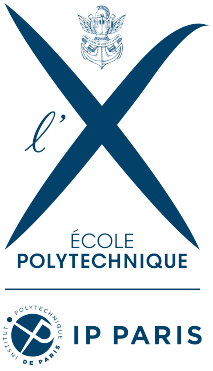


École Polytechnique

BACHELOR THESIS IN COMPUTER SCIENCE

Optimising Gunshot Detection in Tropical Forests for Wildlife Protection with TinyML

Author:

Alexandre Bismuth, École Polytechnique

Advisor:

Professor Alex Rogers, St Anne's College, University of Oxford

Academic year 2024/2025

Abstract

Unsustainable hunting is one of the leading factors of the acceleration of biodiversity loss, prompting the creation of wildlife protection laws in countries home to vast tropical forests. In this context, acoustic detection using low-cost audio loggers field recordings offers a practical way to monitor hunting pressure over wide areas. To advance such monitoring, we optimise a two-stage unconstrained pipeline that employs EfficientNets and Mel Spectrograms for gunshot detection in tropical forests, achieving a 5.72% improvement in F1 score and a 6.67% improvement in AUPRC over the current state-of-the-art. Despite these performance gains, the need for large storage hinders long-term monitoring and the absence of real-time capabilities limits the system's utility in law enforcement scenarios. To address these limitations, we leverage AI-oriented microcontrollers to provide real-time detection and immediate alerts. We notably provide an analytical framework to better understand the design of a compact architecture capable of capturing the distinctive features of a gunshot. Building on this foundation, we present a second two-stage pipeline featuring a smaller convolutional network that employs both model compression and knowledge distillation from the larger, optimised model. This compact process, deployable on an Arduino Nano 33 BLE Sense Rev2, achieves an F1 score of 0.840 and an AUPRC score of 0.849, offering near state-of-the-art accuracy while drastically reducing computational costs. Leveraging TinyML for on-device inference, our approach mitigates storage bottlenecks while enabling instant alerts, yielding a scalable, real-time solution that closes the loop between data collection and active law enforcement.

Contents

1 Introduction

1.1 Motivation

Unsustainable hunting in tropical forests is a leading driver of biodiversity loss. To prevent this practice from endangering or even causing the extinction of rare species, countries with tropical forests have enacted legislation to regulate these activities. However, enforcing these laws is challenging, as tropical forests are difficult to navigate and span vast areas. In light of the recent acceleration in biodiversity decline and the development of artificial intelligence, deploying automated systems to address this issue has become increasingly urgent [21].

Deep learning achieves higher accuracy than traditional machine learning methods by learning complex patterns and modelling long-range dependencies through deeper architectures. It also leverages transfer learning to adapt to changes in data without extensive retraining or modifications to the model architecture [24]. In parallel, as the field of audio classification evolves, enhanced models enable transformative applications—ranging from voice sentiment analysis to background recognition in virtual assistants and remote monitoring systems that detect sounds such as broken glass, fire alarms, or barking dogs. In this context, using deep learning to protect the biodiversity of tropical habitats appears promising, offering robust real-time detection capabilities. The most efficient approach would be to use embedded devices, allowing for the protection of wide areas at relatively low costs. Developing models that can fit within the resource constraints of these devices is therefore a priority, giving rise to the field of TinyML. In the interest of reducing costs, conducting a preliminary study using cheap audio loggers whose data would be analysed in-lab on an unconstrained model is also essential. It allows us to pinpoint the areas which are most subject to illegal hunting and thus considerably reduce costs by focusing our efforts on key regions subject to high hunting pressure.

Spectrogram-based convolutional neural networks have proved to be powerful approaches for robust gunshot detection on low-power devices [1] [17]. However, despite their potential, current solutions have several major flaws [15]. Most datasets rely on gunshots with minimum background noise and high signal-to-noise ratios, which is rarely representative of real-world acoustic conditions in tropical forests. Furthermore, unstudied background sounds such as a woodpecker drumming or a branch cracking can also be confused with a gunshot, triggering many false positives that would be problematic in a commercial application. Our dataset has the capacity to tackle these issues by using field recordings from a tropical forest in Belize, where heavy sound reflection from numerous natural obstacle attenuate the shock wave and elongate the muzzle blast of gunshots (details regarding components of a gunshot can be found in Section 3.1) [14]. It also contains varied background noises for the negative class that should help our model refine its understanding of gunshot audio signatures.

1.2 Contributions

In this study, we optimise an existing pipeline which takes advantage of CNNs to detect gunshot events within tropical forest [15]. As a commercial application for wildlife protection requires the development of a low-cost solution with real-time detection capability, we also develop TinyML models that can fit within low-power embedded devices. Our objective is thus to design a state-of-the-art teacher model with low computational constraints in order to optimise a limited student model. We focus performance on the minimisation of costly false positive events and false negatives which create risk-reward incentives for criminal hunters. Our contributions can be summarised as follows:

1. We provide an analytical framework to approximate a gunshot signal using a handcrafted ReLU networks. Our analysis proves that there exists a ReLU network capable of approximating our gunshot signal with any error $\varepsilon > 0$, and provides valuable insights for the development of TinyML models.

2. We optimise a gunshot detection pipeline for wildlife protection by experimenting with various preprocessing methods and model architectures. We achieve better performance than the current state-of-the-art model while keeping a similar level of pipeline complexity.

3. We develop a TinyML gunshot detection system for wildlife protection through models deployed on Arduino devices. To do so, we leverage various model compression techniques such as quantization and knowledge distillation via our state-of-the-art pipeline. We thereby achieve satisfactory performance while respecting microcontroller-induced constraints.

1.3 Methodology

1.3.1 Development environment

Our codebase was written in AWS SageMaker Studio through a JupyterLab environment with SageMaker Distribution 2.4.1 and 32GB of storage. We used an `ml.t3.medium` instance for development (2 vCPUs, 4 GB of RAM) and an `ml.g5.2xlarge` instance for training (8 vCPUs, 32 GB of RAM, NVIDIA A10G GPU).

The development platform used for Section 5 to experiment with a light gunshot detection prototype is an Arduino Nano 33 BLE Sense Rev2. It contains a 64 MHz Arm® Cortex®-M4F processor with 1 MB of flash memory and 256 kB of RAM. Details regarding its specifications can be found here.

All relevant code for this project is available on our GitHub repository, in which each directory corresponds to a section of this report.

1.3.2 Dataset

Our dataset is made of 747 gunshot sounds and over 35,233 background sounds that were collected in tropical forest sites in central Belize between 2017 and 2021 [14]. The files are in .wav format, and are each 4.00 seconds long. The first 8 characters of the file name corresponds to the UNIX hexadecimal timestamp of recording. Some files contain additional characters which were used as unique IDs during processing and do not convey any additional information. To address the dataset imbalance, we employ a weighted sampler for the training set that rebalances the class distribution.

A limitation of this dataset is its lack of a test set. Since our best models are saved based on their performance on the validation set, it is reasonable to expect that test scores would be lower than our current validation scores.

1.3.3 Performance metrics

To evaluate our models, we take the several considerations into account while choosing our performance metrics:

1. A viable system for a real-world application to prevent illegal hunting requires minimum false positives. One of the things we want to avoid the most is sending forest guards for a background noise.

2. An equally important concern is the minimisation of false negatives. Missing certain gunshots increases the risk to endangered species and creates a counterproductive game-theoretic incentive in the risk-reward trade-off for criminal hunters.

3. Given that our validation set has $\approx 98\%$ of negative labels, certain methods such as accuracy have limited value since consistently guessing a negative label already yields a $\approx 98\%$ accuracy.

Using these guidelines, we therefore choose the two following metrics:

1. **F1 Score**: A harmonic mean of Precision and Recall, balancing the trade-off between false positives and false negatives. It provides a more balanced evaluation than accuracy for datasets with class imbalance.

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = 2 \times \frac{\frac{TP}{TP+FP} \times \frac{TP}{TP+FN}}{\frac{TP}{TP+FP} + \frac{TP}{TP+FN}}$$

   When using this metric, we also define the optimised F1 score as the maximum F1 score achieved by tuning the decision threshold τ (which by default is set to $\tau = 0.5$) to its optimal value.

2. **Precision-Recall AUC**: Area under the plot of Precision ($\frac{TP}{TP+FP}$) and Recall ($\frac{TP}{TP+FN}$) for threshold τ. It only focuses on the classification of positive labels, our priority for gunshot detection applications.

$$\text{Recall} \mapsto \text{Precision}(\text{Recall}), \quad \text{AUC}_{\text{Precision-Recall}} = \int_0^1 \text{Precision}(\text{Recall})\, d\text{Recall}$$

   Using this curve, we can also find the precision-recall pair that maximises the F_1 score to refine τ.

