily modify the ModelZoo to accommodate for the new required approach and thus it was decided to drop the STM32 AI ModelZoo and create a custom framework. A big benefit of creating the custom framework is more control over the individual pipeline stages and a deeper level of integration resulting in faster experimentation. The first version of this framework was created using Jupyter Notebooks running on WSL2. While this increased the performance significantly by caching the datasets and using the TensorFlow Data API [67], it ran into reliability issues due to running on WSL2. Using the ModelZoo, each iteration took 13.5 minutes. In order to ensure consistency in the results, each run would have to be repeated 5 times, taking around 1 hour and 10 minutes for each configuration. With this second version, it only takes 6 minutes to run 10 different iterations. However, the reliability issues made this Jupyter Notebook based approach unfeasible. Based on all the knowledge and insights gained from creating each pipeline stage, I decided to rewrite the framework a final time using Python.

The rest of this section describes the key components of the final version of the custom framework. The goal of this final version was to decrease the runtime per experiment even further and investigate numerous parameter configurations in an automated way. The combination of these two factors ensures that the framework can be run on large datasets and automatically optimize for different species. It also gives the opportunity for investigating different preprocessing methods such as comparing log-Mel spectrograms and PCEN spectrograms which was previously not possible with the ModelZoo. Lastly, it also solves the key issues with the soundscape recordings and generalizes the framework to new datasets by using BirdNET, a state-of-the-art bird classification model.

Label Generation

The most important insight from the first version regards the BirdCLEF dataset and more importantly the main issue which resulted in the inaccuracies with the soundscape recordings. Not only is the soundscape data weakly labeled, the training recordings are as well. To mitigate this issue, new labels are generated using BirdNET [1]. BirdNET analyzes each of the training recordings in segments of 3 seconds and returns all the birds that are detected in that segment. In order to increase the training data even more, an overlap of 1.5 seconds was used. This means that segment 1 spans from 0 to 3 seconds, segment 2 from 1.5 to 4.5 seconds and so forth. Not only does this increase the training data, it also helps find the bird calls that are cut off at the segment's edges.

This new way of generating labels does bring along a new constraint for the system. The labels generated by BirdNET have a granularity of 3 seconds. Previous experiments showed that if this audio fragments get segmented into smaller chunks before entering the model, they might get misclassified. In order to prevent these errors, the input for the model was set to 3 seconds, effectively fixing the n_frames parameter from the previous experiments. In order to calculate the amount of frames based on this new 3 second audio duration, Equation 4.4 is used.

$$F = 1 + \left\lfloor \frac{S * f_s}{R} \right\rfloor, \quad (4.4)$$

where F denotes the amount of frames, S denotes the amount of seconds of audio, f_s the sample rate and R the hop length. In this case, S is always set to 3 seconds. This ensures that regardless of the different parameters, the model is always fed 3 seconds of audio, meaning that the labels match the BirdNET predictions. Lastly, to guard the quality of the labels, a minimum confidence level of 90% is used for the BirdNET predictions.

When comparing the new labels with those from the BirdCLEF competition showed that some recordings had such a low quality, that even a much larger, state-of-the-art model such as BirdNET could not detect the labeled bird. Based on this, it was decided to drop the soundscapes as the representative test set. The quality of these recordings were of such a low quality that they were not representative of the ChirpAlert platform, which produces recordings with fewer artifacts and a more clear signal overall. Continuing to work on improving the soundscape accuracy would mean optimizing for a non-representative use case. In order to validate this theory, multiple soundscapes of a higher quality were downloaded from Xeno-Canto and run through the current state of the model. This showed that the current state of the model closely matched the predictions from BirdNET, further reinforcing the decision to drop the soundscapes from BirdCLEF.

New Metric

In the previous experiments, the key metric was the F1 score of the `target` class. However, this did not fully match up with the end goal of the model. The main goal of the system is to be able to recognize all bird calls of a target species while minimizing the number of false positives. This means that the model must have a high recall while also having a high precision. While this does get captured by the F1 score, precision and recall are weighed evenly. In this use case, the recall is more important than the precision, in other words, false negatives are worse than false positives. It would be better to misclassify a small amount of bird calls if that meant that all bird calls of the `target` class are found. The wrong classifications can always be filtered out when reviewing the data after the deployment. In contrast, classifying all calls correctly at the cost of missing a few, is worse, because those calls are lost forever. From this point, the Recall at Precision (RP) metric, explained in section 3.4 was used with a 90% precision level as this better matches the goal. During the training process, 200 thresholds are used to calculate the precision and recall values. The thresholds are calculated by taking 200 equally spaced fractions between 0 and 1.

Hyperparameter Optimization

The reduced time per experiment combined with more fine control over the preprocessing pipeline resulted in a lot of variables that can be tuned. To find the optimal configuration, a hyperparameter optimization was performed using the Keras HyperBand Tuner [68]. The main benefit of using a HyperBand tuner is that it doesn't have to search the entire hyperparameter space. The main idea is broken down into three ideas: Try lots of settings briefly, in this case by limiting the amount of epochs. Keep only the best fraction and repeat but with an increased amount of epochs. This process is repeated until the maximum amount of epochs is reached. For this research, an expansion of the HyperBand tuner is used called a Data Aware HyperBand tuner. This tuner does not only optimize the model hyperparameters but also the data preprocessing parameters. In this case, this allowed for evaluation of different spectrogram and data augmentation configurations. Table C.1, Table C.2, Table C.3 and Table C.4 in Appendix C show the full hyperparameter space that was optimized. It is important to note that a single configuration might not work for every sound signature and the process should be rerun every time a new target or dataset is selected for the best results. For example, if a signature has high frequency components that are filtered by the configured band pass filter, the model will not be able to recognize them.

One of the key parameters is the addition of a sample rate of 32 kHz. As explained in section 4.3, the sample rate is an important parameter to consider both in terms of performance and hardware requirements. Based on investigation done in the previous iteration of this framework, setting this to a fixed value of 16 kHz could have been a mistake. Due to the performance increase of the framework, examining different sampling rates and subsequent parameter changes such as the hop length is now an

Table 4.8: The configuration used during the learning rate experiments. This configuration was retrieved from the first full hyperparameter optimization run on the `rucwar` class of the BirdCLEF2021 dataset.

Parameter	Value	Paramter	Value
Experiment			
Name	Learning Rate	Target species	*rucwar*
Random state	51		
Dataset			
Dataset used	birdclef	Min. samples per class	50
CV splits	5	Pos./neg. ratio	3
Audio parameters			
Butterworth order	7	Sampling rate [Hz]	32 000
Frames per sample	241	Mel bands (n_{mels})	80
FFT size (n_{fft})	1024	Hop length	400
Window function	hamming		
Power exponent	2.0	Frequency range [Hz]	500–10 000
Normalization	Slaney	Segment duration [s]	3.0
PCEN enabled	False	PCEN (t_c, g, b, p)	(0.400, 0.98, 2.0, 0.5)
PCEN ε	1×10^{-6}		
Augmentations			
Band pass range [Hz]	500–10 000		
p_{loud}	0.8	Loudness range [dB]	[-27, -23]
p_{gaus}	0.1	Gaussian SNR [dB]	16–28
p_{spec}	0.0	SpecAugment (freq,time)	(2,2) widths (16,32)
Model & training			
Batch size	32	Epochs per fold	100
Loss function	FOCAL	Mixup enabled	True ($\alpha = 0.4$, $p = 1.0$)

option. In comparison, BirdNET uses a sample rate of 32 kHz and a band pass filter of 15 kHz during its feature generation step. Based on this insight, a 32 kHz sample rate is added to the parameter list for the hyperparameter optimization. Not only does this result in more frequency information for the model, it also ensures that the framework works for any species of bird, with the Hummingbird having the highest call at 11.8 kHz.

Learning Rate Schedulers

An important part of training a ML model is controlling its learning rate. As explained in Figure 3.4, the learning rate is an important factor in controlling how much the weights of the model are updated. For this framework, three commonly used techniques were evaluated: Reduce On Plateau (ROP), Delayed Reduce On Plateau (DROP) and cosine decay. ROP is a technique that reduces the learning rate by a fixed factor when the target metric for a model has stopped improving, which helps smooth out the model's convergence. DROP is a modified version of ROP that only starts monitoring the metric after a set number of epochs, delaying the first learning rate reduction. The first part of training can be very noisy since the model does not have a clue what it is looking for. By delaying the start of ROP, premature reducing of the learning rate is prevented. Lastly, cosine decay [69] is a technique that reduces the learning rate based on time instead of looking at metrics. This can be beneficial since this also allows for a warm-up period where the learning rate is first slowly increased before it starts to drop again. The main downside is that this is a fixed schedule which doesn't respond to the model performance, which is a big benefit for both ROP and DROP. Table 4.9 shows that ROP outperforms both DROP and cosine decay in terms of the accuracy of the `target` class. Based on these results, ROP is used for the framework, however all three options are supported. The configuration used during this small experiment is shown in Table 4.8. This configuration was determined based on a hyperparameter optimization run on the full BirdCLEF dataset for the `rucwar` class. It is important to note that this was performed on the BirdCLEF dataset using the labels generated by BirdNET. The RP90 metric is not reported as this process used the BirdCLEF soundscapes and a precision of 90% was not reached.

Table 4.9: Learning rate experiments performed using the BirdCLEF2021 dataset with labels generated by BirdNET V2.4.

Scheduler	Class	Max Precision (Recall)	Max Recall (Precision)	F1
No Scheduler	0	0.742 (0.224)	0.786 (0.707)	0.745
	1	0.248 (0.766)	0.796 (0.243)	0.375
ROP	0	0.749 (0.990)	0.990 (0.749)	0.852
	1	0.237 (0.881)	0.881 (0.237)	0.374
DROP	0	0.746 (0.973)	0.973 (0.746)	0.844
	1	0.228 (0.670)	0.670 (0.228)	0.340
Cosine Decay	0	0.742 (0.949)	0.949 (0.742)	0.833
	1	0.240 (0.817)	0.817 (0.240)	0.371

Dataset Generation

For the final system evaluation, a new dataset is required that contains the target species as well as commonly co-occuring species. The goal is to train the model on all the species that it will encounter to ensure that it doesn't misclassify them as `target`. To facilitate this constraint as well as give the framework the flexibility to work on any species recorded on GBIF, the framework was expanded with automatic dataset generation. This feature automatically queries GBIF and Xeno-Canto for the required data. Given a target species and a target region, the script queries GBIF for the most commonly co-occuring species and based on these species, the script queries Xeno-Canto for audio recordings. All recordings are classified with a quality metric, with A being the highest and E the lowest. For the target species, all recordings up to a configurable quality level are downloaded and processed through BirdNET for labels. As seen in the experiments, this helps ensure that the quality of the training model is high. For the co-occuring species, a configurable amount of recordings are downloaded. These slots first get filled up with the highest quality recordings and then the quality slowly gets reduced to fill the remaining slots. One of the main benefits of this approach is that this can be done in a fully automated manner for any species that has occurrence data on GBIF and accompanying recordings on Xeno-Canto. This means that the system can easily be trained on different species in different regions.

The dataset used for the deployment contains 60 different species, including the Pica pica (Ekster). A total of 8,005 recordings are used, of which 1,440 were of the target species. From those recordings, a total of 81352 labels were extracted using BirdNET version 2.4. Of those labels, 14,066 were of the target species. This final dataset was then split into training, validation and test sets with a split of 80%, 10%, and 10% respectively. It is important to note that the splits were done with a stratified approach, meaning that each split preserves the ratio of species. This helps ensure that the model is trained and evaluated on a representative dataset. Additionally, each split has a positive/negative ratio of approximately 1:3 to ensure that the model isn't flooded with negative samples. The final counts of the dataset can be found in Table 4.10. The full species list is available in Appendix D.

Set	Total Labels	Positive Labels	Negative Labels	Used
Training	65,081	11,253	53,828	45,012
Validation	8,135	1,406	6,729	5,624
Test	8,136	1,407	6,729	5,628

Table 4.10: Final dataset specifications

To optimize the configurations of the framework and the model, a hyperparameter optimization was performed on the final dataset. For this process, only a 10%, stratified split of the full dataset was used. This means that the original class distribution was preserved, ensuring that all species are represented in the split. This subset was further divided into an 80%/10%/10% train/val/test split.

Table 4.11 shows the results of the best performing configuration with a very high RP90 of 0.9857. However, in order to reach this score, the most expensive form the model is used. Not only does it use the full 3 MiniResNet stacks, it also uses PCEN and a high amount of Mels. In addition, Table 4.11 also shows a couple of other interesting parameters. The Butterworth order is set to 0, meaning that there is no band pass filter in use. The frequency range of the spectrograms is also set to a very wide range of

Table 4.11: Trial 0181 configuration summary with RP90 score. These results were attained using 10% of the Pica pica dataset.

Parameter	Value	Parameter	Value
Experiment			
Name	Trial 0181	Target species	`Pica pica`
Random state	51	RP90	0.9857
Dataset			
Dataset used	10% of custom Dataset	Min. samples per class	50
Pos./neg. ratio	1		
Audio parameters			
Sampling rate [Hz]	32000	Mel bands (n_{mels})	96
FFT size (n_{fft})	512	Hop length	288
Window function	hamming	Normalization	Slaney
Butterworth order	0	Frequency range [Hz]	400–14999
Segment duration [s]	3	Frames per sample	334
PCEN enabled	True	PCEN (t_c, g, b, p)	(0.45, 0.95, 2.75, 0.4)
Augmentations			
p_{loud}	0.2	Loudness range [dB]	[−36, −18]
p_{gaus}	0.0	Gaussian SNR [dB]	10–15
p_{spec}	0.0	SpecAugment (freq,time) masks	(0, 2)
Freq/time mask widths	(58, 18)	Band pass range [Hz]	400–8499
Mixup	True ($\alpha = 0.3, p = 0.4$)		
Model & training			
Batch size	16	Epochs per fold	40
Loss function	FOCAL ($\gamma = 2$, $\alpha = 0.2$)	Initial learning rate	0.001
LR scheduler	DROP	Regularization	$L_2 = 10^{-4}$, dropout = 0.0
Stacks	3	Pooling	avg

400 Hz to 15 kHz. Since there is no band pass filter, the frequencies outside this range can still influence the results. This is something to keep in mind for the final deployment. For example, if the model performs bad in noisy environments such as wind, the band pass filter should be reintroduced.

Two other important parameters are the probabilities of the Gaussian and SpecAugment augmentations. Both are set to 0.0, meaning that they are disabled during training. This could be explained by the high quality recordings as both augmentations improve the robustness of the model against artifacts in the audio input, which are not present in the training or evaluation data. This should not be an issue as the audio pipeline of the ChirpAlert platform produces high quality recordings, but this is something to keep in mind. If the accuracy of the system drops significantly in windy or noisy conditions, this should be reintroduced to make the model more robust against it. While this can sacrifice model performance in ideal conditions, it will help in non-ideal conditions.

Lastly, Table 4.11 shows that the model performs best with the Focal loss function, explained in Figure 3.4.

4.7. On-Device optimizations

In order to determine the accuracy trade-off of reducing the size of the model as well as tweaking other parameters, a small mini experiment was run using the full Pica pica dataset. While the previous experiments used cross-validation, these experiments did not due to the size of the dataset. The main idea of these experiments was to get a general idea of the effect of changing different components to help estimate the accuracy trade-off between the configurations. As the dataset is quite large and the same splits are used across all experiments, the only non-determinism can come from the model itself. The goal of these experiments to get a general idea if any of the changes significantly impacted performance

and thus this small margin of error was deemed acceptable.