2 Background

2.1 TinyML

Despite the recent advent of machine learning which made it ubiquitous in our daily lives, the computational power needed to run these new powerful models comes at a cost. Certain applications of machine learning imply the use of many edge devices that are run simultaneously to cover wide key areas such as tropical forests. This enables monitoring of endangered species populations, illegal hunting and logging prevention, wildfire surveillance, and much more. However, the need for numerous devices comes with financial limitations. Commercial applications for these use-cases therefore often rely on microcontrollers that have limited computational power and battery. It is notably the case of AudioMoth which enables long-term monitoring with low-power operation through onboard acoustic detection [11].

The need for machine learning models to run with such limited resources yielded the creation of TinyML, which attempts to develop lightweight models that enables low-power inference for prolonged monitoring. It is an actively research field, both on the hardware and software side. On one hand, specialised microcontrollers with integrated AI accelerators provide enhanced support for low-power computation. On the other hand, software frameworks are continuously refined to simplify the integration of machine learning on these devices. TensorFlow Lite Micro notably lightens models by quantizing them and restricting their use to an inference-only mode [7]. This allows for the removal of training-related variables, model checkpoints, and dynamic operations.

2.2 Expressive power of neural networks

The Universal Approximation Theorem states that any continuous function on a compact domain $f : K \to \mathbb{R}$ can be approximated arbitrarily well by a neural network. Nonetheless, it does not provide any information about either the network's architecture or how to determine it. Function approximation in the context of Deep Learning provides a mathematical framework that tackles that issue by studying the space of functions that can be approximated by different neural network architectures bounded in depth and width. Previous work notably came up with powerful bounds, proving that there exists a depth-6 ReLU network with $\frac{c}{\varepsilon \ln(1/\varepsilon)}$ weights, connections, and computations units (where c is an absolute constant) that can approximate functions in the unit ball of the Sobolev space $\mathcal{W}^{1,\infty}([0,1])$ with any error $\varepsilon > 0$ [28]. Details of this space and set of functions are explained in Appendix A.1.

In the context of TinyML, these results provide guidance for the design of neural network architectures under strict resource constraints. Although a perfect approximator might not be achievable in practice, its existence provides a reliable indication as to the order of complexity of our architectural design.

2.3 Model compression

Given the recent trend towards very deep models that require heavy computational resources, model compression has received growing interest, bridging the gap between these architectures and constrained edge devices. Model compression can help multiple use-cases. It can help scale down massive architectures like LLMs with initially hundreds of billion of parameters to run on smaller yet relatively powerful machines such as personal computers [26]. Similarly, it enables smaller models with initially a few millions of parameters to run on low-power devices, the key focus of TinyML. However, despite these varied objectives, compression techniques often remain similar across scenarios.

A simple yet effective way to perform model compression is quantization. This method attempts to make models more lightweight by reducing the precision of model weights and activation values. By downscaling their precision from `float32` to `float16`, `int8`, or even `int4`, we can significantly reduce the size of our model without an overly large compromise on performance [9].

Pruning is another powerful model compression technique. By setting weights, neurons, or filters to zero, optimised algorithm can significantly speed up inference while preventing overfitting [5]. This method does not actually reduce the size of the model but provides a virtual compression by allowing a low-power device to perform inference more efficiently on it. It offers a good compression-performance loss ratio but can be quite destructive at high scale. There are multiple ways to prune a model. One way is to use unstructured pruning, which sets weights to zero based on a criterion such as magnitude (i.e. weights with absolute value below ε are pruned), gradient (i.e. weights with the least impact on loss are pruned), or random selection. In contrast, structured pruning removes entire channels, filters, or layers. These methods can be used layer-wise or globally, during training or after it. Common methods include alternating pruning and training and using a dynamic pruning mask that is continuously updated during training.

Nonetheless, a consideration for unstructured pruning is the need for optimised algorithms and frameworks such as PyTorch Sparse. In practice, setting weights to 0 will have no impact on the inference speed if traditional algorithms are used. This is a very important consideration, especially in the case of TinyML which uses libraries implemented in C++ such as TensorFlow Lite Micro.

Last, another key concept linked to the concept of model compression is the idea of knowledge distillation. In this approach, a teacher model is first trained with little restrictions on computational resources. Then, a smaller student model is trained based on the combination of a traditional loss function and a distillation loss function which computes the difference between the prediction of the student and the teacher. This enables the student to mimic the behaviour of a near-perfect classifier, improving its performance in comparison with a traditional training approach [12]. Recent methods also combine distillation with other model compression techniques for combined performance gains [20].

2.4 Audio feature extraction

Neural networks can process a wide range of input features. The most common examples entail numerical features extracted from a tabular dataset and three-dimensional RGB pixels from flattened images. For audio classification, multiple options can be considered. The most straightforward approach is feed the raw waveform as a single-channel time series into the model. Combined with convolutional layers and max pooling, this

method enables deep networks to isolate specific events and recognise patterns in the waveform [6]. However, while this option preserves all the data from the original audio, its inability to explicitly differentiate between frequencies can limit the detection of certain sound signatures. To overcome this, many modern audio recognition pipelines apply transformations like the Fourier transform to generate spectrograms which are treated as grayscale images. This provides reliable information regarding the presence and amplitude of frequencies over time. There exists many types of spectrogram, with each method performing best in specific scenarios [29]. A comparative study of the most common preprocessing methods is made in Section 4.2.

Spectrograms are computed differently according to the specific feature extraction method used. Nonetheless, most approaches build on a parametrised Short-Time Fourier Transform (STFT) algorithm. Given a discrete signal, this method first segments the signal into overlapping frames of length N. As studying each window sequentially without any overlap could cause transient event to be missed, we also define a hop size H which adjusts the time shift between each window. Since discontinuities near the window's edge can cause spectral leakage, the signal is also multiplied by a window function w that smooths out the signal around its boundaries [16]. For each windowed segment, the algorithm then computes the Discrete Fourier Transform (DFT) using a Fast Fourier Transform (FFT) which returns the frequency components of each window, split into frequency bins. Each bin k provides both the amplitude and phase of that frequency component within the window.

2.5 Data augmentation

Another practical challenge of deep learning is its dependence on the quality of training samples [27]. Unfortunately, many problems lack diverse, trustworthy training datasets, which often leads to overfitting. This problem is extremely challenging in audio fields, where it is hard to collect large amounts of diverse recordings. To tackle that issue, data augmentation techniques such as SpecAugment have alleviated overfitting by introducing transformations to spectrograms from the training set [19]. These methods work by averaging over the orbits of the group that keeps the data distribution invariant, allowing for variance reduction [4]. A common data augmentation strategy in the context of spectrogram-based audio classification is the use of time and frequency masks. By randomly masking frequency bands and time intervals on training samples, we prevent our model from heavily relying on a single frequency to detect gunshot or limiting detection to specific time intervals, enhancing the model's generalisation capabilities.

3 Approximation of a Gunshot Signal using ReLU Networks

Expressive power of a neural network architecture varies depending on its depth, width, and activation functions. If a network architecture is too shallow or narrow, its approximation abilities can be highly limited. In addition to having shown great empirical results, ReLU networks—which use the activation function $\sigma(x) = \max(0, x)$—are widely used in machine learning theory for their advantageous properties [28] [18]. Understanding the ReLU-based architecture needed to capture the distinctive features of a gunshot sound can therefore provide valuable insights for designing a compact gunshot detection model. Throughout this section, we construct a ReLU network capable of approximating our gunshot signal with any error $\varepsilon > 0$. To do so, we build on previous work for functions $f : [0, 1] \to \mathbb{R}$ [28].

3.1 Synthesising a gunshot signal

The sound of a gunshot (given a shot directed towards the microphone) is divided in two parts : shock wave and muzzle blast. The former is due to the dispersion of air caused by the motion of the bullet and is comparable to an aircraft's sonic boom. This sound travels at bullet speed, which for rifles usually goes over Mach 2. The latter is caused by the explosion of the charge in the gun barrel and propagates through the air at the speed of sound. The figure below shows a gunshot signal originated from an M964 Light Automatic rifle (FAL), around

300 meters far from the recording position. The first portion of the signal refers to the shock wave and the following one to the muzzle blast.

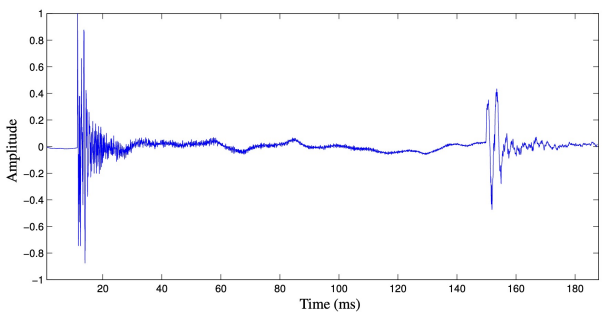


Figure 1: SW and MB originated from a rifle [2]

To evaluate the expressivity of our network, we must first define the signal that we want to approximate. Our goal here will be to recreate something as close as possible to Figure 1. Using this raw waveform, we define the signal approximation as follows. The domain restriction is induced by the fact that our following approximations will only be possible on the segment $[0, 1]$

$$f : [0, 1] \to \mathbb{R}, \quad x \mapsto e^{-200x} \sin(1500\pi(x - 0.01)) + 0.6 \times e^{-200x} \sin(500\pi(x - 0.15)) \tag{1}$$

Which has the following visual representation, given the same time interval as in Figure 1:

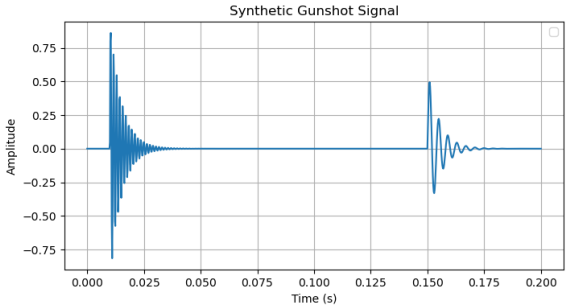
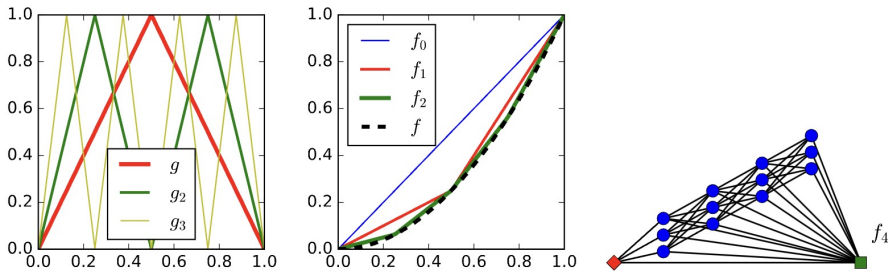


Figure 2: Synthetic SW and MB originated from a rifle

3.2 Constructing a ReLU network capable of approximating our gunshot signal

3.2.1 Setting up elementary functions

The first step to approximate our gunshot signal is to reproduce previous work for elementary functions [28]. We define a "tooth" function $g : x \mapsto 2\sigma(x) - 4\sigma(x - \frac{1}{2}) + 2\sigma(x - 1)$, and enable it to be composed by itself s times as $g_s(x) = g \circ g \circ \cdots \circ g(x)$ to get a sawtooth function. We then use formula $f_m(x) = x - \sum_{s=1}^{m} \frac{g_s(x)}{2^{2s}}$ to approximate the square function with any error $\varepsilon > 0$ on the segment $[0, 1]$ by interpolating it at 2^m+1 points.



(a) (b) (c)

Figure 3: Approximation of the function $f(x) = x^2$: (a) the "tooth" function g and the iterated "sawtooth" functions g_s; (b) the functions f_m approximating f with varying errors; (c) the network architecture for f_4 [28].

Using this result, we can then approximate the multiplication function as $xy = \frac{1}{2}((x+y)^2 - x^2 - y^2)$. This then allows us to approximate polynomials, enabling a reconstruction our gunshot signal through Maclaurin series. For these approximations, parameter m (taken from f_m) will refer to the degree of precision of the approximation. A precision $m = k$ implies that the elementary square function is interpolated at $2^k + 1$ points.

3.2.2 Approximating the negative exponential and sine functions

Now that we can approximate polynomials with ReLU networks easily, we can use Maclaurin series in Landau notation (also called Big-O notation) to reconstruct e^{-x}. Building this function will be essential, as combined with out power function, we will be able to approximate with any error $\varepsilon > 0$ the e^{-200x} component of our gunshot signal.

Indeed, using Maclaurin series, we can derive the formulas for $x \mapsto e^{-x}$ and $x \mapsto \sin(x)$. We get

$$e^{-x} = \sum_{n=0}^{\infty} \frac{(-x)^n}{n!} \quad \text{and} \quad \sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

Combining this formula with Landau notation, we can construct a ReLU network approximating both functions and their product with error $O(x^{10})$. Precision $m = 4$, introduced in Section 3.2.1, is then chosen to allow for a visual difference between the original and approximation functions while ensuring low error rates.

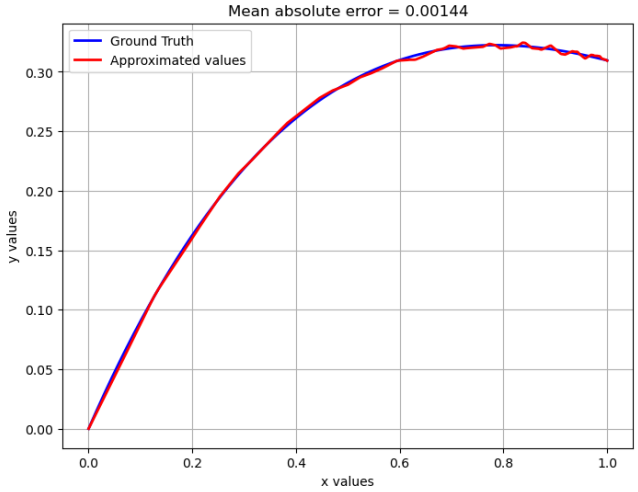


Figure 4: Approximation of $x \mapsto e^{-x}\sin(x)$ with $m = 4$ and error $O(x^{10})$.

Using our previously defined function, we can now reconstruct our final signal. Since ReLU networks enable linear combinations, the only requirement left to build Equation 1 is the construction of $x \mapsto e^{-ax}\sin(bx+c)$ with arbitrary constants $a, b, c \in \mathbb{R}$. To do so, we can approximate e^{-ax} and $\sin(bx+c)$ separately before using our multiplication function.

1. Combining our e^{-x} approximation with our power function, we can build $x \mapsto e^{-ax}$ for any constant a
2. Since we can only approximate functions $f : [0,1] \to \mathbb{R}$, our approximation of $\sin(ax+b)$ is slightly more complex. Using periodicity of the sine function, we can use multiple Taylor series (instead of Maclaurin series) centred at $x = 0, 1, ..., 6$ to construct a period of sine. Since ReLU networks enable shifts and rescaling, we can use this period to approximate $x \mapsto \sin(bx+c)$ for any constants b, c

3.3 Architectural takeaways

Building on our approximations, we can now attempt to recover indication regarding how our network should be designed based off of these approximations. Using our approximation of the function $x \mapsto e^{-x}\sin(x)$ with $m = 3$ and $O(x^5)$, we get a network of 864 neurons with width 3 and depth 288 that is capable of approximating our function with a $\approx 0.5\%$ error rate. This suggests that the expressive power of depth should enable us to develop a reliable gunshot detection system, even if our theoretical analysis is subject to many practical limitations explained in the following section.

3.4 Limitations of theory

Even though we are able to reconstruct our gunshot signal with a handcrafted ReLU network, proof of existence does not imply that an appropriate network architecture will be able to learn this approximation. This issue is easily reflected in the n-ap problem, which describes a discrete function alternating between 0 and 1 n times in the interval $[0,1]$. As shown in Figure 5, a solution with 3 hidden layers can be handcrafted to approximate our sawtooth function g_3 from Section 3.2.1, but cannot be learnt by the appropriate neural network architecture due to the difficult loss landscape.

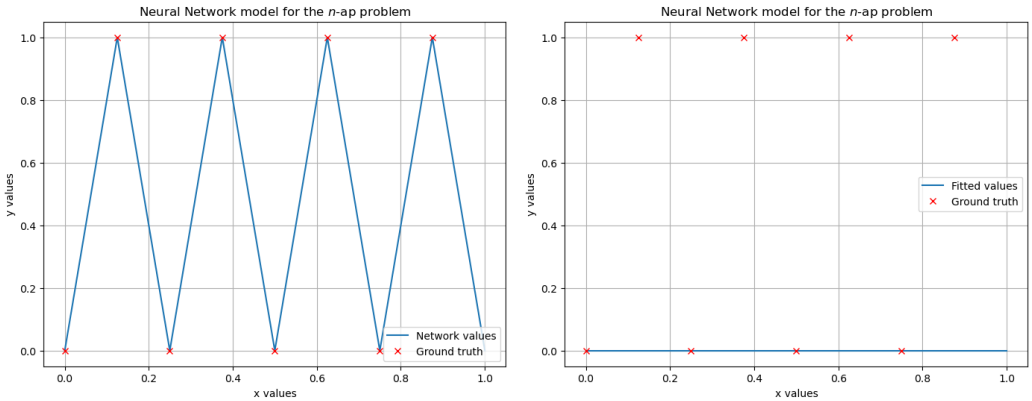


Figure 5: Handcrafted solution (left) and learnt solution (right) for the n-ap problem

In addition, although depth enables precise function approximation in theory, it is sometimes a hindrance in practice. Training a deep neural network with 288 layers will yield bad performance due to a *degradation* problem [10]. Even though the architecture we obtained from our handcrafted solution should give us a good indication in terms of number of neurons, the information it conveys on depth is therefore limited. In practice, a relatively shallower but wider network should be more effective.