Manual Tuning

Figure 4.10 shows the experiments that were performed including their RP90 results. Only the mentioned parameters were changed and the rest remained the same as with trial 181, shown in Table 4.11. Figure 4.10 shows that switching to log-Mel spectrograms and 1 stack configurations only results in a 0.3% drop in performance. The combination of going to 80 Mels with 1 stack, no PCEN and no Gaussian noise seems to perform very well with only an accuracy drop of 0.4% when compared to the baseline, which is very acceptable keeping in mind the resources saved. Since the 2 stack combination does work better, it is worthwhile to evaluate both models in terms of resource requirements on the ChirpAlert platform. Lastly, the hop length will need to be evaluated in terms of resource requirements. Figure 4.10 shows that the results across hop lengths is somewhat inconsistent. A lower hop length means more detail in the spectrogram and thus it is to be expected that a higher hop length would always decrease the accuracy. However, Figure 4.10 shows that this is not the case with a hop length of 288 (No_Gaus_1) outperforming Hop_400. Since the difference is only 0.04%, this could also be a small amount of non-determinism in between runs. In order to verify the results, the five rightmost models were retrained and the experiment was performed again and averaged. This resulted in the exact same values. Since the differences between the accuracy of the frame lengths is small, the hop length is evaluated based on the resource requirements on the ChirpAlert platform.

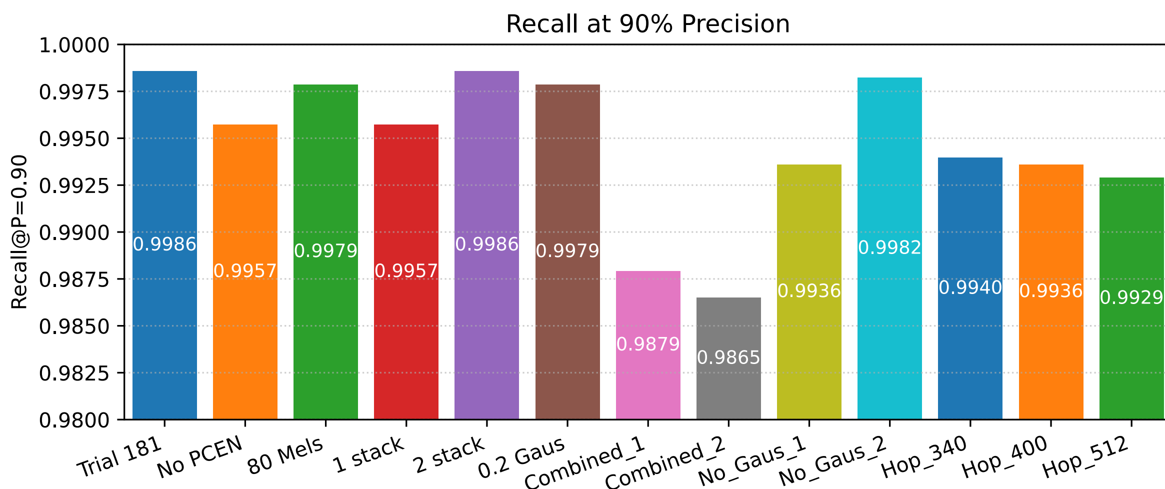


Figure 4.10: Recall at 90% precision per manually tuned variant. Trail 181 was the best performer of the hyperparameter optimization is and is used as the baseline. The five rightmost experiments were run twice and averaged in order to verify their results. Please refer to Table 4.12 for more information regarding the configurations.

The next section will compare the top contenders in terms of resource requirements on the ChipAlert platform.

Quantized Model Performance

In order to select the best model for the embedded platform, the top contending models are converted into TFLite format. As explained in Figure 3.5, TFLite micro allows the conversion of TensorFlow models into a format that can be run on microcontrollers. During this process, some accuracy gets lost which might expose significant differences between the current contenders. The TFLite converter is configured to keep the models inputs and outputs as floating point values. While it is common to only use integer values on microcontrollers, the spectrogram generation pipeline on the embedded platform uses floating point values. In order to prevent unnecessary conversion, the model will take in floating point values while the internal operations are in integer format. The first and last layers of the model will therefore be responsible for (de-)quantization of the input values.

For this research, the calibration process of the (de-)quantization layers, explained in Figure 3.5, is done with the validation split of the training data. Since the splits are stratified, this is a good representation

Table 4.12: Recall at 90% precision per manually tuned variant. Trail 181 was the best performer of the hyperparameter optimization is and is used as the baseline.

Name	Mels	Stacks	PCEN	Gaussian Probability	Gaussian Range	Hop Length
Trail 181	96	3	Yes	0.0	[10, 15]	288
No PCEN	96	3	No	0.0	[10, 15]	288
80 Mels	80	3	Yes	0.0	[10, 15]	288
1 Stack	96	1	Yes	0.0	[10, 15]	288
2 Stacks	96	2	Yes	0.0	[10, 15]	288
0.2 Gaus	96	3	No	0.2	[10, 15]	288
Combined_1	80	1	No	0.2	[10, 15]	288
Combined_2	80	1	No	0.2	[1, 10]	288
No_Gaus_1	80	1	No	0.0	[10, 15]	288
No_Gaus_2	80	2	No	0.0	[10, 15]	288
Hop_340	80	2	No	0.0	[10, 15]	340
Hop_400	80	2	No	0.0	[10, 15]	400
Hop_512	80	2	No	0.0	[10, 15]	512

Table 4.13: RP90 scores for both the full scale and converted TFLite model on the test split of the final dataset. The full scale model is used as a baseline to compare the TFLite model against in order to see the accuracy difference after conversion. For all runs, the average was taken over two runs in order to help ensure consistency.

Stacks	Hop Length	Keras	TFLite
1	288	0.9929	0.9936
2	288	0.9922	0.9922
1	340	0.9957	0.9964
1	400	0.9929	0.9922
1	512	0.9936	0.9936

of the different species in the dataset and a magnitude of different input values. As stated above, this process is embedded in the TFLite model and will not have to be handled separately on the embedded platform.

After the conversion process, the model is evaluated using the test split of the dataset. The same split is passed through the full scale floating point model in order to compare the performance of the TFLite model. In order to ensure an as fair as a comparison as possible, the full scale models were also evaluated on the exact same test split in the same environment. For every configuration, the two full scale models from section 4.7 were converted and evaluated. Table 4.13 shows the results of the contending models. It shows that the TFLite model closely matches the full scale model and that the accuracy drop is minimal. In some cases, the TFLite model even performs better than the full scale model. It shows that the 340 hop length model performs the best, closely followed by the 340 and 512 models. The 2 stack model doesn't outperform the 1 stack models, meaning that the 2 stack model is dropped from further consideration. Since all the configurations are very close in terms of performance, the final hop length is determined based on the spectrogram generation and inference timings.

Table 4.14 compares the different hop length configurations and their respective performance and resource requirements on the ChirpAlert platform. This shows that by decreasing the hop length, a

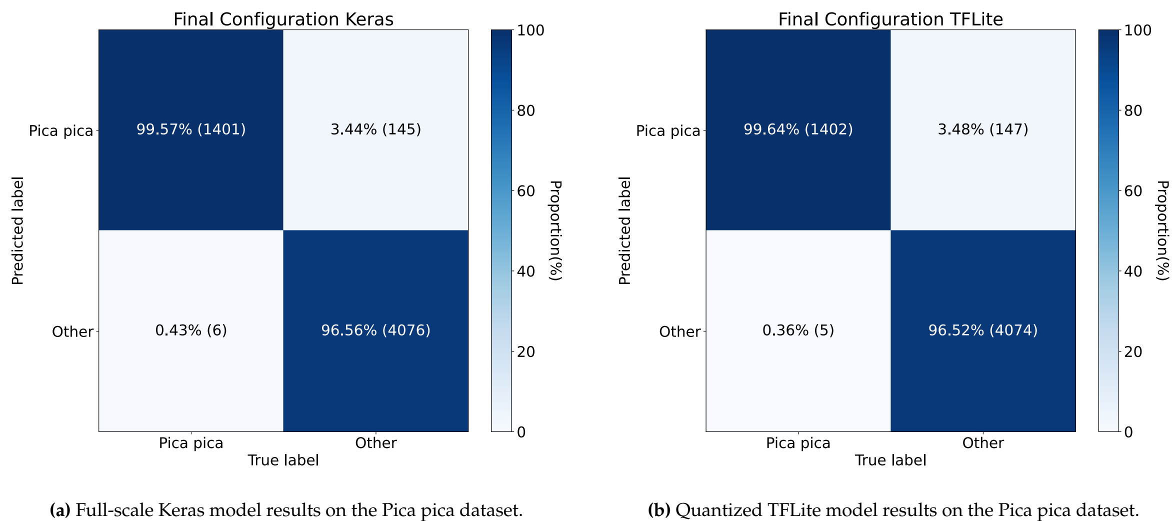
Table 4.14: Model resource requirements and on-device performance, reported by the model analysis in STM32CubeIDE V1.19.0.

Hop Length	288	340	400	512
Multiply-Accumulates (MACs)	58.2M	49.7M	42.6M	33.1M
Flash (KiB)	141.56	141.55	141.55	141.55
RAM (KiB)	166.93	142.32	121.97	96.21
Spectrogram (ms)	312	264	224	173
Inference (ms)	734	629	545	425

significant amount of time can be saved. Not only on the spectrogram generation but on the model run-time itself. By combining Table 4.13 and Table 4.14, a hop length of 340 was selected for the final configuration based on the trade-off between speed and accuracy.

Final Configuration

Figure 4.11a shows the performance of the full scale model when evaluated on the full Pica pica dataset with the optimal parameters, which are shown in Table 4.15. It is clear that the model has a very high performance and is capable of distinguishing between the `target` and `other` class with a small margin of error. The misclassifications are predominately false positives, which are preferred as they can be filtered out after the deployment has been completed. With a very low false positive rate of 0.14%, the system rarely misses a call of the target species which aligns with the end goal of the system. The RP90 score of the full scale model increases from 0.9957 to 0.9964 after converting it to TFLite format using the default optimizations and quantizing it to use integer operations. Figure 4.11b further shows that the difference between the Keras and TFLite model is small.



(a) Full-scale Keras model results on the Pica pica dataset.

(b) Quantized TFLite model results on the Pica pica dataset.

Figure 4.11: Comparison of the Keras and TFLite model confusion matrices using the parameters in Table 4.15 on the full Pica pica dataset. The Gaussian and SpecAugment augmentations were not used.

Signal Power Calculation

One of the final key elements for the embedded system is the part responsible for efficiently detecting silence. In order to prevent unnecessary computation on sound segments that don't contain any useful information, a method defined by a related work called TinyChirp [54] is used. This method can efficiently calculate the signal power P of a segment using

$$P = \frac{1}{N} \sum_{n=1}^{N} x(n)^2, \tag{4.5}$$

where $x(n)$ and N denote the data points and the length of a segment, respectively. This operation is done on the raw audio, meaning that it can be performed before the spectrogram calculation. Based on a predefined threshold, the system decides if the model should be run or not. The tuning of the threshold can not be done on the training data as this is all collected on different hardware. The actual threshold will have to be determined on the ChirpAlert platform by monitoring the signal power in quiet and noisy environments. The benefit is that this gives the end user the option to control the sensitivity of the system. The embedded implementation of this formula is done using optimized kernels from the Arm CMSIS library [70]. This helps ensure that the signal power calculation is done very efficiently. Preliminary testing showed that this is done in 4ms on the Cortex-M33 core of the ChirpAlert platform, which is very acceptable when compared with running the full spectrogram calculation.

Table 4.15: The configuration parameters used for the final system evaluation.

Parameter	Value	Parameter	Value
Experiment			
Random state	51	Target species	`Pica pica`
Dataset			
Dataset used Pos./neg. ratio	Pica pica 3	Min. samples per class	50
Audio parameters			
Sampling rate [Hz] FFT size Window function Butterworth order Segment duration [s] PCEN enabled	32000 1024 hamming 0 3 False	Mel bands Hop length Normalization Frequency range [Hz] Frames per sample PCEN (t_c, g, b, p)	80 340 Slaney 400–14999 283 (0.45, 0.95, 2.75, 0.4)
Augmentations			
p_{loud} p_{mixup}	0.2 0.4	Loudness range [dB] alpha	[−36, −18] 0.3
Model & training			
Batch size Loss function LR scheduler Stacks	16 FOCAL ($\gamma = 2$, $\alpha = 0.2$) DROP 1	Epochs per fold Initial learning rate Regularization Pooling	40 0.001 $L_2 = 10^{-4}$, dropout = 0.0 avg

5

On-Device Implementation

After determining the key configuration parameters and design challenges, the software for the ChirpAlert platform was created and the full Audio Event Detection (AED) pipeline was implemented. After implementing all components, the system was evaluated on the resource requirements, spectrogram generation and power characteristics before moving on to the field deployment.

5.1. Sensor Design

To achieve the set goals of low-power on-device audio detection, a specialized sensor platform was created. The main goal of the ChirpAlert platform is to allow standalone audio collection and processing with frequencies up to 80 kHz while being as non-invasive as possible to the ecosystem that it is deployed in. In addition to collecting audio, the platform is also equipped with multiple environmental sensors to allow for a more enriched sensor suite while also opening up different triggers for when to record audio. Figure 5.2 gives a system overview of the ChirpAlert platform, including key components and their features. It is important to note that some sensors were placed to make this sensor applicable for other projects, such as the Enact project [71], for which this sensor is also going to be used.

Analog

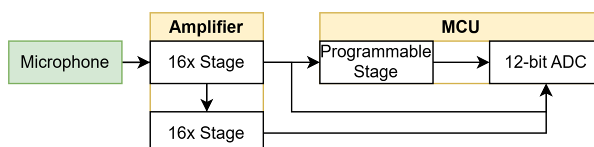


Figure 5.1: Analog system overview showcasing the different paths from the microphone to the Micro-Controller Unit (MCU).

The ChirpAlert platform features a single microphone capable of recording ultrasonic frequencies up to 80 kHz while retaining a very low power consumption of 120 μA. It was decided to go with a different microphone than the one used on the breadboard prototype as the selected microphone matches or exceeds its specifications while officially supporting frequencies up to 80 kHz. The audio signal from the microphone is passed through both channels of the operational amplifier, as shown in Figure 5.1. Both of them are configured as non-inverting amplifiers with a gain of 16x each, resulting in a total amplification of 256x or 48.2 dB. In order to make the system as configurable as possible, the first and second stage of the amplification process are connected to different pins of the MCU. This allows the system to switch between amplifications using software. In addition, the first stage is connected to the integrated amplifier of the STM32U5A9 which features programmable gain, meaning that the signal can be amplified by different factors on the MCU at run-time. The combination of a two-stage amplification process combined with high quality, low-power components and careful design gives the ChirpAlert platform a flexible analog frontend for high quality recording, using only 460 μA to get the audio to the MCU.