4 Optimising a State-of-the-Art Gunshot Detection Model

The design of a state-of-the-art gunshot detection pipeline without major computational constraints is a fundamental step in our study. Firstly, having access to an optimised model also allows for better results on our TinyML model through improved model distillation [12]. Furthermore, in the context of a commercial application for wildlife protection in tropical forest, this model also allows for preliminary studies on the presence of illegal hunting in different areas. By covering a vast areas with audio loggers over a large period of time, we are able to pinpoint the areas which are most subject to illegal hunting. This then allows for an efficient placement of embedded devices with real-time detection capacities, optimising the cost-benefit ratio of our solution.

4.1 Updating and improving the existing pipeline

Previous work on this dataset already should give us a good starting point for the design of an optimised detection pipeline [15]. However, the versioning of the previous project combined with its lack of documentation on dependency versions has rendered its models unusable. The pipeline relied on OpenSoundscape 0.5.0, which is now incompatible with essential libraries like PyTorch. This lack of information, coupled with the use of a version of ResNet18 which is no longer supported, thus makes it necessary to rebuild the pipeline using updated libraries and fixed documented versioning. To achieve this, we first replicate the original pipeline with OpenSoundscape 0.12.0, before creating a new pipeline with Torchaudio—which offers greater modularity. In addition to maintaining the pipeline's operability, this change enables experimentations with recent techniques and models for further improvements.

To go from the audio to the classification prediction, the current pipeline starts by converting the Waveform Audio File (WAV) to a spectrogram by using a Discrete-Time Short-Time Fourier Transform (DT STFT). Using a discrete signal $x[n]$, which in our case has a 8 kHz sample rate, the DT-STFT algorithm explained in Section 2.4 is computed as:

$$X(m,k) = \sum_{n=0}^{N-1} x[n+mH]w[n]e^{-i\frac{2\pi kn}{N}}$$

Here, m and k denote the frame and frequency bin index, while H, N, and w respectively represent the hop size, window length, and the window function. In our pipeline, we set $N = 256, H = 128$, and w to be the Hanning window $w[n] = 0.5(1 - \cos(\frac{2\pi n}{N-1}))$.

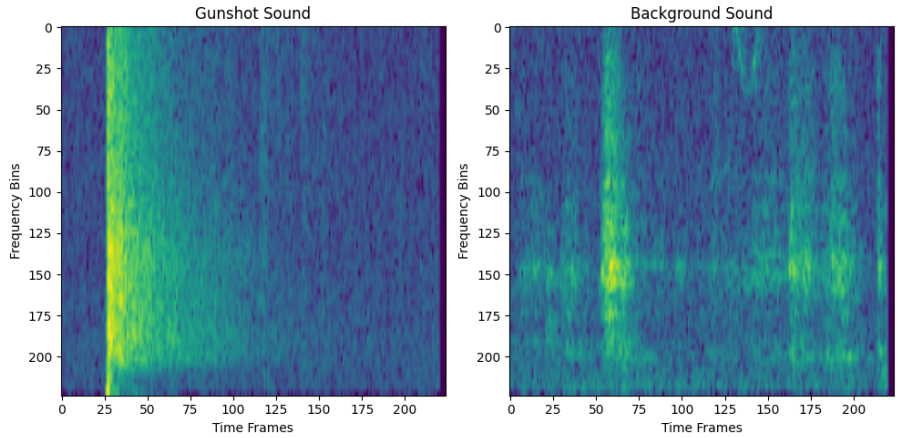


Figure 6: Example DT-STFT spectrograms for a gunshot and background audio

Then, our pipeline performs data augmentation to improve the generalisation capacity of our model [27]. Based on previous work, we choose to narrow the frequency range, add time and frequency masks, as well as add noise

to our spectrogram [19]. We therefore restrict our analysis to frequencies between 0 and 2 kHz, randomly apply between 0 and 20 masks along the time and frequency dimensions (each with a maximum size of 5% of the respective dimension) using a uniform distribution, and add background noise drawn from a normal distribution with a standard deviation of $\sigma = 0.005$. The preprocessed spectrogram are then finally fed into a neural network to predict whether the original audio was a gunshot or a background sound. In our pipeline, we use a modified version of ResNet18 for single-channel inputs (greyscale images) and binary classification [10].

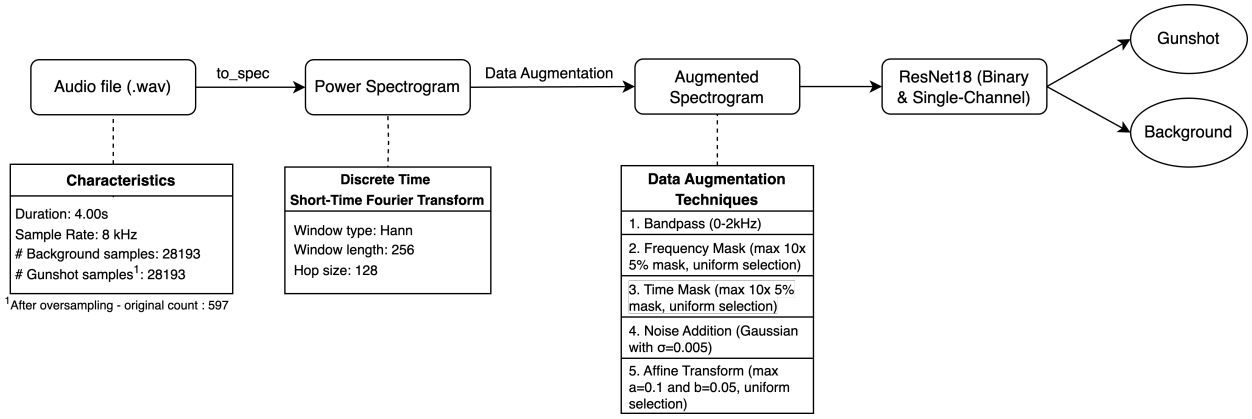


Figure 7: Current Gunshot Prediction Pipeline

Using this method, we reproduce the original results within an order of magnitude of 10^{-3}, likely due to minor training randomness. Our model achieves an optimised F1 score of 0.895 and an AUPRC of 0.934. Moreover, the confusion matrix confirms that given our few false positives and false negatives, a smaller model capable of retaining a similar performance would reveal useful in commercial applications.

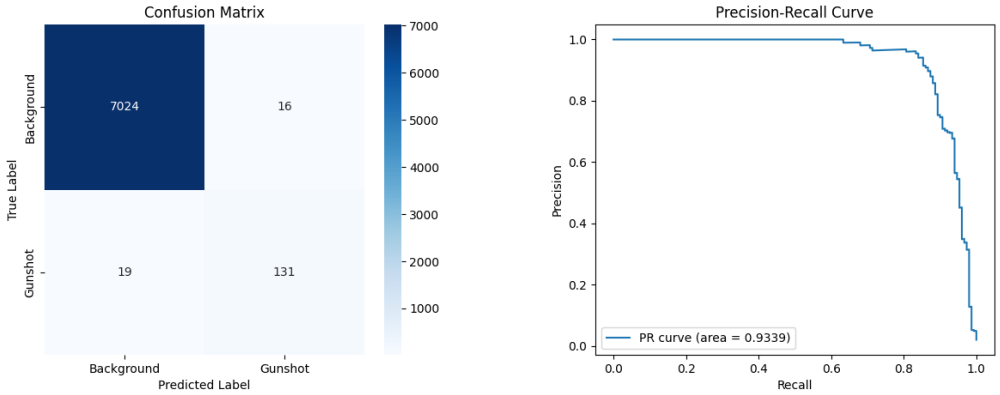


Figure 8: Confusion matrix (left) and precision-recall curve (right) for gunshot detection model

4.2 Comparing various feature extraction methods

There are many ways to treat audio data in the context of Machine Learning. A comparative study of preprocessing techniques can therefore be conducted to assess whether the STFT method used in the original pipeline is optimal. Given the lack of modularity of OpenSoundscape which only provides a couple of different preprocessing techniques, we choose to first rebuild the pipeline with Torchaudio. This allows for greater modularity through the `torchaudio.transforms` module and better compatibility with the rest of the PyTorch library. Although the new pipeline closely mimics the behaviour of the original one, we have to replace the random affine transform data augmentation with a random time shift due to a lack of support on Torchaudio.

4.2.1 Raw waveforms

The simplest way to classify our audio data is to leave it in the form of a raw waveform. This will enable our model to consider it as a simple time series, with each input value being treated as the amplitude of the signal at a given time. Then, thanks to the convolutional and pooling layer, our model should be capable of detecting sudden changes in amplitude and distinguish the audio signature of a gunshot.

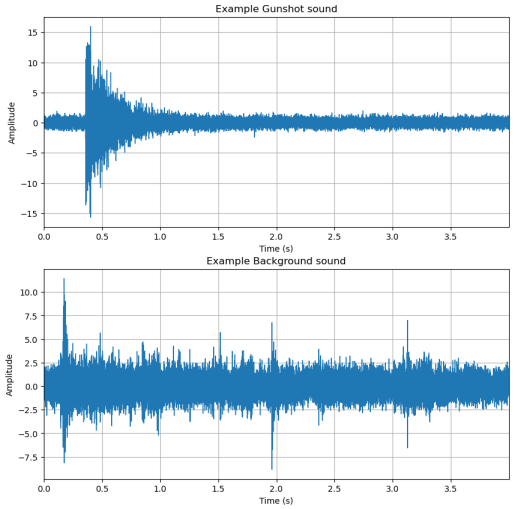


Figure 9: Augmented Raw Waveform Samples

This naive method, which is often disregarded in most analyses, yields a very satisfactory F1 score of 0.834 and AUPRC of 0.894. This surprising performance likely comes from the fact that gunshot are violent sudden occurrences. While it is typically difficult to obtain a viable waveform-based model for tasks requiring precision at low amplitudes like voice or instrument recognition, the convolutional layers in our model enable it to detect the sudden change in amplitude and frequency triggered by a gunshot. Nonetheless, it is important to note that this approach may be limited, as it might only learn to detect loud events rather than the nuanced waveform patterns of a gunshot. Distant gunshots are thus more likely to be confused with ambient noise while loud background events, such as branches cracking near the microphone, are likely to trigger false positives.

4.2.2 Power spectrograms and LFCCs

Going back to an augmented Discrete-Time STFT preprocessing method explained in Section 4.1, we can construct a power spectrogram. Using this method, we obtain a baseline F1 score of 0.852 and a AUPRC of 0.884. Even though this is does not obtain the same performance as the original OpenSoundscape pipeline which uses the same approach, this provides us with an essential comparison point which will allow us to determine the best preprocessing method given the same environment. The difference in performance likely comes from the slight difference in data augmentation induced by the change in library. This means that even though Torchaudio is better suited for research experimentations, a final pipeline could benefit from being designed with OpenSoundscape.

From these power spectrogram, we can also compute the Linear Frequency Cepstral Coefficients (LFCCs), which have revealed powerful in certain audio classification tasks [8]. Given a spectrogram, we start by applying triangular linear filters along the frequency axis to smoothly capturing the spectral energy distribution of the signal. After filtering, we compress the dynamic range by taking the logarithm of the filter bank energies. Finally, a Discrete Cosine Transform (DCT) is applied on the outputs to decorrelate the features and compact the energy into a smaller number of coefficients.

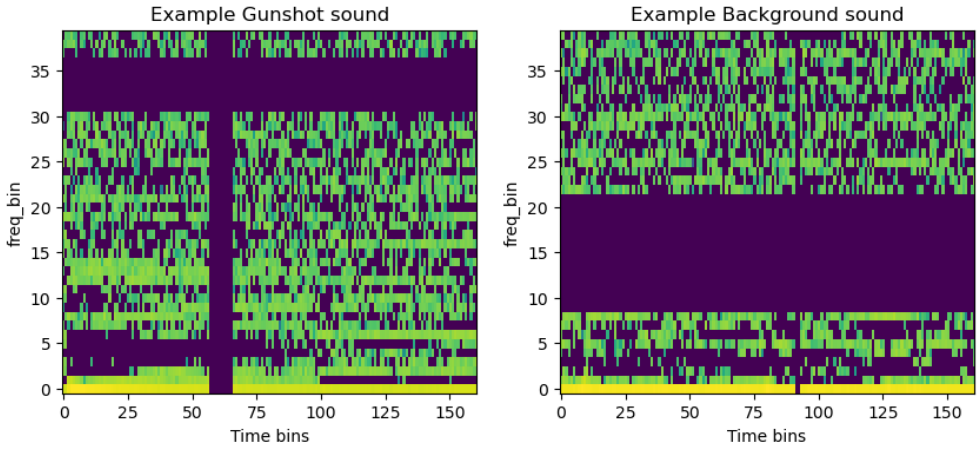


Figure 10: Augmented Sample with LFCC Preprocessing

LFCCs allow us to capture the spectral energy distribution in a compact yet informative way. Using this method, we obtain an F1 score of 0.842 and an AUPRC of 0.903.

4.2.3 Mel spectrograms and MFCCs

A major flaw of power spectrograms in frequency ranges close to the sounds that we hear best is their linear scale. On them, a difference of 200 Hz is always represented in the same way. Comparing it to music notes, this means that the difference between a C2 and a C4 will be the same as the difference between a G6 and an A6. However, human ears have a different way of perceiving sounds. Given our ability to distinguish lower sounds better, the difference between C2 and C4 (16 tones) will appear much bigger than the difference between G6 and A6 (1 tone). This is because our ear uses a logarithmic scale to compare audio signals instead of a linear one. From this distinction, we can therefore use Mel spectrograms, which rescales power spectrograms to a perceptually meaningful scale. Since gunshots typically have a frequency of about 500 Hz, this scale should enable our model to better distinguish them from background noises.

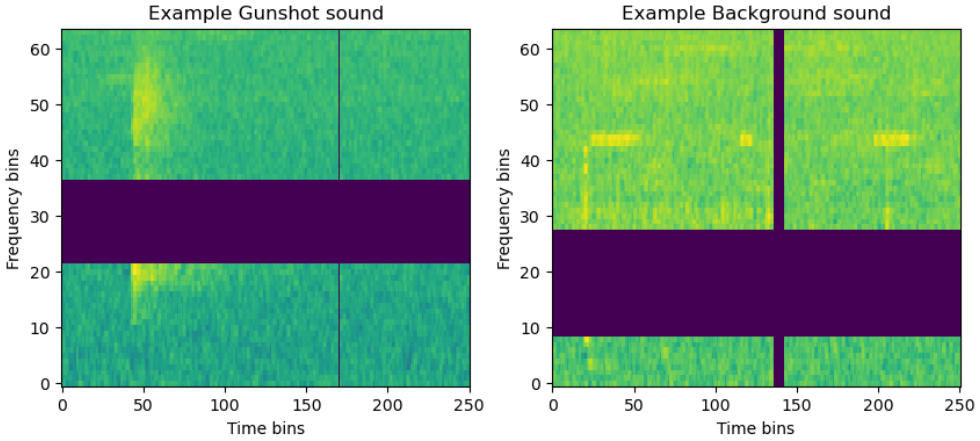


Figure 11: Mel Filterbank (left) and Augmented Mel Spectrogram Samples (middle, right)

This method allows for an improved performance compared to power spectrograms, obtaining an F1 score of 0.869 and an AUPRC of 0.890. Similarly to LFCCs, MFCCs can also build on Mel spectrograms to provide a compact and informative way of representing the spectral information of the audio signals. This approach achieves an AUPRC score of 0.890 and an F1 Score of 0.825.

4.3 Evaluating different model architectures and design adjustments

Model architecture can also greatly impact the performances of a model. In the original pipeline, a ResNet18 adjusted for a single channel (greyscale image) and binary classification was chosen due to its empirical ability to produce accurate predictions on most image-based classification tasks. However, given the specific nature of gunshot detection, this choice of architecture could reveal sub-optimal. Some adjustments, such as kernel dimension changes, can reveal useful given the violent perturbation induced by a gunshot. Given these considerations, we proceed to compare a series of models and architectures as well as slight design changes in an attempt to improve the performance of our pipeline. All experimental results can be found in Table 2.

4.3.1 Residual networks

Today, Residual Networks (ResNets) are still considered as very powerful image classifier. By combining convolutional layers with identity connections that propagate information throughout the network, this allows the model to recognise subtle patterns while dealing with the training degradation [10].

There exists multiple version of ResNets, each with different depth. The smallest model integrates 18 layers while the biggest one available on PyTorch accumulates 152 layers. Beyond computational constraints which are not a consideration in this section, the most important factor to choose a ResNet is data quantity and quality. A model that is too small may be limited by its architecture and underperform by being unable to distinguish subtle patterns, while overly large models also underperform due to an inability to optimise trainable parameters given a lack of data or poor data quality [3].

Design adjustments to the model's architecture can also allow models to accommodate for the specificity of a given task. One of the most effective changes in ResNets is the modification of the kernel dimensions within the convolutional layers. On one hand, large kernels detect prominent features and enable shallow networks to receive information from the integrality of the image faster, although this comes at a precision trade-off. On the other hand, smaller kernels better detect fine-grain details, at the cost of needing higher depth to receive global information on the studied picture. One can also modify the kernel ratio, to put an emphasis on one axis of the picture. This can notably be useful for detecting gunshots, as increased attention to the time axis helps capture sudden changes in spectral energy.