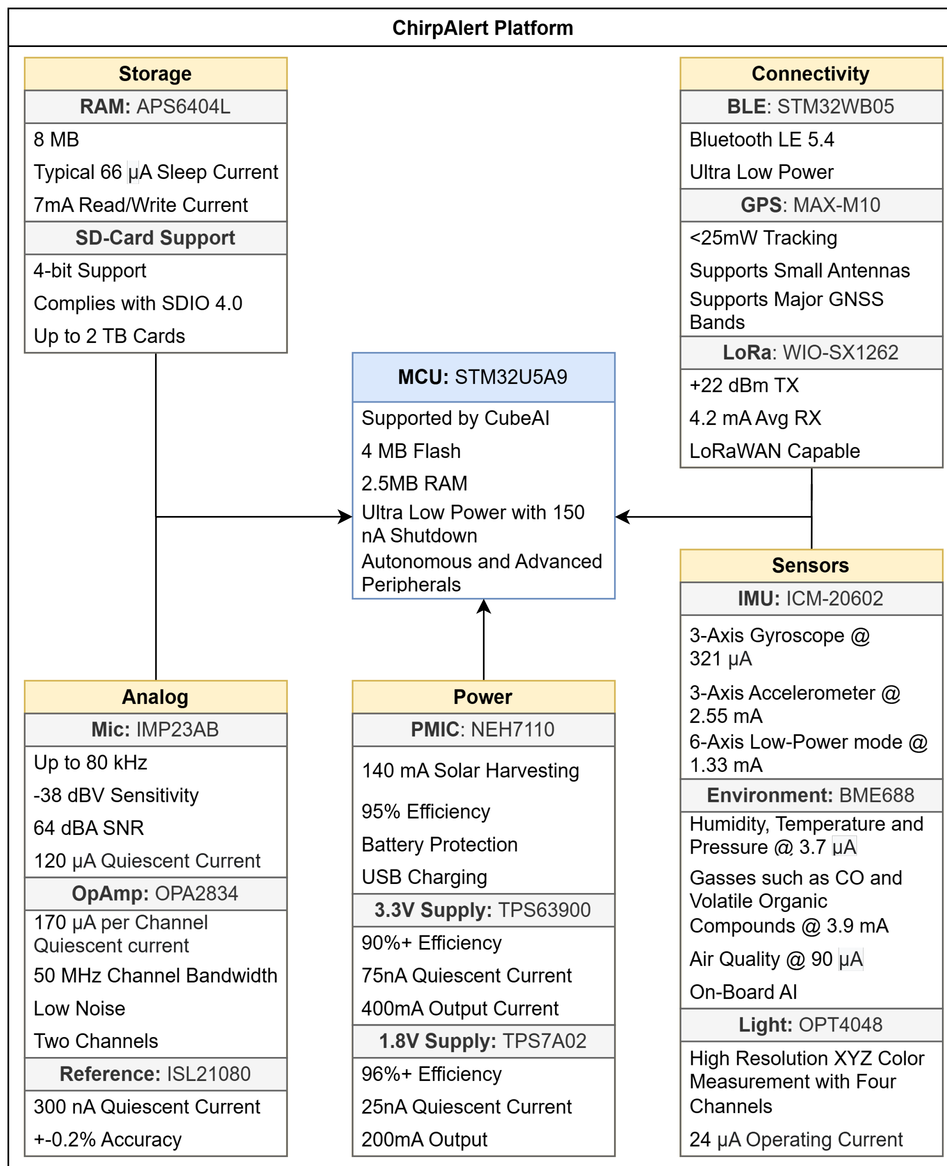
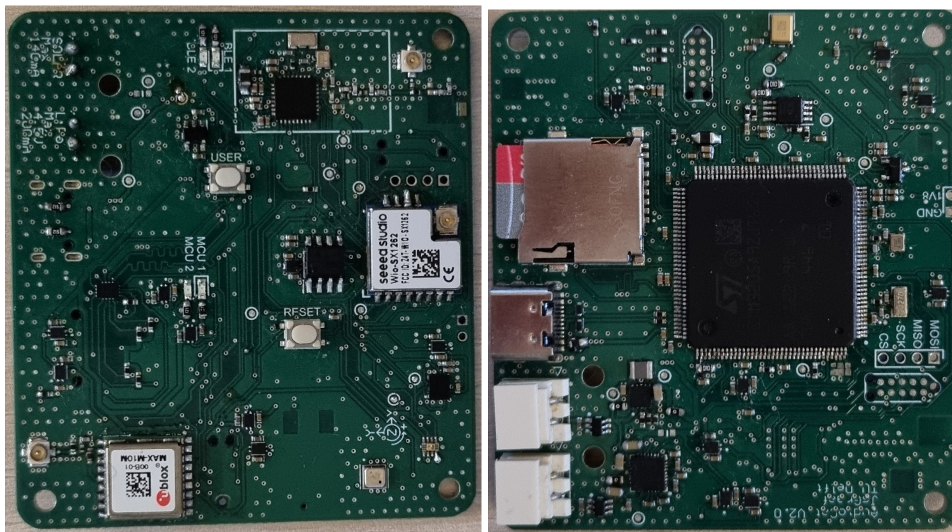


Figure 5.2: System overview of the ChirpAlert platform, highlighting the key components of the system.

Connectivity

The ChirpAlert platform also has multiple connectivity options in order to increase the deployment options. By incorporating a Long-Range (LoRa) radio, the platform is able to communicate measurements or inference results over distances of up to 15 km. An example of a use case where this would be beneficial is on-device AED. The ChirpAlert platform is able to communicate the inference results over long distances while remaining deployed, as well as communicate errors. This allows researchers to act immediately instead of having to wait for the results until after the full deployment. The selected radio supports connection to LoRaWAN, a commonly used protocol for LoRa based communication networks allowing easy integration of ChirpAlert sensors into existing infrastructure.

The Bluetooth Low Energy (BLE) radio allows for short range communication with a higher data rate than LoRa. This allows for on-site device configuration without physical access to the device and also further increases the use cases of the device. The radio fully supports the Bluetooth LE 5.4 specification meaning that the sensor platform can be configured to function in a Wireless Sensor Network (WSN). This opens up use cases such as audio based localization which are further supported by the on-board GPS. Furthermore, the presence of the GPS allows for precise geotagging of the collected data, improves the sensor retrieval and allows triggers based on geolocation. It is important to note that the BLE radio



(a) Front view

(b) Back view

Figure 5.3: Front and backside of the assembled ChirpAlert prototype.

is capable of communication with the LoRa and Global Positioning System (GPS) modules with the large MCU. This allows for full connectivity while the MCU is occupied with computationally intensive tasks such as model inference.

Sensors

Another goal of the ChirpAlert platform is to be a one-stop solution for environmental monitoring in general, not only acoustic monitoring. In order to facilitate this, a full environmental sensor suite is present. The first sensor is capable of measuring gasses such as Carbon Monoxide and Volatile Organic compounds, in addition to humidity, temperature and air pressure. Apart from providing the individual measurements, it also combines them into an air quality index using on-board processing. Apart from the environmental sensor, a high-quality light sensor is also present, capable of measuring light levels as well as color composition. The combination of acoustic monitoring together with environmental monitoring and on-device machine learning capabilities allow for new research avenues to be explored.

Apart from the environmental monitoring suite, a 6-axis motion sensor is also present to allow the system to trigger based on movement. For example, the system can autonomously decide to start recording after being deployed by a drone.

Storage

In order to allow the system to be deployed for long periods of time, the recorded audio can be stored on a SD-card up to 2 TB. In order to further expand the capabilities of the platform, additional RAM is also available to be able to run more complex Machine Learning (ML) models or to buffer the audio when dealing with samples rates that exceed the real-time performance of the MCU.

Power

As the system is designed to be deployed in remote locations for long periods of time, it contains a Power Management Integrated Circuit (PMIC) capable of harvesting energy from solar panels as well as charge a Lithium-Polymer battery. Apart from being able to charge a battery using solar power, it is also capable of starting the system when the battery is depleted, under the condition that enough solar power is generated. This means that the system can recover when the battery is drained to much and possibly send out a signal for it to be retrieved. The PMIC also enables charging the battery over USB, increasing the ease-of-use in the field.

5.2. Audio Event Detection Implementation

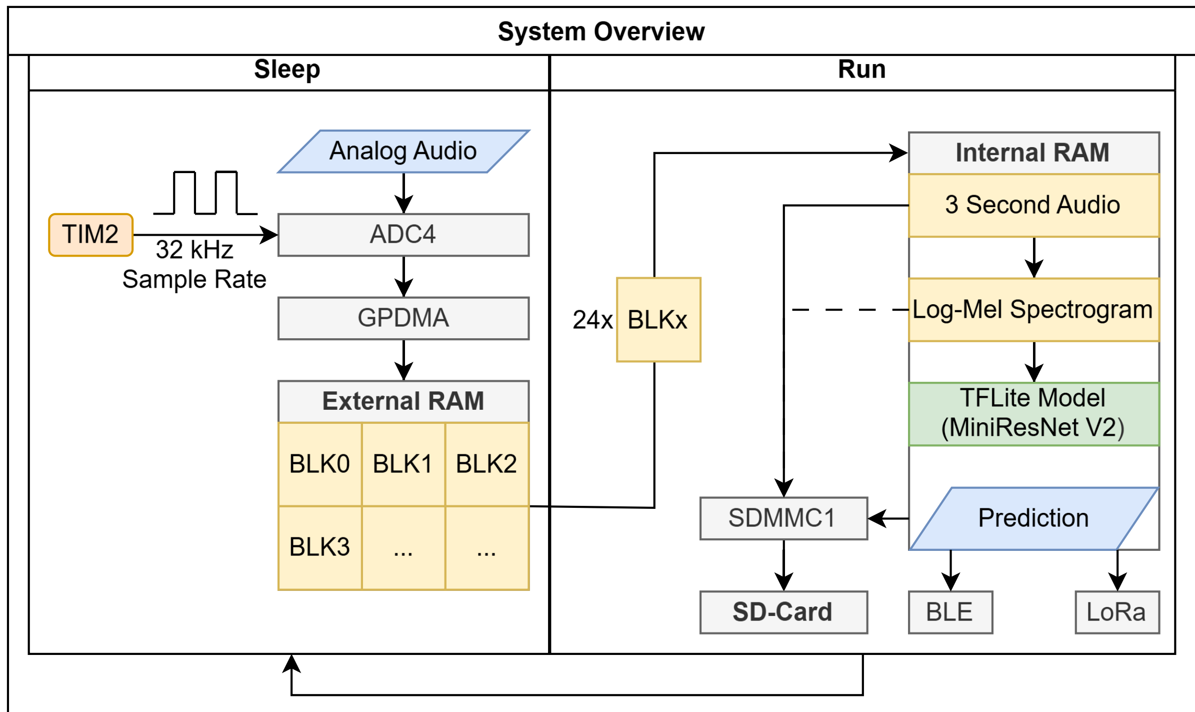


Figure 5.4: Implementation including peripherals and data locations.

Figure 5.4 shows the block diagram of the on-device implementation of the AED pipeline. The loop starts by collecting audio using the ADC4 peripheral from the STM32U5A9. This is a 12-bit Analog to Digital Converter (ADC) which is capable of running autonomously in sleep and even stop modes. A timer is configured to fire at 32 kHz and can be easily changed to facilitate different sampling rates. These 12-bit values are stored as 16-bit integers which are copied over to the external memory by the GPDMA peripheral. This process runs completely independent of the main CPU, meaning that it can be fully asleep while recording audio. The system was designed to be able to perform this process in stop1 and stop2 modes, during which the MCU reaches an even lower power state. However, this is currently not implemented due to missing documentation from STMicroelectronics.

When the external memory contains at least 24 blocks, each consisting of 4096 samples, the system wakes up and copies 24 blocks over to the internal memory for processing. These 24 blocks contain 3.032 seconds of audio at 32 kHz. After copying them over, the signal power is computed using Equation 4.5. A threshold of 2.0×10^{-6} was found to be representative of silence while still picking up faint background sounds. If the signal power exceeds this, the audio is converted to a Log-Mel spectrogram using the parameters from Table 4.15 and fed into the MiniResNet model for inference. The inference result can be stored to the SD-card or broadcast over BLE or LoRa. After this is completed, the system re-enters sleep mode in order to conserve power.

Table 5.1a shows that the total time the system has to be awake per 3-second spectrogram is 979ms. This means that there is around 2 seconds left for additional processing such as handling the environmental sensors, GPS and other desired functions. Table 5.1a also shows that the most time is spent during model inference, which takes 629 ms. This is expected as the model is the most complex part of the pipeline which also can't be optimized further. In contrast, the spectrogram generation takes 805ms without compiler optimizations. With optimizations, this is reduced to 224ms. While the STM32CubeAI software does offer optimizations for the model runtime, these did not provide any speedup for the model.

Table 5.1b shows the resource consumption of the model in terms of Multiply-Accumulates (MACs), flash and RAM. The chosen STM32U5A9 has 4 MB of flash and 2514 KB of RAM, meaning that the model doesn't even consume 10% of either. This leaves plenty of room for future improvements to the

(a) Performance information for the embedded platform. The steps are performed on 3.032 seconds of audio.

System Timings	
Signal Power Calculation (ms)	4ms
Normalization (ms)	7ms
Spectrogram Generation (ms)	264ms
Model inference (ms)	629ms
Save to Disk (ms)	75ms
Total (ms)	979ms

(b) Model resource requirements, reported by the model analysis in STM32CubeIDE V1.19.0.

Model Requirements	
MAC	49.7M
Flash (KiB)	141.55
RAM (KiB)	142.32

Table 5.1: Runtime performance and resource usage of the deployed system.

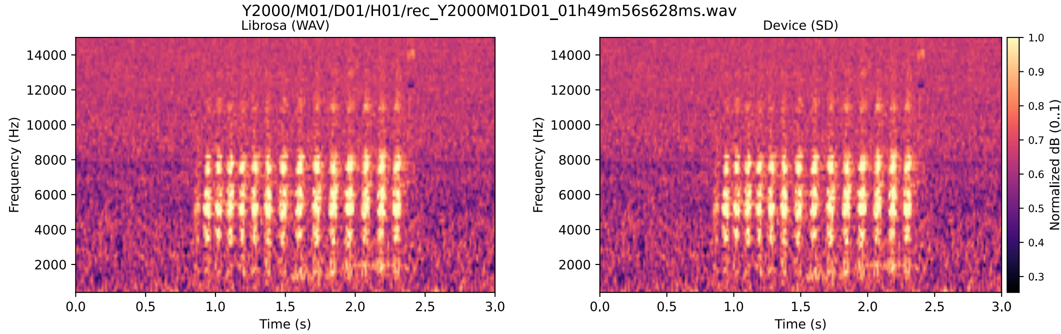


Figure 5.5: Example spectrogram containing a Pica pica call. The left image shows the result from the Python pipeline, while the right image has been created on-device and extracted from the SD-card.

model, such as increasing the input size or adding more layers. It also means that there is plenty of room for additional functionality on the device such as handling the LoRa communications, GPS and sensor suite.

5.3. Spectrogram Verification

In order to ensure that the system is working as expected and the model gets the correct input, the system can be configured to store the spectrograms to disk for post-hoc verification. If any component of the spectrograms is not correct, the model would be unable to correctly perform predictions. Figure 5.5 shows an example spectrogram which was created during testing of the ChirpAlert platform. When comparing the two images, it is clear that the on-device spectrogram verification works with no noticeable differences.

5.4. Power Characteristics

In order to gain an understanding of the power characteristics of the ChirpAlert platform, the system is evaluated in different modes of operation. During these experiments, different power modes are evaluated over 100-second windows on their minimum, average and maximum power consumption. All experiments were performed using a STLinkV3-PWR from STMicroelectronics, which offers a 0.5% accuracy on current measurements [72]. Figure 5.6 shows the experiment setup. An input voltage of 3.6V was applied to the battery input of the ChirpAlert platform, as this is the highest voltage that can be supplied by the STLinkV3-PWR. Table 5.2 shows the power consumption during different power configuration and modes of operation. Five different power modes of the MCU are investigated: shutdown, stop2, stop1, sleep and run.