Using this intuition, we experiment with different variants of ResNets. A first experiment studies the impact of kernel size on ResNet18, showing that smaller kernels yield better performance. This supports our claim that the localised nature of gunshots make smaller kernels more able to identify fine details. A second experiment also confirms our intuition for kernel dimensions. Using a 3×15 initial kernel instead of 7×7, our model is able to improve its capacity to detect sudden changes in spectral energy. However, pushing this idea further, we nonetheless see that extreme choices such as a 1×15 initial kernel generates worse results. Regarding depth, we find that ResNet50 is best capable of capturing fine-grain details. It provides a more potent architecture than ResNet18 and ResNet34 while avoiding the overfitting issues seen in ResNet101.

4.3.2 Efficient networks

Traditional networks typically scale up by increasing depth. This was notably observed in our use-case, as ResNet50 performs better than its shallower counterparts. However, this process is not truly understood and a lot of trial and error is required to find a near-optimal solution. To solve that issue, newer architectures such as EfficientNets propose a way to systematically scale up the depth, width, and image resolution of CNNs [23]. Given a scaling coefficient ϕ, depth d, width w, and resolution r, we impose:

$$\alpha \times \beta^2 \times \gamma^2 \approx 2, \text{ with } d = \alpha^\phi, w = \beta^\phi, r = \gamma^\phi, \text{ and } \alpha, \beta, \gamma \geq 1$$

Here, α, β, and γ are constants determined by grid search, while ϕ, typically ranging from 0 to 7, determines the size of the network, with $\phi = k$ denoting EfficientNetB-k.

In our experiments, we test out different values of ϕ. We observe that while EfficientNetB3 performs better than EfficientNetB2, EfficientNetB4 already suffers from over-parametrisation and underperforms in comparison with EfficientNetB3. Furthermore, in comparison with ResNets, EfficientNets reveal much more powerful, outperforming the older architecture in all performance metrics. However, in comparison with ResNets, it appears that changing the dimensions of the initial kernel negatively affects performance. This is likely due to the difference in architecture which can induce different optimal parameters.

4.4 Results

Our comparative study of preprocessing methods allow us to find an approach that outperforms the power spectrogram technique used in the current pipeline. After experimenting with various function parameters and selecting those with the best empirical results, we observe a 2.33% and 2.15% improvement in F1 score and AUPRC when using LFCCs. This can be explained by the LFCC's linear filter bank, which improves the spectrogram's ability to capture sudden changes in energy that might be smoothed out in a power spectrogram.

Preprocessing Method	**Function Parameters**	**F1 Score**	**Optimised F1 Score**	**AUPRC**
Raw Waveform	Sample Rate $= 8$ kHz	0.834	0.840	0.894
Power Spectrogram	$N = 512,\ H = 256,\ w =$ Hanning	0.852	0.856	0.884
LFCC	$\#\text{Filters} = 128,\ \#\text{LFCCs} = 40$	0.842	**0.876**	**0.903**
Mel Spectrogram	$N = 512,\ H = 128,\ \#\text{Mels} = 64$	**0.869**	0.873	0.890
MFCC	$N = 1024, H = 512, \#\text{Coefs} = 32$	0.824	0.825	0.890

Table 1: Performance of Various Preprocessing Methods with ResNet18 for Gunshot Detection

Thanks to our experimentations with various model architecture and design adjustment, we also find a model that significantly outperforms the ResNet18 from the original pipeline. With EfficientNetB3, we notably provide a 5.72% increase in F1 score and a 6.67% increase in AUPRC. This is a significant improvement—especially in wildlife protection applications, where minor errors incur substantial costs.

Model Architecture	Design Adjustment	F1 Score	Optimised F1 Score	AUPRC
ResNet18	Default	0.836	0.839	0.884
ResNet18	5×5 Initial Kernel	0.823	0.852	0.885
ResNet18	3×3 Initial Kernel	0.857	0.863	0.880
ResNet18	3×7 Initial Kernel	0.851	0.860	0.862
ResNet18	3×15 Initial Kernel	0.857	0.866	0.880
ResNet18	1×25 Initial Kernel	0.790	0.790	0.830
ResNet34	Default	0.842	0.848	0.892
ResNet50	Default	0.868	0.878	0.897
ResNet101	Default	0.853	0.874	0.891
EfficientNetB2	Default	0.846	0.871	0.904
EfficientNetB3	Default	**0.871**	**0.887**	**0.943**
EfficientNetB3	3×7 Initial Kernel	0.844	0.861	0.908
EfficientNetB3	3×15 Initial Kernel	0.851	0.868	0.924
EfficientNetB4	Default	0.814	0.838	0.906

Table 2: Performance of Various Model Architectures for Gunshot Detection using Mel Spectrograms

From these results, another interesting analysis lies in the study of the time between misclassifications. Predicting false positives within a short time frame could suggest that specific background noises such as a woodpecker or a thunderstorm are consistently confused with gunshots by our model. Conversely, a series of consecutive false negatives could indicate that certain rifles categories or combinations of gunshots with particular weather conditions go undetected, suggesting that our model should be retrained on a more diverse dataset. Reassuringly, using the UNIX timestamps present in the file names, our model obtains good results for both false positives and false negatives. Even though the mean time between occurrences is skewed by the four-year gap in our data collection, the median values demonstrates the robustness of our model. With a median time of several hours between false positives and half an hour between false negatives, both concerns are alleviated.

Type of Error	Mean Time Delta	Median Time Delta
False Positives	79 days, 13 hours, 30 minutes, 5 seconds	3 hours, 19 minutes, 29 seconds
False Negatives	46 days, 2 hours, 27 minutes, 8 seconds	30 minutes, 11 seconds

Table 3: Time Analysis of Detection Errors using EfficientNetB3

Taken together, our findings indicate that our final pipeline is robust and would most benefit from the use of EfficientNetB3 combined with an LFCC preprocessing method. However, considering OpenSoundscape's enhanced data augmentation, its lack of LFCC support, and the fact that Mel spectrograms perform nearly as well as LFCCs, we opt for this approach in our final pipeline design.

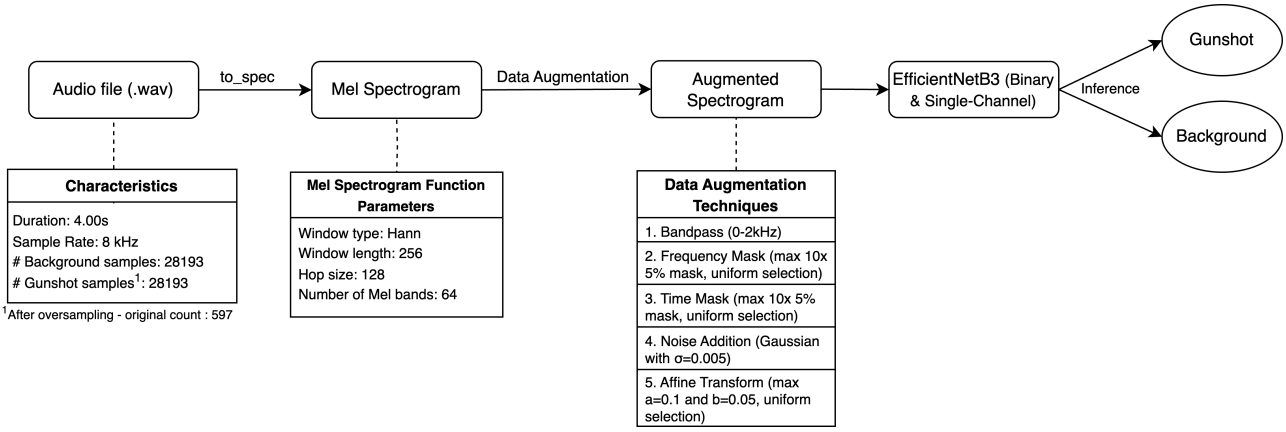


Figure 12: Optimised Gunshot Prediction Pipeline

Using this approach, our model provides a maximal improvement in F1 score and AUPRC in comparison to the previous OpenSoundscape pipeline. Beyond refining the current system for gunshot detection in tropical forests, this result enables us to improve the effectiveness of an in-lab preliminary study while providing an enhanced teacher model essential to Section 5.1.

5 Building a TinyML Application for Biodiversity Protection via Model Compression

As shown in our comparative study, the most performant models for spectrogram classifications are typically very deep convolutional neural networks that take up tens of megabytes of storage and can require gigabytes of RAM to perform inference. However, with only a few megabytes of flash memory and hundreds of kilobytes of RAM on average, embedded devices are not able to run these large models. Building on the theoretical analysis as well as the teacher model provided by the optimised pipeline, we thus aim to develop a smaller model that is capable of retaining a near state-of-the-art performance. In the context of a commercial application for biodiversity protection law enforcement in tropical forests, this model would allow for real-time detection and instant alerts to local authorities when combined with AI-oriented connected embedded devices.

As explained in Section 1.3.1, our prototypes leverage the Arduino framework through the Nano 33 BLE devices. Alongside it, we use TensorFlow Lite Micro, which lightens our models by restricting their use to inference and removing training-related variables as well as dynamic operations. Given the limited compatibility between PyTorch and TensorFlow, we however have to rewrite the existing pipeline. Having concluded our comparative studies, we can disregard TensorFlow's lower adaptability and focus on implementing a straightforward pipeline without worrying about modularity. This process nonetheless requires us to remove random affine transforms from the data augmentation pipeline due to a lack of TensorFlow support.

5.1 Distilling the knowledge of a teacher model

To help our TinyML model retain most of the accuracy of their bigger counterparts, another option is Distillation. Given a performant teacher model and a compressed student model, this process performs a forward pass on both networks before computing a parametrized distillation loss. This new term is then added to the traditional loss with a weighting factor λ. This gives us the following loss formula:

$$\mathcal{L} = \mathcal{L}_{\text{hard}}(y, p_s) + \lambda T^2 \left(\sigma\left(\frac{z_t}{T}\right)_{\text{Background}} \log \frac{\sigma\left(\frac{z_t}{T}\right)_{\text{Background}}}{\sigma\left(\frac{z_s}{T}\right)_{\text{Background}}} + \sigma\left(\frac{z_t}{T}\right)_{\text{Gunshot}} \log \frac{\sigma\left(\frac{z_t}{T}\right)_{\text{Gunshot}}}{\sigma\left(\frac{z_s}{T}\right)_{\text{Gunshot}}} \right)$$

After experimentations and given the context of binary classification, we also choose to set the temperature parameter T—which softens the output probability distribution of the softmax function σ—to $T = 3$.

Although it may seem counter-intuitive to train models on teacher-generated labels that are less precise than the ground truth, this approach has empirically proven highly effective at boosting the accuracy of small models. [12]. Distillation enables the student to mimic the behaviour of the teacher by learning to recognise the same patterns, which is often easier than learning directly from true labels that include outliers and particularly challenging examples. In practice, the Keras library accelerates distillation using a dedicated class that implements distillation loss with minimal effort. After loading both the teacher and student models, the latter is retrained based on its new loss function, enhancing its generalisation capabilities despite moving away from the ground truth. Nonetheless, in our case, a preliminary required step is the recreation of the teacher model in a Keras format using TensorFlow. We thus re-instantiate and train an adapted EfficientNetB3 model, achieving the same performance as in the PyTorch pipeline up to a 10^{-2} precision. Note that, as with PyTorch, we have to remove affine transforms due to lack of support. Proceeding with distillation in our TinyML gunshot detection system, this method yields a considerable F1 improvement of 12.0% and an AUPRC improvement of 5.6%.

More recent methods that combine distillation with other compression techniques in an attempt to retain more performance during model compression. This is notably the case of Quantized Distillation, which aims to combine the memory savings offered by quantization with the performance benefits provided by knowledge distillation [20]. At each iteration, given full-precision weights w, this approach first computes the traditional and distillation loss on the quantized weights w^q before updating the weights w with full-precision. Finally, after the last iteration, the full-precision weights are quantized one last time and returned as w^q. This approach makes w^q converge to the minimal quantized losses while preserving information that would have been lost by standard quantization, leading to better accuracy in comparison to naive post-training quantization.

Algorithm 1 Quantized Distillation [20]

Let w be the network weights
loop
  $w^q \leftarrow$ quant_function(w, s)
  Run forward pass and compute distillation loss $l(w^q)$
  Run backward pass and compute $\frac{\partial l(w^q)}{\partial w^q}$
  Update original weights using SGD **in full precision** $w = w - \nu \cdot \frac{\partial l(w^q)}{\partial w^q}$
Finally quantize the weights before returning: $w^q \leftarrow$ quant_function(w, s)
return w^q

In practice, after experimenting with quantized distillation on our TinyML gunshot detection pipeline, we see no signifiant improvement in performance in comparison to traditional methods. This could be explained by the relatively shallow architecture we use, which may not fully leverage the benefits of quantized distillation. Additionally, since Mel spectrograms compress and transform the raw audio data, optimising the quantized losses instead of the regular losses might be counter-productive.

5.2 Designing and deploying TinyML models

An important challenge when designing TinyML resides in determining the architecture of the uncompressed model. Even though it is not necessarily the case in unconstrained environments, bigger models generally

perform better in the context of TinyML. The aim should therefore be to have a compressed model that fits exactly within the computational constraints of the embedded device. Since there is no exact method to predict how much space is freed during compression, the space savings from compression, we adopt a trial-and-error approach: we design the architecture, compress the model, observe the savings from compression, adjust the model's size to maximise its potential, and start over. Despite its inefficiency, this method allows us to pinpoint the best architecture and compare the impact of various design choice on our gunshot detection model. In our case, after experimenting with different configurations of the pipeline using a method similar to grid search, we find that the optimal model should aim to have close to 115 thousand parameters in our compressed model before conversion to the TensorFlow Lite format.

Similarly, another challenge resides in the RAM constraints of our Arduino Nano 33 BLE Sense Rev2. As stated before, all computations have to stay underneath the 256 kB RAM threshold to be run on our device. To ensure this is the case, we use an open-source customised Python script for computing TensorFlow Lite RAM requirements.

5.2.1 Proposed architecture and compression pipeline

With these considerations in mind, we come up with an architecture tailored to the Arduino Nano 33 BLE Sense Rev2. Our model leverages a handcrafted convolutional neural network inspired from other audio classification spectrogram-based models and adjusted iteratively according to empirical results in order to maximise performance while staying within the computational constraints of our Arduino platform. It is made of 5 convolutional blocks of kernel size 3×3 and max pooling size of 2×2. Additionally, each block includes a batch normalization layers and dropout layers which masks 40% of the weights during training. In our design, the number of convolutional filters used is also doubled between each blocks—going from 8 to 128. After being passed through these blocks, the output is then flattened into a vector before being inputted into a width 8 dense layer. These last neurons are then subject to a 50% dropout layer and combined into a single sigmoid-activated output responsible for classifying the input as "Gunshot" or "Background". With 113,585 parameters, our model approaches the 115,000 limit without exceeding it, yielding a near-optimal model size.

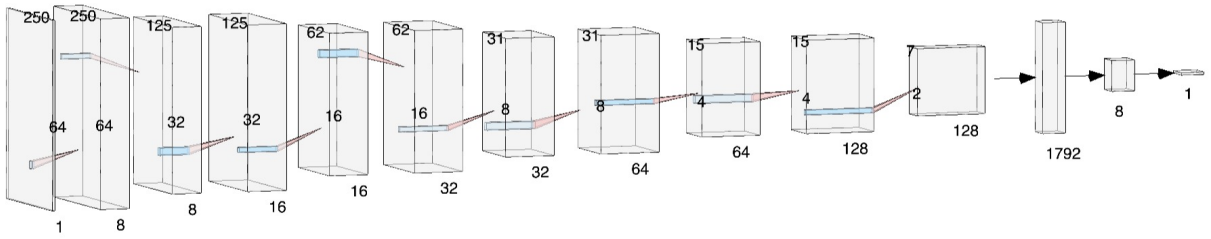


Figure 13: Architecture of our TinyML Convolutional Neural Network

In our architecture, the combination of convolutional and max pooling layers should allow our model to extract patterns and downsample the feature map by focusing on the most important feature. Doubling filter size between every block also early layers to focus on simple patterns while allowing deeper layers to learn very abstract patterns. In addition, batch normalization adjusts the scaling of activations within a layer, stabilising and speeding up training by reducing internal covariate shift [13]. Furthermore, dropout layers reduce overfitting by setting random activation outputs to zero, ensuring that the model doesn't rely too heavily on specific neurons [22]. Lastly, the flatten and dense layers transform the feature maps into a vector, combining the detected patterns to produce a final prediction.

Following the creation and training of this model, we can convert it into a Tensorflow Lite format using a built in converter. After evaluating various options through a grid search, we quantize the weights to `int8` format and we leave the activation functions to `float32` format. This allow us to compress the model by 91.4%. Given the model in TensorFlow Lite format, the last requirement to make it compatible with Arduino devices is conversion to a C array. Since AWS uses a Linux environment, we take advantage of the xxd tool. This final method yields a C header (.h) file of 782 kB containing the length of our model as well as its array representation. This result is a 46.1% compression relative to the original Keras model, achieving a near-optimal model size as discussed in the following section.