Shutdown

During the shutdown mode of the system, all peripherals on the MCU are disabled. By default, all the components of the ChirpAlert platform are also disabled. This allows for an ultra-low-power consumption of 0.25 mW while the battery is still connected and capable of charging using solar power. During shutdown mode, the system can still be woken up by the on-board LoRa, BLE, GPS and Inertial

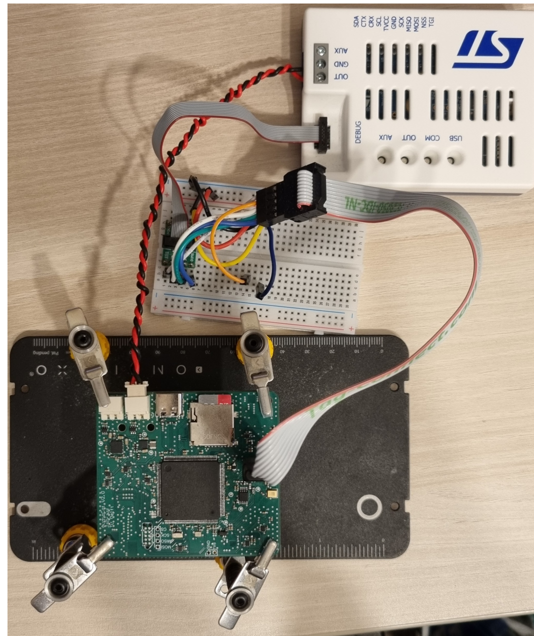


Figure 5.6: Setup for the experiments on the power characteristics. The STLinkV3-PWR is connected to the battery interface and supplies a 3.6V input. It measures with a sampling rate of 1000 Hz and has a reported accuracy of 0.5%.

Measurement Unit (IMU) components. As an example, the shutdown consumption with the BLE radio enabled is also given, showcasing that the system is capable of maintaining a low-power consumption of 0.37 mW while being fully addressable over BLE. As the software for the low-power modes of the LoRa, GPS and IMU are currently underdeveloped, these have not been evaluated.

Stop2 and Stop1

The different stop modes of the MCU offer compromises between low-power consumption and available peripherals. The stop2 mode offers a lower power profile at the cost of some peripherals being unavailable. Meanwhile, stop1 mode offers more peripherals at the cost of a higher energy consumption. Table 5.2 shows this with stop2 and stop1 using 1.97 mW and 2.4 mW, respectively.

The ChirpAlert hardware was designed such that it can keep on recording to memory while the MCU is in stop2 mode. While this is possible, due to an error in the manufacturers libraries for the MCU, this was not possible at this time. In order to gain an idea of the consumption during the stop modes, the microphone, amplifier and ADC were configured in such a way that estimates recording during stop modes, while not moving the data to memory. Table 5.2 gives a rough indication that showcases it might be possible to record audio to memory using 3.05 mW and 3.39 mW for stop2 and stop1 modes, respectively.

Sleep

In the current state of implementation, sleep mode is the lowest power mode that can be attained while simultaneously recording audio to disk. Table 5.2 shows that this is a significant jump from the previous power modes with a power consumption of 31.58 mW. During sleep mode, all system peripherals retain their state and are not disabled or put into low-power modes, resulting in the increase in power consumption.

Recording

Next, the different recording and inference configurations are examined. For these experiments, the system is programmed to record continuously record in sleep mode. When the buffer fills sufficiently, the system wakes up and processes the audio depending on the current configuration, as explained in Figure 5.4. In order to investigate the energy cost of the different stages, they are all investigated separately. It is important to note that these configurations build on each other since they depend on

Table 5.2: Energy Characteristics collected with the STLINK-V3PWR tool. All values are collected over a 100-second window with an input voltage of 3.6V at a sampling rate of 1000 Hz. **Note:** Due to a hardware design error, all configurations except the shutdown configuration have the BLE radio enabled. When disabling the BLE radio, the status LEDs enter a floating state, meaning that the can remain on, resulting in a high current draw. While the BLE radio is enabled, these LEDs can still flicker when the BLE radio exits sleep modes, resulting in short current spikes.

System Power Consumption			
Configuration	**Min (mW)**	**Avg (mW)**	**Max(mW)**
Shutdown	0.08	0.26	0.26
Shutdown w BLE	0.08	0.37	74.09
Stop2	1.31	1.97	68.83
Stop1	1.97	2.4	69.95
Recording Stop2	2.23	3.05	68.94
Recording Stop1	2.82	3.39	69.55
Sleep	31.58	31.58	57.35
Recording w/o PSRAM	32.88	33.52	73.22
Recording	39.71	41.02	94.57
Recording + Audio Storage	46.33	52.01	335.27
Recording + Specs	39.78	43.35	91.66
Recording + Specs + ML	39.71	48.39	103.97
Recording + Specs + ML + SD	46.37	55.61	299.59

the data from the previous stage.

First, the energy cost of the external RAM is investigated. The system was evaluated both with and without buffering to the external RAM. Table 5.2 shows that a significant amount of energy can be saved by recording directly to memory instead of buffering to the external RAM first, saving around 8.5 mW of power by disabling the required peripherals and components.

When ChirpAlert is configured to only record audio and immediately store it to the SD-card, 52.01 mW is used with high peaks up to 335.27 mW. These peaks occur when the recorded audio is written to the SD-card. However, due to the high-speed SD-card interface, this is only a very short peak, resulting in a very respectable power profile while functioning as a static recorder. For comparison, the AudioMoth system consumes an estimated 936 mW while continuously recording audio.

Moving on from the static recorder configuration, the different AED tasks are investigated. When comparing the recording and recording with spectrogram generation steps from Table 5.2, it shows that the system requires only 2.33 mW of additional power to generate a spectrogram every three seconds. The model inference requires an additional 5.04 mW of power as this requires the system to be active for longer periods of time. It also shows that the peak power of the system increases slightly. When saving the prediction results to the SD-card, the power consumption jumps up significantly, again due to the high energy cost of the SD-card interface. The system requires 55.61 mW in order to run the full AED pipeline if the results were saved to disk after every inference results. In practice, this will first be cached in memory and then written to the SD-card in a single burst.

All in all, the power consumption of the different ChirpAlert platform is well within the requirements for long-term, solar-powered deployment. Table 5.2 shows that the ChirpAlert hardware requires 52.01 mW to function as a static recorder and 55.61 mW in order to run on-device model inference including storing the results to disk. In addition, these experiments also show that the system has the potential to be optimized to reduce the power consumption further.

6 Deployment Results

In order to verify the model performance in combination with the ChirpAlert microphone frontend, a deployment was carried out near a popular birdwatching spot called De Nieuwe Driemanspolder in South-Holland, the Netherlands. The goal is to evaluate the performance of the full system and determine if the system is capable of autonomously monitoring the presence of the target species in a real-world environment. In order to validate the performance of the system, the sensor is configured to record all audio to SD card, regardless of the inference result. This allows for a post-hoc analysis of the data using BirdNET with a confidence of 80% to determine the ground truth. In order to fully isolate the Audio Event Detection (AED) component of the system, all modules outside the ones required for AED were disabled during this experiment. For the deployments, the ChirpAlert platform was placed in a somewhat water resisting, 3d printed housing, shown in Figure 6.1.

Figure 6.1: An image of how the sensor was deployed during the field tests. The image shows a 3d printed case with holes for the microphone, environmental sensor and light sensor. A 3300 mAh LiPo battery and 110x80mm solar panel are connected to power the ChirpAlert system.

10th of October

The deployment took place on the 10th of October, 2025, from 10:30am to 17:30pm. The location was chosen as it is known to have a high density of the target species as well as a lot of bird diversity in general. The system was configured using the parameters found in Table 4.15 resulting of audio snippets of 3.032 seconds. If the audio exceeded the set signal power threshold, it was processed and saved to disk, regardless of the inference results. In total, the system recorded 2333 3.032 second audio snippets. All of these recordings were passed through BirdNET to determine 96 detections in the recordings. Of

(a) 10th of October deployment (2333 recordings, 197 detections).

BirdNET	Count
None	32
Eurasian Jackdaw	16
Carrion Crow	1
Hooded Crow	1
Great Tit	1

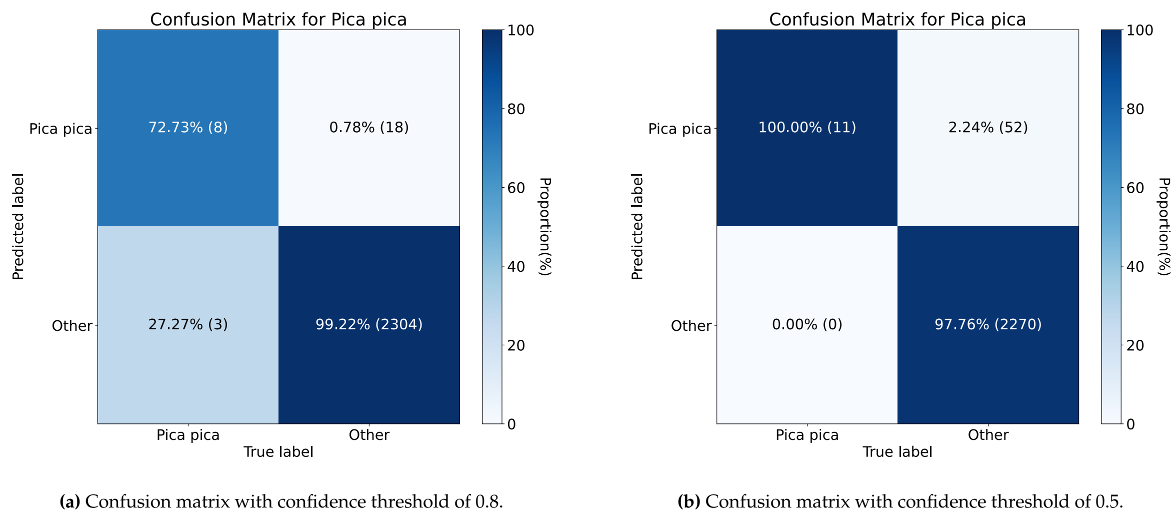
(b) 20th of October deployment (7382 recordings, 160 detections).

BirdNET	Count
None	112
Eurasian Jackdaw	14
European Roller	1
Plush-crested Jay	1

Table 6.1: Comparison of false positive predictions between the 10th and 20th of October deployments near De Nieuwe Driemanspolder.

these 96 detections, 11 were labeled as the target species by BirdNET. The remaining 85 detections were labeled as other species or as noise such as car engines or power tools.

In order to determine the performance of the system, the results from BirdNET were compared to the results from the embedded model. Initially, a threshold of 0.8 was set, effectively this means that the model must be 80% confident or higher that this is the Pica pica species.



(a) Confusion matrix with confidence threshold of 0.8.

(b) Confusion matrix with confidence threshold of 0.5.

Figure 6.2: Confusion matrices from the deployment on the 10th of October near De Nieuwe Driemanspolder, comparing the results from BirdNET with the results from the embedded model at different confidence thresholds.

Figure 6.2a shows that the model performs well during the deployment. It has a 99.22% accuracy for the `other` class and a respectable 72.73% accuracy for the `target` class. While the accuracy of the tttarget class is significantly lower than the 99.86% on the dataset, this is the first time that the hardware of the ChirpAlert platform is used. An important note is that the false positive rate of the model, where it predicts as the `target` class while the truth is the `other` class, is significant. This contradicts the goal of the system, missing a lot of calls. Therefore, the decision threshold was relaxed to 0.5. This allows for a higher recall at the risk of increasing the false positive rate, thereby reducing the precision. As mentioned before, this supports the goal of the system where recall is more important.

Figure 6.2b shows that this has the desired effect of reducing the false negative rate to 0% only increasing the false positive rate.

Table 6.1a shows the true values of the false positive predictions. This shows that a large part of the false positives contain no sound at all according to BirdNET, while a significant chunk is mislabeled as the Eurasian Jackdaw.

20th of October

Based on the results, another deployment was done near De Nieuwe Driemanspolder on the 20th of October from 9:00am till 16:00pm. In total, the system recorded 7382 3.032 second audio snippets. All of these recordings were passed through BirdNET to determine 160 detections in the recordings. Of these

160 detections, 52 were labeled as the target species by BirdNET. The remaining 108 detections were labeled as other species or as noise such as car engines or power tools.

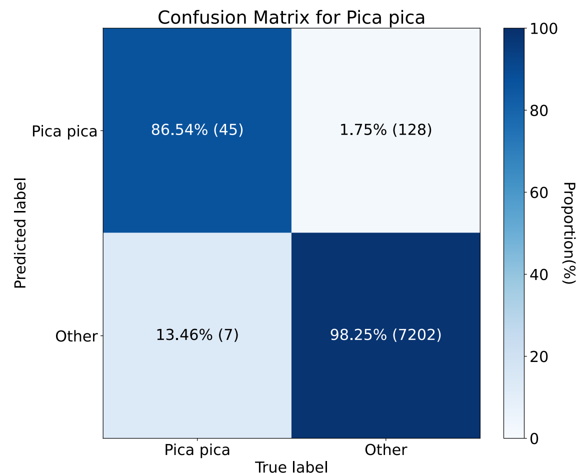
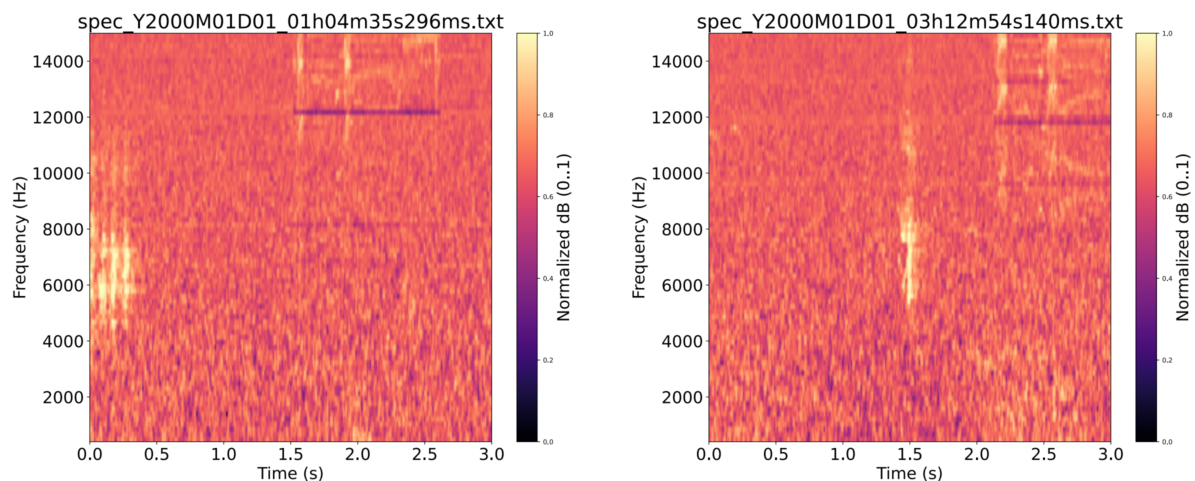


Figure 6.3: Deployment results from the 20th of October near De Nieuwe Driemanspolder, comparing the results from BirdNET with the results from ChirpAlert. The deployment took place from 09:00am till 16:00pm.

Figure 6.3 shows that the model again performs well, albeit with a higher amount of false negatives than with the previous deployment. When investigating the false positives, shown in Table 6.1b, we can again see that the model gets confused when it comes to the Eurasian Jackdaw and a high amount of predictions do not contain a detection at all.



(a) False negative example. A partial Pica pica call is visible from 0.0 till 0.5 seconds.

(b) Another false negative example. A partial Pica pica call is visible around 1.5 seconds.

Figure 6.4: Two false negatives from the 20th of October deployment near De Nieuwe Driemanspolder. BirdNET labeled these as the Pica pica, identifying a call at the left (a) and center (b) of the spectrograms. Figure 5.5 can be used as reference for a full Pica pica call.

Figure 6.4a and Figure 6.4b show two of the spectrograms which were predicted as a false negative. When comparing Figure 6.4a with Figure 5.5, we see that only a partial call is present in the first 0.4 seconds of the spectrogram. Figure 6.4b shows a similar pattern where only a very short call is present at 1.5 seconds. This might be a different type of call that the Pica pica produces which the MiniResNet model is simply not trained on or is to short to be accurately recognized. A more complex model like BirdNET might have a better resolution, making it possible to detect it.

All in all, both deployments show that the system is capable of detecting the Pica pica with a high accuracy while retaining a low-power profile.

7 Future Work

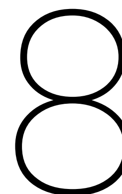
While this research shows that it is possible to create an accurate and efficient bird detection platform using embedded hardware, there is much to be explored. A big aspect that can be improved is the power consumption of the ChirpAlert platform, which is mostly dominated by not being able to record in lower power modes. A big part of this is due to the external flash memory that is used to buffer the audio. However, since the AED pipeline only uses a small portion of the available RAM, the external memory can be dropped all together at a sample rate of 32 kHz. Furthermore, by utilizing the LPDMA peripheral, the system can work in stop2 and stop1 modes without any high speed clock enabled, lowering the power profile to the neighborhood of stop2 and stop1 modes. Additionally, the power profile of running the LoRa, GPS and IMU in shutdown mode should be investigated. This requires finishing their software implementation to include more complex features such as low-power modes.

Another aspect that should be investigated is converting the framework to classify multiple bird species at once. This would greatly expand the usability of the ChirpAlert platform, being able to report more information to the biologist. Previous research has shown that this might also help the model learn to distinguish between classes that have similar sound signatures. In order to facilitate this change, the input and output layers of the model need to be modified to handle multiple classes. Subsequently, the RP90 metric and processing scripts of the framework also need to be able to accommodate multiple classes. The framework can also be expanded to be able to fine-tune the ML models with recordings collected with the ChirpAlert platform. This allows the model to learn from its mistakes and improve the accuracy for the next deployments.

Closely related, the AED pipeline should be tested more extensively, not only on different bird species but also on other types of sounds. The AED pipeline is designed to work with any type of sounds, not only birds and the hyperparameter optimization should find a usable configuration autonomously. There is a good chance that this platform also works for environmental monitoring by recognizing sounds such as rain or thunder. It might be possible to recognize sounds in the urban landscape such as car horns or sirens. A possible way to verify this is by using a dataset such as ESC-10 [63] or UrbanSound8k [73], running a hyperparameter optimization for the optimal parameters and evaluating the system.

Before being able to hand the ChirpAlert platform over to end users, the software needs to be further developed to increase the easy of use as well as make the system more robust against malfunctions. Currently, the software only contains the basic infrastructure to verify and communicate with the different components of the board and only the components that are used for this research are further developed. This can be achieved by simply implementing the desired features.

Lastly, the system should be deployed on a long-time deployment of at least multiple weeks. While the power estimation give a good indication of how long the system can perform on a single charge, real world factors such as temperature and humidity might influence the performance of the battery as well as change the current consumption of the system.



Discussion

Looking back on the process of my thesis, I wish I would have moved on from the STM32 AI ModelZoo sooner. The process of creating the custom framework taught me a lot about the intricacies of the machine learning aspect of the research. I spent too long on trying to optimize the accuracy of the soundscape recordings which in hindsight were not representative for the real world deployment. The lesson I learned from this was to sometimes take a step back and reevaluate what you are trying to achieve instead of blindly moving forward. Another noteworthy aspect is that over the course of my thesis, I had three different supervisors. While this had no effect on the final quality of my thesis, it did slow down my progress.

Looking back on the knowledge I gained during my thesis, I learned a lot regarding hardware design. When starting this endeavor, I had a hobbyist level of PCB design, capable of reading data sheets and combining components into basic circuits. During the development of the ChirpAlert platform, I learned how to use Altium design, the industry standard for PCB design. I learned complex PCB design concepts such as how to deal with high speed signals and how to create a Bluetooth Low Energy (BLE) capable board. In order to debug the hardware, I learned how to use an oscilloscope and how to interpret the results. Apart from the hardware design, I got to refine my skills in regard to writing embedded software. In order to get a feasible system, writing embedded code that efficiently dealt with the incoming data while also being capable of running inference was essential. But the most knowledge I gained was regarding the machine learning part of the system. Apart from a basic understanding of machine learning concepts that I gained during both my studies, I never worked with the intricacies of the machine learning pipeline. This project allowed me to gain a deep understanding of how machine learning models worked, as well as how to deal with large amounts of data in a highly efficient manner.

My new knowledge in hardware, software and machine learning, combined with the important lesson of sometimes taking a step back will be of significant help in my next endeavors. Apart from learning a lot, this thesis is something I am very proud of and I will definitely be continuing the development of the ChirpAlert platform in the future.

9 Conclusion

This thesis set out to address: "How to perform accurate and efficient bird detection using low-cost, resource-constrained embedded hardware?". In order to address this research question, this work presents the design, implementation and evaluation of ChirpAlert, a low-power acoustic sensor platform capable of performing on-device Audio Event Detection (AED) for wildlife monitoring in an efficient, accurate and autonomous matter.

Apart from the ChirpAlert platform, a custom framework was developed which spans from automated dataset generation based on a target bird species up to the deployment on TFLite capable microcontrollers. Through experimentation, several TinyML compatible MiniResNet architectures and preprocessing methods were compared and optimized using hyperparameter tuning, leading to a final configuration that achieved a Recall at 90% Precision (RP90) score of 0.9964 on the custom Eurasian Magpie (Pica pica) dataset. In order to further evaluate the system, a field deployment was performed to demonstrate that the system can autonomously record, process and classify environmental audio in real-world conditions using the ChirpAlert hardware. Over both the field deployments, the system was able to detect 88.8% of the Eurasian Magpie calls when compared with the state-of-the-art and significantly more complex BirdNET model. It was able to correctly classify 98.1% of the other audio segments as not containing the Eurasian Magpie, while only wrongfully predicting 1.9% as Eurasian Magpie calls. A total processing time of 929ms per 3-second audio segment is required, of which 900ms is feature extraction and model inference, showcasing that the system easily meets real-time requirements. ChirpAlert can collect, process and classify audio while retaining a power consumption of 55.61 mW, making it suitable for solar and battery powered deployment with a good indication that this can be lowered further in future work. All in all, these results confirm that lightweight Machine Learning (ML) models are capable of delivering competitive performance while remaining suitable for real-time execution on low-cost, low-power hardware.

In conclusion, ChirpAlert successfully demonstrates that autonomous, low-cost and low-power on-device acoustic monitoring is both feasible and effective. By combining efficient ML architectures with a flexible data pipeline, this platform lays the foundation for scalable environmental monitoring systems that can operate independently in the field for extended periods. The work contributes not only a functional prototype but also an extensible framework for future research into multi-species detection, adaptive learning and long-term ecological observation.

References

[1] Stefan Kahl et al. "BirdNET: A deep learning solution for avian diversity monitoring". In: *Ecological Informatics* 61 (2021), p. 101236.

[2] Annamaria Mesaros et al. "Sound Event Detection: A tutorial". In: *IEEE Signal Processing Magazine* 38.5 (Sept. 2021), pp. 67–83. ISSN: 1558-0792. DOI: 10.1109/msp.2021.3090678. URL: http://dx.doi.org/10.1109/MSP.2021.3090678.

[3] Annamaria Mesaros et al. "Acoustic event detection in real life recordings". In: *2010 18th European Signal Processing Conference*. 2010, pp. 1267–1271.

[4] A.J. Eronen et al. "Audio-based context recognition". In: *IEEE Transactions on Audio, Speech, and Language Processing* 14.1 (2006), pp. 321–329. DOI: 10.1109/TSA.2005.854103.

[5] Selina Chu, Shrikanth Narayanan, and C.-C. Jay Kuo. "Environmental Sound Recognition With Time–Frequency Audio Features". In: *IEEE Transactions on Audio, Speech, and Language Processing* 17.6 (2009), pp. 1142–1158. DOI: 10.1109/TASL.2009.2017438.

[6] Daniele Barchiesi et al. "Acoustic Scene Classification: Classifying environments from the sounds they produce". In: *IEEE Signal Processing Magazine* 32.3 (2015), pp. 16–34. DOI: 10.1109/MSP.2014.2326181.

[7] Shawn Hershey et al. "CNN Architectures for Large-Scale Audio Classification". In: *CoRR* abs/1609.09430 (2016). arXiv: 1609.09430. URL: http://arxiv.org/abs/1609.09430.

[8] Annamaria Mesaros et al. *A decade of DCASE: Achievements, practices, evaluations and future challenges*. 2024. arXiv: 2410.04951 [eess.AS]. URL: https://arxiv.org/abs/2410.04951.

[9] Cagdas Bilen et al. *A Framework for the Robust Evaluation of Sound Event Detection*. 2020. arXiv: 1910.08440 [eess.AS]. URL: https://arxiv.org/abs/1910.08440.

[10] Biao Liu, Yuki Koizumi, and Noboru Harada. "Investigating Waveform and Spectrogram Feature Fusion for Acoustic Scene Classification". In: *Proceedings of the Detection and Classification of Acoustic Scenes and Events (DCASE)*. 2022, pp. 106–110.

[11] Justin Salamon and Juan Pablo Bello. "Comparison of Time-Frequency Representations for Environmental Sound Classification using Convolutional Neural Networks". In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2017, pp. 126–130. DOI: 10.1109/ICASSP.2017.7952132.

[12] Biao Liu, Yuki Koizumi, and Noboru Harada. "Waveforms and Spectrograms: Enhancing Acoustic Scene Classification Using Multimodal Feature Fusion". In: *2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2023, pp. 1–5. DOI: 10.1109/ICASSP49357.2023.10094807.

[13] Yuxuan Wang et al. "Trainable Frontend For Robust and Far-Field Keyword Spotting". In: *CoRR* abs/1607.05666 (2016). arXiv: 1607.05666. URL: http://arxiv.org/abs/1607.05666.

[14] Christopher Ick and Brian McFee. *Sound Event Detection in Urban Audio With Single and Multi-Rate PCEN*. 2021. arXiv: 2102.03468 [eess.AS]. URL: https://arxiv.org/abs/2102.03468.

[15] Vincent Lostanlen et al. "Per-Channel Energy Normalization: Why and How". In: *IEEE Signal Processing Letters* PP (Nov. 2018). DOI: 10.1109/LSP.2018.2878620.

[16] Anastasios Vafeiadis et al. "Audio-based event recognition system for smart homes". In: *2017 IEEE SmartWorld, Ubiquitous Intelligence and Computing, Advanced and Trusted Computed, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. 2017, pp. 1–8. DOI: 10.1109/UIC-ATC.2017.8397489.

[17] Vincent Lostanlen et al. "Robust sound event detection in bioacoustic sensor networks". In: *CoRR* abs/1905.08352 (2019). arXiv: 1905.08352. URL: http://arxiv.org/abs/1905.08352.

[18] Emre Çakir et al. "Convolutional Recurrent Neural Networks for Polyphonic Sound Event Detection". In: *CoRR* abs/1702.06286 (2017). arXiv: 1702.06286. URL: http://arxiv.org/abs/1702.06286.

[19] Sophiya Elangovan and s Jothilakshmi. "Audio event detection using deep learning model". In: *International Journal of Computer Aided Engineering and Technology* 16 (Jan. 2022), p. 328. DOI: 10.1504/IJCAET.2022.10046064.

[20] Tuomas Virtanen, Sharath Adavanne, and Dan Stowell. "Sound Event Detection: A Tutorial". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 31 (2023), pp. 2083–2105. DOI: 10.1109/TASLP.2023.3275160.

[21] Addison Howard et al. *BirdCLEF 2021 - Birdcall Identification*. https://kaggle.com/competitions/birdclef-2021. Kaggle. 2021.

[22] Holger Klinck et al. *BirdCLEF+ 2025*. https://kaggle.com/competitions/birdclef-2025. Kaggle. 2025.

[23] Andrew P. Hill et al. "AudioMoth: A low-cost acoustic device for monitoring biodiversity and the environment". In: *HardwareX* 6 (2019), e00073. ISSN: 2468-0672. DOI: https://doi.org/10.1016/j.ohx.2019.e00073. URL: https://www.sciencedirect.com/science/article/pii/S2468067219300306.

[24] Richard D. Beason, Rüdiger Riesch, and Julia Koricheva. "AURITA: an affordable, autonomous recording device for acoustic monitoring of audible and ultrasonic frequencies". In: *Bioacoustics* 28.4 (2019), pp. 381–396. DOI: 10.1080/09524622.2018.1463293. eprint: https://doi.org/10.1080/09524622.2018.146329. URL: https://doi.org/10.1080/09524622.2018.1463293.

[25] Zhaolan Huang et al. "TinyChirp: Bird Song Recognition Using TinyML Models on Low-power Wireless Acoustic Sensors". In: *arXiv* 2407.21453 (2024). DOI: 10.48550/arXiv.2407.21453. URL: https://arxiv.org/abs/2407.21453.

[26] Lies Zandberg and RF Lachlan. "SongBeam: an automated recorder using beamforming to make high-quality recordings". In: *Open Science Framework Preprint* (2023).

[27] Lukas Schulthess et al. "TinyBird-ML: An ultra-low Power Smart Sensor Node for Bird Vocalization Analysis and Syllable Classification". In: *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, May 2023, pp. 1–5. DOI: 10.1109/iscas46773.2023.10181431. URL: http://dx.doi.org/10.1109/ISCAS46773.2023.10181431.

[28] BirdWeather / Scribe Labs. *BirdWeather PUC: Portable autonomous bird audio recorder and classifier*. Product documentation and technical description. Weather-resistant, battery-powered acoustic recorder with dual microphones, GPS, environmental sensors, on-board storage, and deferred BirdNET-based species identification via the BirdWeather platform. 2024. URL: https://www.birdweather.com/.

[29] Yonina C. Eldar and Gitta Kutyniok. "Generalizations of the Sampling Theorem: Seven Decades After Nyquist". In: *Proceedings of the IEEE* 100.Special Centennial Issue (2012), pp. 1415–1447. DOI: 10.1109/JPROC.2012.2197158.

[30] Orhan Gazi. *Understanding Digital Signal Processing*. Vol. 13. Springer Topics in Signal Processing. Springer Singapore, 2018. ISBN: 978-981-10-4961-3. DOI: 10.1007/978-981-10-4962-0. URL: https://link.springer.com/book/10.1007/978-981-10-4962-0.

[31] Meryam Telmem, Naouar Laaidi, and Hassan Satori. "The impact of MFCC, spectrogram, and Mel-Spectrogram on deep learning models for Amazigh speech recognition system". In: *International Journal of Speech Technology* 28 (2025), pp. 299–312. DOI: 10.1007/s10772-025-10183-3.

[32] Piotr Kukliński and Tomasz Kryszczyński. "Audio Signal Mapping into Spectrogram-Based Images for Deep Learning Applications". In: *2021 20th International Symposium INFOTEH-JAHORINA (INFOTEH)*. IEEE, 2021. DOI: 10.1109/INFOTEH51037.2021.9400698. URL: https://ieeexplore.ieee.org/document/9400698.

[33] Brian A. Oppenheim et al. *The Fourier Transform and Its Applications*. https://see.stanford.edu/materials/lsoftaee261/book-fall-07.pdf. Stanford University, EE261 Course Materials. 2007.

[34] Pau Gairí, Tomàs Pallejà, and Marcel Tresanchez. "Environmental sound recognition on embedded devices using deep learning: a review". In: *Artificial Intelligence Review* 58 (2025). DOI: 10.1007/s10462-025-11106-z. URL: https://link.springer.com/article/10.1007/s10462-025-11106-z.

[35] E. Uyanik and M. Doğan. "Performance Evaluation of Different Window Functions for Audio Fingerprint Based Audio Search Algorithm". In: *Proceedings of the International Conference on Audio Processing*. Presented at ICAp 2020; details from ResearchGate. 2020. URL: https://www.researchgate.net/publication/347041422.

[36] Azamat Mukhamediya, Siamac Fazli, and Amin Zollanvari. "On the Effect of Log-Mel Spectrogram Parameter Tuning for Deep Learning-Based Speech Emotion Recognition". In: *IEEE Access* 11 (2023), pp. 61950–61957. DOI: 10.1109/ACCESS.2023.3287093. URL: https://www.researchgate.net/publication/371663406_On_the_Effect_of_Log-Mel_Spectrogram_Parameter_Tuning_for_Deep_Learning-based_Speech_Emotion_Recognition.

[37] Keunwoo Choi et al. "A Comparison of Audio Signal Preprocessing Methods for Deep Neural Networks on Music Tagging". In: *arXiv* 1709.01922 (2017). DOI: 10.48550/arXiv.1709.01922. URL: https://arxiv.org/abs/1709.01922.

[38] Brian McFee et al. "librosa: Audio and music signal analysis in Python". In: *Proceedings of the 14th Python in Science Conference*. Citeseer. 2015, pp. 18–25.

[39] Guus van Duin. *Xeno-canto recording XC741985: Corvus corone, Zwarte Kraai*. https://xeno-canto.org/741985. Field recording, Netherlands. Recorded on 2022-08-07. Length: 26s, mp3, 44.1kHz stereo. 2022.

[40] Annamaria Mesaros et al. "Detection and Classification of Acoustic Scenes and Events: Outcome of the DCASE 2016 Challenge". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 26.2 (2017), pp. 379–393. DOI: 10.1109/TASLP.2017.2763457.

[41] Sharath Adavanne, Konstantinos Drossos, and Tuomas Virtanen. "Impact of Sound Duration and Inactive Frames on Sound Event Detection Performance". In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2017, pp. 758–762. DOI: 10.1109/ICASSP.2017.7952248.

[42] Annamaria Mesaros, Toni Heittola, and Tuomas Virtanen. "Metrics for Polyphonic Sound Event Detection". In: *Applied Sciences* 11.18 (2021), p. 8689. DOI: 10.3390/app11188689.

[43] Luca Pasa, Enrico Biondi, and Danilo Comminiello. "Sound-Event Detection of Water-Usage Activities Using Transfer Learning". In: *2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2023, pp. 1–5. DOI: 10.1109/ICASSP49357.2023.10094812.

[44] Huy Phan et al. "An Event-based End-to-End Model for Sound Event Detection". In: *2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2020, pp. 66–70. DOI: 10.1109/ICASSP40776.2020.9053796.

[45] Daniel S. Park et al. "SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition". In: *Proceedings of Interspeech 2019*. ISCA. 2019, pp. 2613–2617. DOI: 10.21437/Interspeech.2019-2680.

[46] Dan Stowell, Vassilis Morfi, and Lisa Gill. "Self-Supervised Learning for Few-Shot Bird Sound Classification". In: *2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2022, pp. 306–310. DOI: 10.1109/ICASSP43922.2022.9746539.

[47] Hongyi Zhang et al. "mixup: Beyond Empirical Risk Minimization". In: *International Conference on Learning Representations (ICLR)*. 2018. URL: https://openreview.net/forum?id=r1Ddp1-Rb.

[48] Naoya Takahashi and Yuki Mitsufuji. "Deep Convolutional Neural Network with Mixup for Environmental Sound Classification". In: *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2020 Workshop (DCASE2020)*. 2020, pp. 141–145.

[49] Keiron O'Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks*. 2015. arXiv: 1511.08458 [cs.NE]. URL: https://arxiv.org/abs/1511.08458.

[50] Omar Elharrouss et al. *Task-based Loss Functions in Computer Vision: A Comprehensive Review*. 2025. arXiv: 2504.04242 [cs.LG]. URL: https://arxiv.org/abs/2504.04242.

[51] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[52] Claude E Shannon. "A mathematical theory of communication". In: *The Bell system technical journal* 27.3 (1948), pp. 379–423.

[53] Tsung-Yi Lin et al. "Focal loss for dense object detection". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2980–2988.

[54] Song Han et al. "Tiny Machine Learning: Progress and Futures". In: *Proceedings of the IEEE* 110.9 (2022), pp. 1412–1440. DOI: 10.1109/JPROC.2022.3192940.

[55] STMicroelectronics. *STM32N6x5xx and STM32N6x7xx High-Performance Microcontrollers Datasheet*. Arm Cortex-M55 core, 800MHz, Neural-ART Accelerator NPU, 4.2MB SRAM. STMicroelectronics. July 2025. URL: https://www.st.com/resource/en/datasheet/stm32n655i0.pdf.

[56] Edgaras Liberis. "Taming TinyML: Deep Learning Inference at Computational Extremes". Develops µNAS, differentiable pruning, and Pex techniques for MCU deep inference :contentReference[oaicite:1]index=1. Ph.D. thesis. Cambridge, United Kingdom: University of Cambridge, Dec. 2022. URL: https://www.repository.cam.ac.uk/bitstreams/93172f85-3087-45ff-97a1-7644d14a7eb9/download.

[57] Stanislava Soro. *TinyML for Ubiquitous Edge AI: End of the Year Report*. Technical Report MTR200519. Approved for public release; Distribution unlimited (Public Release Case No. 202709). Bedford, MA, USA: The MITRE Corporation, Nov. 2020. URL: https://arxiv.org/pdf/2102.01255.

[58] Robert David et al. "TensorFlow Lite Micro: Embedded Machine Learning for TinyML Systems". In: *Proceedings of Machine Learning and Systems (MLSys)*. Vol. 3. 2021, pp. 800–811. URL: https://proceedings.mlsys.org/paper/2021/file/6c44dc73014d66ba49b28d483a8f8b0d-Paper.pdf.

[59] Yi Hu et al. "Tin-Tin: Towards Tiny Learning on Tiny Devices with Integer-based Neural Network Training". In: *arXiv preprint arXiv:2504.09405* (Apr. 2025). Published online April 13 2025. URL: https://arxiv.org/abs/2504.09405.

[60] Xiaotian Zhao, Ruge Xu, and Xinfei Guo. "Post-training Quantization or Quantization-aware Training? That is the Question". In: *2023 China Semiconductor Technology International Conference (CSTIC)*. 2023, pp. 1–3. DOI: 10.1109/CSTIC58779.2023.10219214.

[61] Analog Devices, Inc. *LTspice*. https://www.analog.com/ltspice. Version XVII, available at https://www.analog.com/ltspice. 2023.

[62] Lukas Rauch et al. *BirdSet: A Large-Scale Dataset for Audio Classification in Avian Bioacoustics*. 2025. arXiv: 2403.10380 [cs.SD]. URL: https://arxiv.org/abs/2403.10380.

[63] Karol J. Piczak. "ESC: Dataset for Environmental Sound Classification". In: *Proceedings of the 23rd Annual ACM Conference on Multimedia*. Brisbane, Australia: ACM Press, Oct. 13, 2015, pp. 1015–1018. ISBN: 978-1-4503-3459-4. DOI: 10.1145/2733373.2806390. URL: http://dl.acm.org/citation.cfm?doid=2733373.2806390.

[64] Alana D Demko et al. "Divergence in plumage, voice, and morphology indicates speciation in Rufous-capped Warblers (Basileuterus rufifrons)". In: *The Auk* 137.3 (May 2020), ukaa029. ISSN: 1938-4254. DOI: 10.1093/auk/ukaa029. eprint: https://academic.oup.com/auk/article-pdf/137/3/ukaa029/33530063/ukaa029.pdf. URL: https://doi.org/10.1093/auk/ukaa029.

[65] Marcelo Villegas and Paulo Zannin. "Spectral overlap of traffic noise and acoustic signals by birds in an urban forest in Southern Brazil". In: Jan. 2016, pp. 301–328.

[66] STMicroelectronics. *STM32 AI Model Zoo*. Version 3.1.0. GitHub repository. 2025. URL: https://github.com/STMicroelectronics/stm32ai-modelzoo (visited on 10/06/2025).

[67] TensorFlow Developers. *TensorFlow Guide: tf.data*. https://www.tensorflow.org/guide/data. Accessed: 2025-10-03. 2025.

[68] Lisha Li et al. "Hyperband: A novel bandit-based approach to hyperparameter optimization". In: *Journal of Machine Learning Research* 18.185 (2018), pp. 1–52.

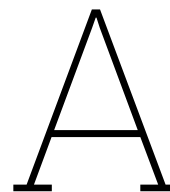
[69] Ilya Loshchilov and Frank Hutter. "SGDR: Stochastic Gradient Descent with Warm Restarts". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2017. arXiv: 1608.03983 [cs.LG]. URL: https://arxiv.org/abs/1608.03983.

[70] ARM CMSIS Team. *CMSIS Version 5: Tools, Software, and Interfaces*. Version 5, GitHub repository. ARM Ltd. 2016. URL: https://github.com/ARM-software/CMSIS_5.

[71] *ENACT: Advancing Health Through Environmental Risk Analytics*. Project website. ENACT Project (Horizon Europe). 2025. URL: https://enact-he.eu/ (visited on 10/06/2025).

[72] *STLINK-V3PWR: Programmer/Debugger with Advanced Power Measurement*. User manual and technical reference. STMicroelectronics. Geneva, Switzerland, 2024.

[73] Justin Salamon, Christopher Jacoby, and Juan Pablo Bello. "A Dataset and Taxonomy for Urban Sound Research". In: *Proceedings of the 22nd ACM International Conference on Multimedia (ACM-MM)*. ACM, 2014, pp. 1041–1044. DOI: 10.1145/2647868.2655045. URL: https://www.justinsalamon.com/uploads/4/3/9/4/4394963/salamon_urbansound_acmmm14.pdf.



ModelZoo Experiments Results

Table A.1: Experiment 1: Short Audio Results

Model	Ver	Stacks	Mels	Frames	Category	F1_target	F1_other	F1_background
miniresnet	v1	2	64	32	rucwar	0.283(0.158)	0.333(0.048)	0.501(0.294)
miniresnet	v1	2	64	32	reevir1	0.203(0.093)	0.479(0.122)	0.722(0.142)
miniresnet	v1	2	128	64	bobfly1	0.443(0.038)	0.552(0.078)	0.705(0.170)
miniresnet	v1	2	64	32	bobfly1	0.288(0.112)	0.498(0.087)	0.729(0.070)
miniresnet	v1	2	64	64	bobfly1	0.351(0.060)	0.489(0.076)	0.597(0.268)
miniresnet	v1	2	128	96	reevir1	0.378(0.031)	0.642(0.030)	0.756(0.106)
miniresnet	v1	2	128	64	rucwar	0.445(0.079)	0.639(0.078)	0.850(0.098)
miniresnet	v1	2	64	96	reevir1	0.277(0.153)	0.538(0.083)	0.732(0.069)
miniresnet	v1	2	64	64	rucwar	0.276(0.131)	0.466(0.086)	0.693(0.208)
miniresnet	v1	2	64	96	bobfly1	0.255(0.120)	0.477(0.057)	0.533(0.234)
miniresnet	v1	2	128	32	reevir1	0.343(0.144)	0.676(0.073)	0.842(0.065)
miniresnet	v1	2	64	96	rucwar	0.239(0.135)	0.432(0.159)	0.760(0.100)
miniresnet	v1	2	128	32	rucwar	0.328(0.113)	0.435(0.070)	0.739(0.108)
miniresnet	v1	2	64	64	reevir1	0.229(0.123)	0.595(0.049)	0.409(0.332)
miniresnet	v1	2	128	32	bobfly1	0.350(0.164)	0.510(0.104)	0.735(0.088)
miniresnet	v1	2	128	96	rucwar	0.360(0.122)	0.406(0.189)	0.777(0.098)
miniresnet	v1	2	128	64	reevir1	0.293(0.117)	0.631(0.105)	0.560(0.253)
miniresnet	v1	2	128	96	bobfly1	0.375(0.069)	0.537(0.071)	0.764(0.096)
miniresnet	v1	1	64	96	bobfly1	0.329(0.140)	0.582(0.065)	0.729(0.097)
miniresnet	v1	1	64	96	reevir1	0.432(0.043)	0.653(0.072)	0.791(0.059)
miniresnet	v1	1	128	64	bobfly1	0.541(0.050)	0.639(0.061)	0.854(0.039)
miniresnet	v1	1	128	96	rucwar	0.437(0.189)	0.594(0.112)	0.870(0.080)
miniresnet	v1	1	64	64	bobfly1	0.472(0.039)	0.560(0.094)	0.679(0.163)
miniresnet	v1	1	64	96	rucwar	0.353(0.184)	0.475(0.111)	0.799(0.074)
miniresnet	v1	1	64	32	rucwar	0.396(0.106)	0.430(0.060)	0.751(0.092)
miniresnet	v1	1	64	64	reevir1	0.339(0.109)	0.650(0.089)	0.682(0.238)
miniresnet	v1	1	128	32	bobfly1	0.482(0.084)	0.532(0.108)	0.813(0.043)
miniresnet	v1	1	64	32	reevir1	0.294(0.193)	0.647(0.091)	0.857(0.024)
miniresnet	v1	1	64	64	rucwar	0.442(0.113)	0.409(0.092)	0.824(0.044)
miniresnet	v1	1	128	64	reevir1	0.382(0.054)	0.654(0.095)	0.690(0.229)
miniresnet	v1	1	64	32	bobfly1	0.391(0.050)	0.556(0.070)	0.769(0.049)
miniresnet	v1	1	128	32	rucwar	0.347(0.056)	0.471(0.055)	0.810(0.040)
miniresnet	v1	1	128	32	reevir1	0.395(0.123)	0.651(0.071)	0.840(0.040)
miniresnet	v1	1	128	96	reevir1	0.377(0.157)	0.702(0.053)	0.851(0.067)

Continued on next page

Model	Ver	Stacks	Mels	Frames	Category	F1_target	F1_other	F1_background
miniresnet	v1	1	128	64	rucwar	0.492(0.092)	0.626(0.085)	0.838(0.102)
miniresnet	v1	1	128	96	bobfly1	0.443(0.088)	0.639(0.054)	0.819(0.051)
miniresnet	v2	2	64	96	bobfly1	0.344(0.061)	0.481(0.069)	0.553(0.210)
miniresnet	v2	2	128	32	rucwar	0.289(0.072)	0.403(0.070)	0.532(0.116)
miniresnet	v2	2	64	64	reevir1	0.241(0.098)	0.705(0.124)	0.403(0.315)
miniresnet	v2	2	128	64	rucwar	0.390(0.119)	0.499(0.090)	0.669(0.179)
miniresnet	v2	2	64	64	rucwar	0.359(0.057)	0.350(0.078)	0.446(0.242)
miniresnet	v2	2	64	32	rucwar	0.247(0.086)	0.313(0.087)	0.570(0.176)
miniresnet	v2	2	128	64	reevir1	0.293(0.123)	0.681(0.082)	0.506(0.271)
miniresnet	v2	2	128	96	bobfly1	0.405(0.042)	0.547(0.069)	0.547(0.115)
miniresnet	v2	2	128	96	reevir1	0.411(0.058)	0.594(0.127)	0.660(0.139)
miniresnet	v2	2	128	32	bobfly1	0.374(0.044)	0.502(0.073)	0.641(0.106)
miniresnet	v2	2	64	32	reevir1	0.250(0.120)	0.420(0.138)	0.647(0.106)
miniresnet	v2	2	64	96	rucwar	0.253(0.040)	0.449(0.060)	0.674(0.128)
miniresnet	v2	2	64	64	bobfly1	0.349(0.065)	0.540(0.076)	0.401(0.215)
miniresnet	v2	2	64	32	bobfly1	0.272(0.142)	0.421(0.087)	0.539(0.244)
miniresnet	v2	2	128	96	rucwar	0.388(0.080)	0.494(0.111)	0.667(0.163)
miniresnet	v2	2	128	32	reevir1	0.386(0.045)	0.561(0.127)	0.728(0.142)
miniresnet	v2	2	128	64	bobfly1	0.379(0.074)	0.551(0.030)	0.617(0.225)
miniresnet	v2	2	64	96	reevir1	0.349(0.039)	0.580(0.111)	0.386(0.196)
miniresnet	v2	1	64	32	reevir1	0.403(0.074)	0.704(0.074)	0.837(0.076)
miniresnet	v2	1	64	96	reevir1	0.420(0.025)	0.689(0.071)	0.749(0.080)
miniresnet	v2	1	64	32	bobfly1	0.371(0.174)	0.557(0.099)	0.823(0.039)
miniresnet	v2	1	128	64	bobfly1	0.436(0.054)	0.634(0.054)	0.715(0.136)
miniresnet	v2	1	128	32	reevir1	0.415(0.123)	0.664(0.038)	0.829(0.113)
miniresnet	v2	1	128	64	rucwar	0.481(0.051)	0.617(0.082)	0.850(0.097)
miniresnet	v2	1	64	64	bobfly1	0.400(0.081)	0.537(0.125)	0.673(0.193)
miniresnet	v2	1	64	96	rucwar	0.308(0.069)	0.482(0.100)	0.676(0.169)
miniresnet	v2	1	128	32	rucwar	0.428(0.071)	0.520(0.092)	0.779(0.059)
miniresnet	v2	1	64	32	rucwar	0.382(0.064)	0.452(0.054)	0.707(0.069)
miniresnet	v2	1	128	96	bobfly1	0.383(0.084)	0.609(0.081)	0.742(0.128)
miniresnet	v2	1	64	64	rucwar	0.332(0.141)	0.468(0.069)	0.745(0.110)
miniresnet	v2	1	128	32	bobfly1	0.442(0.047)	0.530(0.092)	0.819(0.050)
miniresnet	v2	1	64	96	bobfly1	0.375(0.097)	0.568(0.071)	0.592(0.165)
miniresnet	v2	1	128	96	rucwar	0.418(0.043)	0.547(0.075)	0.834(0.025)
miniresnet	v2	1	128	96	reevir1	0.458(0.095)	0.615(0.140)	0.766(0.068)
miniresnet	v2	1	64	64	reevir1	0.331(0.180)	0.629(0.195)	0.692(0.231)
miniresnet	v2	1	128	64	reevir1	0.317(0.153)	0.715(0.069)	0.660(0.208)

Table A.2: Experiment 1: Soundscape Results

Model	Ver	Stacks	Mels	Frames	Category	F1_target	F1_other	F1_background
miniresnet	v1	2	64	32	rucwar	0.241(0.175)	0.069(0.063)	0.466(0.244)
miniresnet	v1	2	64	32	reevir1	0.176(0.154)	0.163(0.102)	0.479(0.254)
miniresnet	v1	2	128	64	bobfly1	0.312(0.181)	0.072(0.073)	0.533(0.259)
miniresnet	v1	2	64	32	bobfly1	0.367(0.134)	0.068(0.017)	0.665(0.064)
miniresnet	v1	2	64	64	bobfly1	0.339(0.168)	0.049(0.046)	0.456(0.269)
miniresnet	v1	2	128	96	reevir1	0.122(0.086)	0.124(0.067)	0.595(0.101)
miniresnet	v1	2	128	64	rucwar	0.250(0.209)	0.073(0.044)	0.661(0.077)
miniresnet	v1	2	64	96	reevir1	0.123(0.189)	0.133(0.073)	0.577(0.119)
miniresnet	v1	2	64	64	rucwar	0.302(0.149)	0.126(0.071)	0.602(0.071)

Continued on next page

Model	Ver	Stacks	Mels	Frames	Category	F1_target	F1_other	F1_background
miniresnet	v1	2	64	96	bobfly1	0.250(0.203)	0.051(0.040)	0.442(0.208)
miniresnet	v1	2	128	32	reevir1	0.105(0.126)	0.225(0.165)	0.672(0.109)
miniresnet	v1	2	64	96	rucwar	0.136(0.087)	0.116(0.100)	0.664(0.094)
miniresnet	v1	2	128	32	rucwar	0.231(0.171)	0.063(0.053)	0.566(0.133)
miniresnet	v1	2	64	64	reevir1	0.297(0.172)	0.182(0.102)	0.303(0.284)
miniresnet	v1	2	128	32	bobfly1	0.360(0.160)	0.084(0.081)	0.535(0.177)
miniresnet	v1	2	128	96	rucwar	0.313(0.091)	0.164(0.122)	0.631(0.163)
miniresnet	v1	2	128	64	reevir1	0.374(0.094)	0.142(0.052)	0.267(0.206)
miniresnet	v1	2	128	96	bobfly1	0.246(0.117)	0.087(0.053)	0.540(0.090)
miniresnet	v1	1	64	96	bobfly1	0.385(0.195)	0.098(0.077)	0.488(0.139)
miniresnet	v1	1	64	96	reevir1	0.193(0.094)	0.129(0.092)	0.670(0.076)
miniresnet	v1	1	128	64	bobfly1	0.337(0.166)	0.098(0.049)	0.480(0.173)
miniresnet	v1	1	128	96	rucwar	0.072(0.069)	0.143(0.095)	0.608(0.166)
miniresnet	v1	1	64	64	bobfly1	0.466(0.100)	0.069(0.012)	0.362(0.190)
miniresnet	v1	1	64	96	rucwar	0.133(0.080)	0.125(0.067)	0.472(0.279)
miniresnet	v1	1	64	32	rucwar	0.213(0.168)	0.144(0.072)	0.526(0.122)
miniresnet	v1	1	64	64	reevir1	0.236(0.186)	0.155(0.094)	0.450(0.262)
miniresnet	v1	1	128	32	bobfly1	0.357(0.135)	0.056(0.046)	0.631(0.107)
miniresnet	v1	1	64	32	reevir1	0.116(0.176)	0.244(0.126)	0.710(0.072)
miniresnet	v1	1	64	64	rucwar	0.228(0.153)	0.112(0.086)	0.693(0.040)
miniresnet	v1	1	128	64	reevir1	0.246(0.167)	0.121(0.043)	0.397(0.288)
miniresnet	v1	1	64	32	bobfly1	0.358(0.162)	0.103(0.030)	0.616(0.060)
miniresnet	v1	1	128	32	rucwar	0.261(0.114)	0.110(0.061)	0.674(0.054)
miniresnet	v1	1	128	32	reevir1	0.187(0.152)	0.172(0.088)	0.638(0.106)
miniresnet	v1	1	128	96	reevir1	0.139(0.088)	0.113(0.050)	0.652(0.012)
miniresnet	v1	1	128	64	rucwar	0.248(0.141)	0.129(0.071)	0.586(0.120)
miniresnet	v1	1	128	96	bobfly1	0.242(0.213)	0.032(0.028)	0.616(0.203)
miniresnet	v2	2	64	96	bobfly1	0.344(0.183)	0.071(0.041)	0.350(0.292)
miniresnet	v2	2	128	32	rucwar	0.338(0.160)	0.055(0.042)	0.345(0.148)
miniresnet	v2	2	64	64	reevir1	0.319(0.186)	0.146(0.161)	0.306(0.282)
miniresnet	v2	2	128	64	rucwar	0.352(0.148)	0.101(0.097)	0.499(0.226)
miniresnet	v2	2	64	64	rucwar	0.300(0.192)	0.065(0.057)	0.245(0.285)
miniresnet	v2	2	64	32	rucwar	0.229(0.181)	0.069(0.079)	0.449(0.246)
miniresnet	v2	2	128	64	reevir1	0.348(0.170)	0.114(0.086)	0.228(0.264)
miniresnet	v2	2	128	96	bobfly1	0.315(0.144)	0.111(0.028)	0.194(0.216)
miniresnet	v2	2	128	96	reevir1	0.246(0.157)	0.039(0.043)	0.369(0.283)
miniresnet	v2	2	128	32	bobfly1	0.355(0.149)	0.075(0.033)	0.475(0.162)
miniresnet	v2	2	64	32	reevir1	0.242(0.202)	0.168(0.080)	0.626(0.089)
miniresnet	v2	2	64	96	rucwar	0.259(0.114)	0.130(0.069)	0.291(0.246)
miniresnet	v2	2	64	64	bobfly1	0.432(0.029)	0.107(0.047)	0.150(0.135)
miniresnet	v2	2	64	32	bobfly1	0.311(0.167)	0.057(0.058)	0.401(0.309)
miniresnet	v2	2	128	96	rucwar	0.299(0.168)	0.099(0.100)	0.334(0.269)
miniresnet	v2	2	128	32	reevir1	0.238(0.132)	0.121(0.089)	0.487(0.180)
miniresnet	v2	2	128	64	bobfly1	0.470(0.023)	0.080(0.042)	0.156(0.139)
miniresnet	v2	2	64	96	reevir1	0.362(0.052)	0.093(0.078)	0.054(0.095)
miniresnet	v2	1	64	32	reevir1	0.166(0.125)	0.201(0.156)	0.630(0.102)
miniresnet	v2	1	64	96	reevir1	0.251(0.088)	0.060(0.055)	0.608(0.124)
miniresnet	v2	1	64	32	bobfly1	0.278(0.196)	0.090(0.060)	0.569(0.226)
miniresnet	v2	1	128	64	bobfly1	0.382(0.080)	0.133(0.021)	0.407(0.151)
miniresnet	v2	1	128	32	reevir1	0.237(0.137)	0.205(0.059)	0.473(0.209)
miniresnet	v2	1	128	64	rucwar	0.296(0.170)	0.136(0.076)	0.607(0.059)
miniresnet	v2	1	64	64	bobfly1	0.432(0.092)	0.075(0.079)	0.308(0.240)
miniresnet	v2	1	64	96	rucwar	0.210(0.114)	0.098(0.066)	0.338(0.297)
miniresnet	v2	1	128	32	rucwar	0.293(0.144)	0.068(0.073)	0.439(0.155)

Continued on next page

Model	Ver	Stacks	Mels	Frames	Category	F1_target	F1_other	F1_background
miniresnet	v2	1	64	32	rucwar	0.259(0.142)	0.055(0.041)	0.549(0.187)
miniresnet	v2	1	128	96	bobfly1	0.342(0.182)	0.096(0.041)	0.377(0.233)
miniresnet	v2	1	64	64	rucwar	0.274(0.187)	0.100(0.068)	0.507(0.270)
miniresnet	v2	1	128	32	bobfly1	0.337(0.118)	0.087(0.036)	0.524(0.097)
miniresnet	v2	1	64	96	bobfly1	0.423(0.200)	0.146(0.061)	0.287(0.251)
miniresnet	v2	1	128	96	rucwar	0.157(0.139)	0.084(0.063)	0.492(0.214)
miniresnet	v2	1	128	96	reevir1	0.266(0.061)	0.099(0.094)	0.506(0.089)
miniresnet	v2	1	64	64	reevir1	0.251(0.162)	0.136(0.080)	0.396(0.315)
miniresnet	v2	1	128	64	reevir1	0.332(0.158)	0.156(0.080)	0.272(0.284)

Table A.3: Experiment 2: Short Audio Results

Model	Ver	Stacks	Mels	Frames	Category	F1_target	F1_other	F1_background
miniresnet	v1	2	128	64	bobfly1	0.456(0.100)	0.567(0.083)	0.697(0.153)
miniresnet	v1	2	128	64	rucwar	0.419(0.143)	0.611(0.026)	0.852(0.063)
miniresnet	v1	2	128	64	reevir1	0.313(0.167)	0.615(0.051)	0.624(0.265)
miniresnet	v1	2	128	96	reevir1	0.390(0.080)	0.655(0.078)	0.823(0.075)
miniresnet	v1	2	128	96	rucwar	0.359(0.068)	0.560(0.069)	0.770(0.079)
miniresnet	v1	2	128	96	bobfly1	0.409(0.037)	0.584(0.079)	0.662(0.094)
miniresnet	v2	2	128	64	rucwar	0.347(0.046)	0.555(0.085)	0.680(0.184)
miniresnet	v2	2	128	96	bobfly1	0.367(0.041)	0.532(0.093)	0.662(0.162)
miniresnet	v2	2	128	96	rucwar	0.365(0.065)	0.427(0.186)	0.767(0.079)
miniresnet	v2	2	128	64	bobfly1	0.390(0.094)	0.599(0.046)	0.637(0.211)
miniresnet	v2	2	128	64	reevir1	0.350(0.118)	0.715(0.066)	0.616(0.241)
miniresnet	v2	2	128	96	reevir1	0.422(0.061)	0.612(0.111)	0.602(0.124)
miniresnet	v2	1	128	64	reevir1	0.312(0.140)	0.603(0.140)	0.669(0.194)
miniresnet	v2	1	128	96	bobfly1	0.437(0.037)	0.603(0.067)	0.766(0.072)
miniresnet	v2	1	128	64	rucwar	0.451(0.032)	0.563(0.080)	0.841(0.044)
miniresnet	v2	1	128	96	rucwar	0.353(0.206)	0.485(0.111)	0.796(0.063)
miniresnet	v2	1	128	64	bobfly1	0.456(0.068)	0.608(0.046)	0.751(0.075)
miniresnet	v2	1	128	96	reevir1	0.429(0.061)	0.626(0.071)	0.681(0.124)
miniresnet	v1	1	128	64	bobfly1	0.535(0.045)	0.643(0.047)	0.817(0.113)
miniresnet	v1	1	128	64	reevir1	0.414(0.087)	0.710(0.045)	0.713(0.221)
miniresnet	v1	1	128	64	rucwar	0.505(0.059)	0.666(0.029)	0.892(0.026)
miniresnet	v1	1	128	96	bobfly1	0.465(0.041)	0.616(0.073)	0.777(0.089)
miniresnet	v1	1	128	96	reevir1	0.476(0.047)	0.738(0.019)	0.850(0.044)
miniresnet	v1	1	128	96	rucwar	0.471(0.085)	0.645(0.055)	0.849(0.075)

Table A.4: Experiment 2: Soundscape Results

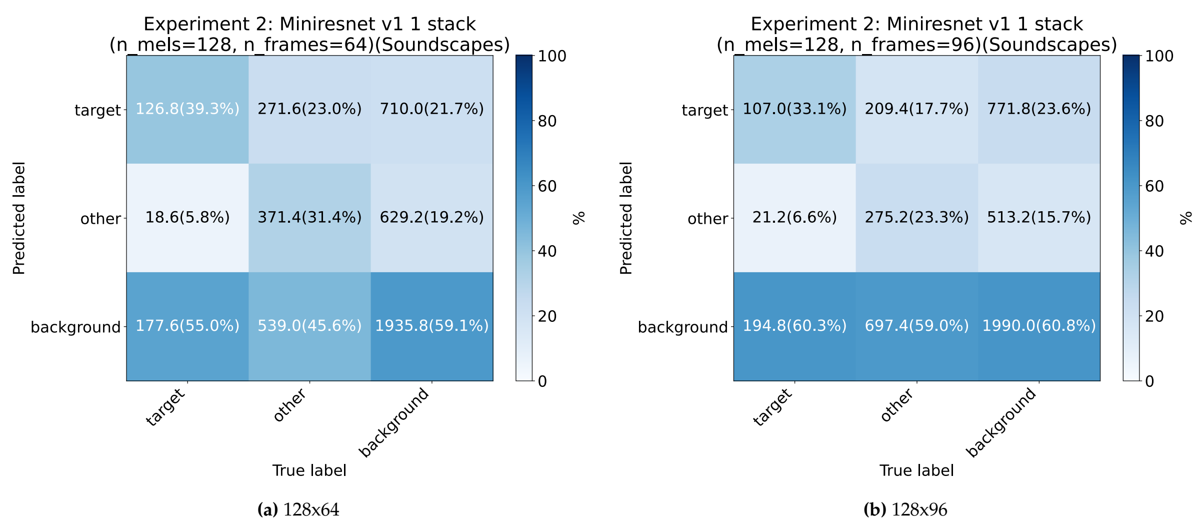
Model	Ver	Stacks	Mels	Frames	Category	F1_target	F1_other	F1_background
miniresnet	v1	2	128	64	bobfly1	0.328(0.114)	0.080(0.050)	0.535(0.190)
miniresnet	v1	2	128	64	rucwar	0.250(0.143)	0.082(0.066)	0.660(0.089)
miniresnet	v1	2	128	64	reevir1	0.253(0.138)	0.181(0.051)	0.433(0.269)
miniresnet	v1	2	128	96	reevir1	0.127(0.091)	0.135(0.091)	0.685(0.078)
miniresnet	v1	2	128	96	rucwar	0.272(0.129)	0.121(0.031)	0.520(0.187)
miniresnet	v1	2	128	96	bobfly1	0.324(0.123)	0.111(0.083)	0.421(0.162)
miniresnet	v2	2	128	64	rucwar	0.233(0.181)	0.125(0.036)	0.523(0.266)
miniresnet	v2	2	128	96	bobfly1	0.249(0.190)	0.067(0.042)	0.351(0.242)

Continued on next page

Model	Ver	Stacks	Mels	Frames	Category	F1_target	F1_other	F1_background
miniresnet	v2	2	128	96	rucwar	0.141(0.151)	0.061(0.061)	0.594(0.265)
miniresnet	v2	2	128	64	bobfly1	0.369(0.067)	0.119(0.054)	0.332(0.211)
miniresnet	v2	2	128	64	reevir1	0.298(0.150)	0.160(0.183)	0.281(0.327)
miniresnet	v2	2	128	96	reevir1	0.297(0.078)	0.082(0.095)	0.259(0.232)
miniresnet	v2	1	128	64	reevir1	0.266(0.118)	0.104(0.071)	0.225(0.214)
miniresnet	v2	1	128	96	bobfly1	0.268(0.154)	0.116(0.027)	0.453(0.136)
miniresnet	v2	1	128	64	rucwar	0.263(0.162)	0.047(0.051)	0.549(0.201)
miniresnet	v2	1	128	96	rucwar	0.044(0.075)	0.050(0.062)	0.603(0.247)
miniresnet	v2	1	128	64	bobfly1	0.411(0.044)	0.084(0.028)	0.298(0.151)
miniresnet	v2	1	128	96	reevir1	0.312(0.070)	0.072(0.062)	0.301(0.248)
miniresnet	v1	1	128	64	bobfly1	0.313(0.166)	0.053(0.042)	0.540(0.213)
miniresnet	v1	1	128	64	reevir1	0.204(0.160)	0.146(0.085)	0.437(0.283)
miniresnet	v1	1	128	64	rucwar	0.317(0.099)	0.141(0.066)	0.639(0.097)
miniresnet	v1	1	128	96	bobfly1	0.326(0.164)	0.103(0.058)	0.447(0.249)
miniresnet	v1	1	128	96	reevir1	0.174(0.103)	0.068(0.078)	0.629(0.087)
miniresnet	v1	1	128	96	rucwar	0.233(0.111)	0.117(0.066)	0.608(0.179)

B

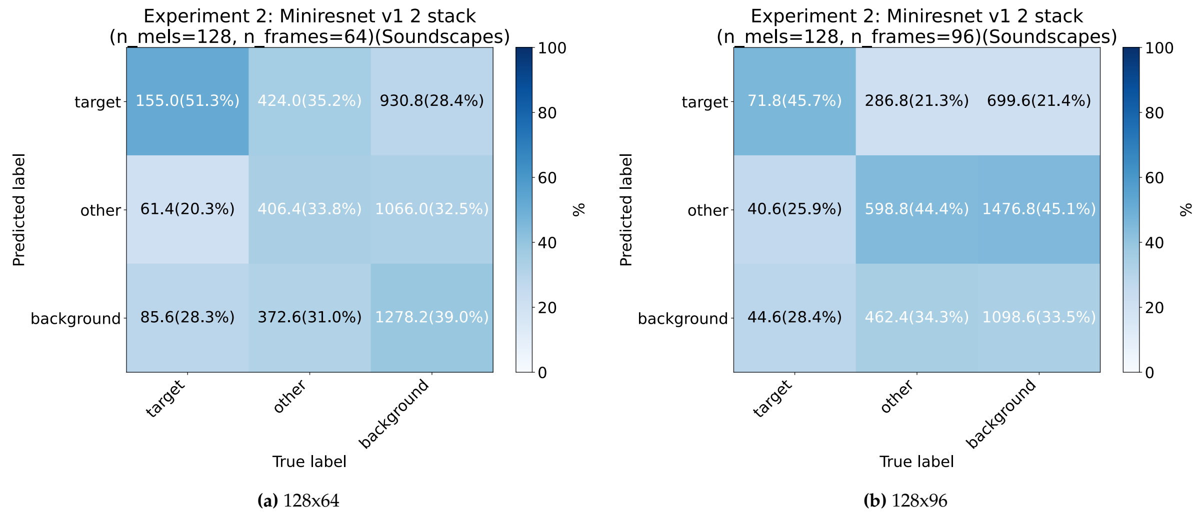
MiniRes Experiment 2 Confusion Matrices



(a) 128x64

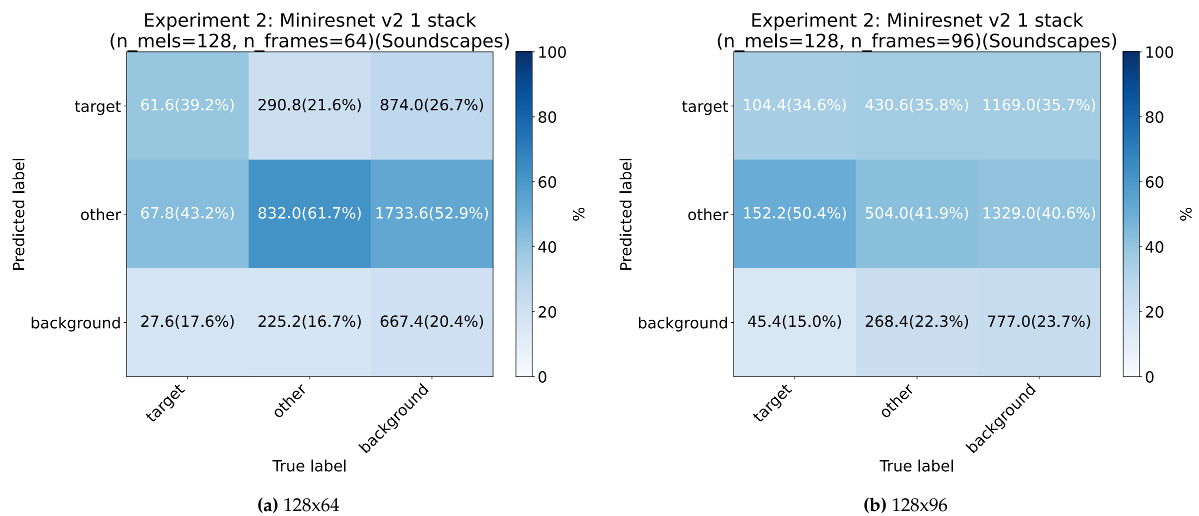
(b) 128x96

Figure B.1: MiniResNet v1 (1-stack): F1 across folds on soundscapes.



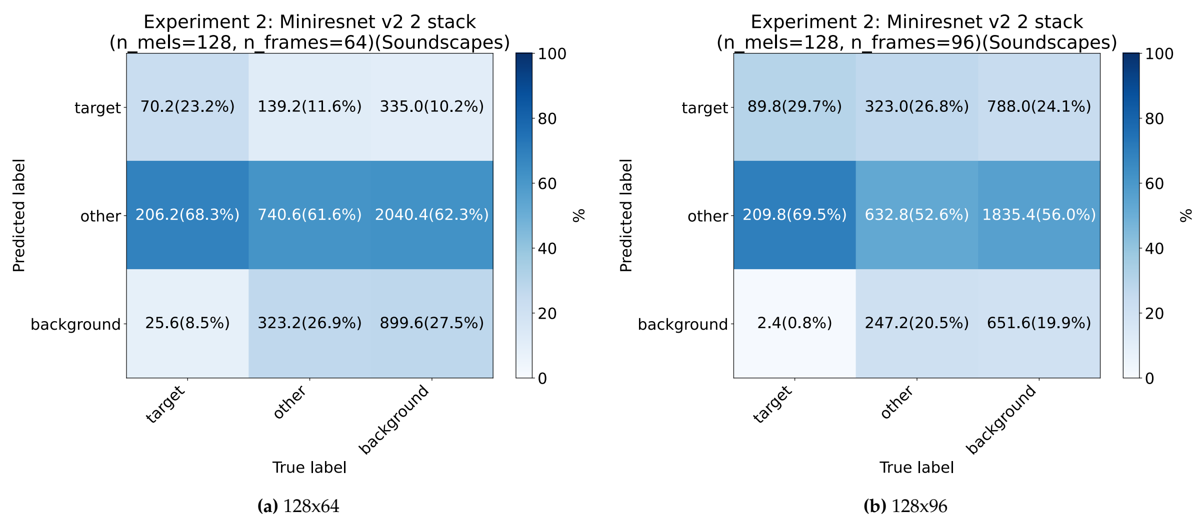
(a) 128x64 **(b)** 128x96

Figure B.2: MiniResNet v1 (2-stack): F1 across folds on soundscapes.



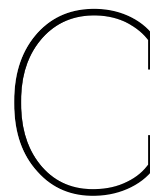
(a) 128x64 **(b)** 128x96

Figure B.3: MiniResNet v2 (1-stack): F1 across folds on soundscapes.



(a) 128x64 **(b)** 128x96

Figure B.4: MiniResNet v2 (2-stack): F1 across folds on soundscapes.



Hyperparameter Space

Table C.1: Data hyperparameter search space.

Data Parameter	**Search Space**	**Description**
Sample Rate	$\{16\text{kHz}, 32\text{kHz}\}$	Sample Rate of the Audio, will automatically be resampled
Mels	$\{64, 80, 96, 128\}$	Amount of Mel bins
FFT Length	$\{512, 1024\}$	Length of the FFT window
Hop Length	$[96,\ 512]_{\Delta=16}$	Number of samples between successive frames
Window Type	$\{\text{Hann}, \text{Hamming}\}$	Type of window, see
Min Frequency	$[0,\ 1000]_{\Delta=200}$	Minimum frequency of the Mel spectrogram
Max Frequency	$[6000,\ 15000]_{\Delta=1000}$	Maximum frequency of the Mel spectrogram
Butterworth Order	$\{0, 2, 4, 6, 8\}$	Order for the Butterworth bandpass filter, 0 means disabled
Bandpass Low Frequency	$[\text{Min Frequency},\ 1000]_{\Delta=200}$	Bottom frequency of the bandpass filter
Bandpass High Frequency	$[7500,\ \lfloor \text{Sample Rate}/2 \rfloor - 1]_{\Delta=500}$	Top Frequency of the bandpass filter
Positive/Negative Ratio	$[1,\ 4]_{\Delta=1}$	Ratio between positive and negative samples
Use PCEN	$\{\text{True}, \text{False}\}$	Use PCEN
Time Constant	$[0.2,\ 0.8]_{\Delta=0.05}$	PCEN smoothing timescale
Gain	$[0.9,\ 0.995]_{\Delta=0.005}$	PCEN amplification/compression strength
Bias	$[0.5,\ 5.0]_{\Delta=0.25}$	PCEN floor to stabilize low energies
Power	$[0.3,\ 0.7]_{\Delta=0.05}$	PCEN exponent controlling compression

Table C.2: Model hyperparameter search space.

Model Parameter	**Search Space**	**Description**
ResNet Stacks	$\{1, 2, 3\}$	The amount of stacks in the ResNet Model
Pooling	$\{avg, max, None\}$	Type of pooling applied to the output of the model
L2	{1e-2, 1e-3, 1e-4, 1e-5}	Kernel regularization term
lr	{1e-2, 1e-3, 1e-4}	The initial learning rate
dropout	$[0, 0.5]_{\Delta=0.05}$	Dropout probability
loss	{BCE, FOCAL}	The used loss function
gamma	$[1, 4]_{\Delta=1}$	BCE parameter
alpha	$[0.1, 0.5]_{\Delta=0.05}$	BCE parameter

Table C.3: Training hyperparameter search space.

Training Parameters	**Search Space**	**Description**
Batch Size	$\{16, 32, 64\}$	Samples passed before optimizer update
Epochs Per Fold	$[30, 100]_{\Delta=10}$	Full passes over the training set per fold

Table C.4: Augmentation hyperparameter search space.

Augmentation Parameter	**Search Space**	**Description**
Loudness Jitter Probability	$[0, 1]_{\Delta=0.2}$	Probability to randomly change clip loudness
Max Jitter	$[-36, -6]_{\Delta=2}$	Upper bound of the loudness change (dB)
Min Jitter	$[-24, 0]_{\Delta=2}$	Lower bound of the loudness change (dB)
Gaussian Noise Probability	$[0, 1]_{\Delta=0.2}$	Chance to add gaussian noise
Max SNR	$[0, 20]_{\Delta=5}$	Highest target SNR for gaussian noise
Min SNR	$[15, 35]_{\Delta=5}$	Lowest target SNR for gaussian noise
SpecAugment Probability	$[0, 1]_{\Delta=0.2}$	Chance to apply time/frequency masking
Frequency Masks	$[0, 2]_{\Delta=1}$	Number of frequency-band masks to apply
Max Freq Mask Width	$[2, 64]_{\Delta=8}$	Maximum frequency mask width (Hz)
Time Masks	$[0, 2]_{\Delta=1}$	Number of time masks
Max Time Mask Width	$[2, 64]_{\Delta=8}$	Maximum time mask width (frames)

D

Pica pica Dataset Species List

Table D.1: Species counts by `primary_label`

Species	Count	Species	Count
Pica pica	14066	*Alopochen aegyptiaca*	959
Turdus merula	3285	*Podiceps cristatus*	912
Troglodytes troglodytes	2673	*Anthus pratensis*	910
Carduelis carduelis	2645	*Mareca penelope*	900
Aegithalos caudatus	2479	*Sturnus vulgaris*	895
Emberiza schoeniclus	2283	*Fulica atra*	888
Cyanistes caeruleus	2138	*Tadorna tadorna*	880
Erithacus rubecula	2028	*Garrulus glandarius*	852
Fringilla coelebs	2018	*Psittacula krameri*	819
Numenius arquata	2015	*Buteo buteo*	817
Branta leucopsis	1969	*Gallinula chloropus*	769
Parus major	1825	*Chroicocephalus ridibundus*	756
Prunella modularis	1814	*Falco tinnunculus*	732
Phylloscopus collybita	1679	*Anas platyrhynchos*	712
Vanellus vanellus	1621	*Bucephala clangula*	698
Saxicola rubicola	1560	*Columba oenas*	660
Branta bernicla	1506	*Tachybaptus ruficollis*	638
Tringa totanus	1352	*Ardea cinerea*	622
Anas crecca	1325	*Aythya ferina*	617
Dendrocopos major	1252	*Anas acuta*	578
Ciconia ciconia	1185	*Platalea leucorodia*	543
Branta canadensis	1166	*Cygnus olor*	483
Anser anser	1126	*Alcedo atthis*	478
Streptopelia decaocto	1108	*Aythya fuligula*	465
Columba palumbus	1075	*Passer domesticus*	461
Turdus philomelos	1070	*Spatula clypeata*	455
Limosa limosa	1063	*Mergus merganser*	334
Phalacrocorax carbo	1029	*Ardea alba*	110
Mareca strepera	1024	*Corvus corone*	43
Phasianus colchicus	962	*Egretta garzetta*	25