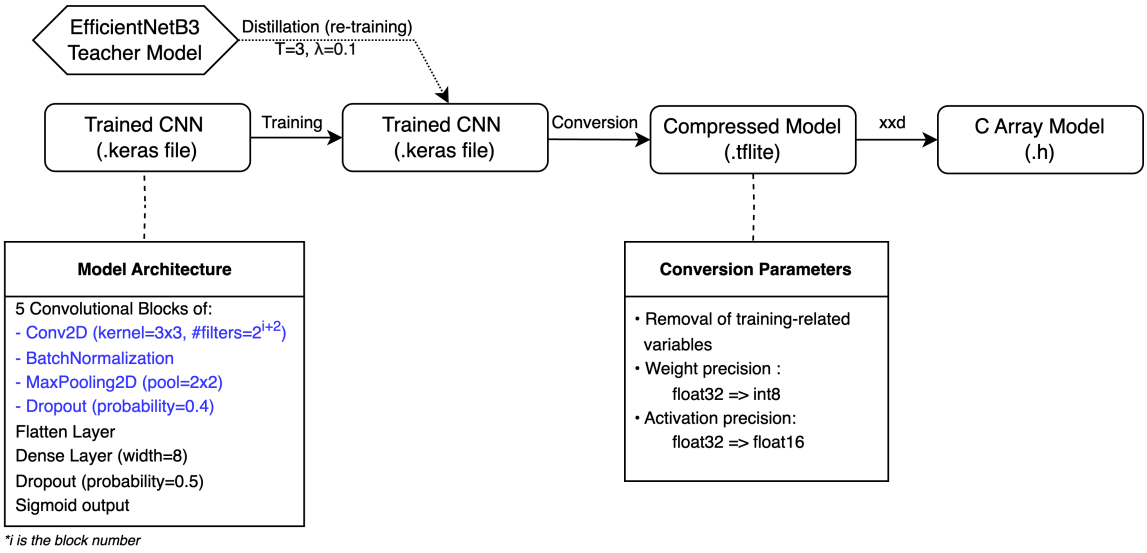


Figure 14: Model Compression Pipeline

Using the available Python script, we also confirm that this model fits within the computational constraints of an Arduino Nano 33 BLE Sense Rev2 limited to 1 MB of flash memory and 256 kb of RAM. This script provides us with an approximation of the RAM usage at every step of inference. It observes a peak usage of 250 kB over the first convolutional block, suggesting that we are successfully maximising the potential of our Arduino device.

5.2.2 Arduino integration

Building on the Arduino TFLite library and built-in tutorials, we also define a C++ environment of about 75 kB capable of performing inference on our array model. Combined with our gunshot detection model in a C array format and the required space to run system scripts on our device which occupy about 100 kB of flash memory, our project maximises memory usage. Reminding that we had also maximised RAM usage with a 250/256 kB peak RAM requirement, our model is therefore optimised in every respect. Compiling it and running it locally, we observe that our pipeline is error-free, fits within the computational constraints, and is capable of generating predictions. It detects background sounds almost continuously and recognises many (but not all) gunshots sounds from the original dataset via an external speaker despite the additional ambient noise and interference introduced by the speaker.

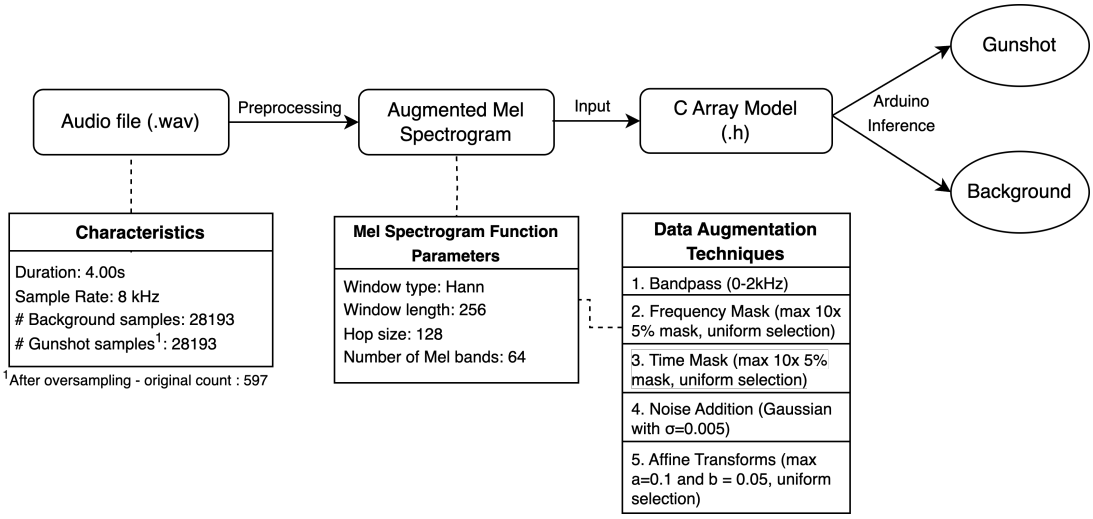


Figure 15: Arduino Inference Pipeline

5.3 Results

In summary, we have trained, distilled, and compressed a convolutional neural network of near-optimal size and deployed it for a TinyML application using an Arduino Nano 33 BLE Sense Rev2. We review each step of the compression and inference pipeline to describe and interpret our results.

To start, our convolutional model achieves an F1 score of 0.750 and an AUPRC of 0.804 after the first training, a satisfactory starting point for our compression pipeline. Here, one option would have been to use the TensorFlow Model Optimization library which provides a way to perform structured and unstructured pruning on our model in a progressive way while continuously retraining the pruned model. However, since this method has not yielded any satisfactory result—degrading both performance metrics by over 5% while yielding only marginal savings—we decide to remove it from our training pipeline.

Then, we continue our training process by leveraging knowledge distillation. Using an EfficientNetB3 model and a distillation loss described in Section 5.1, we observe significant improvements.

Model	**F1 Score**	**AUPRC**
Teacher Model - EfficientNetB3	0.876	0.885
TinyML Student Model - Pre-Distillation	0.750	0.804
TinyML Student Model - Post-Distillation	0.840 ($\uparrow$ 12.0%)	0.839 ($\uparrow$ 5.6%)

Table 4: Impact of Knowledge Distillation on Performance

This result can be explained by the trickling capabilities of distillation which enables the student to learn to detect the same patterns as the teacher. The considerable increase in F1 score, when put in comparison with the AUPRC, also suggests that distillation enable better balance between precision and recall at threshold $\tau = 0.5$.

Following distillation, our compression pipeline provides great memory savings, which enable our model to fit within the computational constraints of our Arduino device.

Model	Occupied Disk Space
Teacher Model - EfficientNetB3	48.322 MB
TinyML Model - Keras Format	1.468 MB
TinyML Model - Tensorflow Lite Format	126 kB ($\downarrow 91.4\%$)
TinyML Model - C Header File Format	792 kB ($\downarrow 46.1\%$)

Table 5: Memory Savings Achieved through the Compression Pipeline

Lastly, after importing our model on the Arduino software, we can conclude by running the inference pipeline. This transforms our device in a gunshot detection system that leverage its microphone to build Mel spectrograms and run our handcrafted convolutional neural network. Despite several mistakes in a error-prone environment with ambient noise and distortion from an external speaker, our Arduino generates background predictions almost continuously and classifies most gunshots sounds accurately.

6 Discussion

In summary, our study provided an analytical framework capable of approximating a gunshot signal with any error, optimised an unconstrained gunshot detection pipeline for wildlife protection in tropical forests, and proposed a TinyML gunshot detection model capable of fitting within Arduino devices.

Despite promising results in our TinyML application, we still believe that further performance improvements are necessary before advancing to a commercial application. As stated throughout the study, false positives incur heavy costs while false negatives create a risk-reward incentive for criminal hunters. To manufacture a credible device for enforcing wildlife protection laws, we estimate that performance should first be improved by close to 10% before being integrated into embedded devices. This would allow our TinyML models to achieve F1 and AUPRC scores above 0.950, which would be satisfactory for a commercial application.

Nonetheless, our results suggest that developing a gunshot detection system for wildlife protection in tropical forests should be viable. With TinyML models that can fit within microcontrollers, we can imagine partnerships with governments or local authorities in an attempt to better enforce biodiversity protection laws. Moreover, our further optimisation of the state-of-the-art gunshot detection pipeline also indicates that a preliminary study using cheap audio loggers can reliably identify high-risk areas, reducing the costs of deploying a large fleet of embedded devices with real-time capabilities across vast regions.

6.1 Limitations of our study

Our experiment have a few limitations. To start, our results are isolated to tropical forests. From an acoustic standpoint, the dense vegetation present in these areas acts as an obstacle, generating heavy sound reflections and attenuating the sound wave. Compared to a clean gunshot sound in an isolated environment, the initial shock wave can barely be heard in tropical forests. This makes the audio signature of a gunshot quite different, especially when compared to events such as a long-distance gunshot in a savannah environment. Due to the difference in speed between the shock wave and the muzzle blast as explained in Section 3.1, the audio signature of the gunshot becomes a two-time event instead of one. Models for such applications would therefore be different to ours, limiting the versatility of our model.

Another limitation of our work also resides in our lacks of a test set. Due to the dataset's relatively small size, adding a test set would have a overly large impact on performance. This nonetheless comes at a potential overfitting trade-off. Since our best models are kept based on their performance on the validation set, it is possible that our model overfits the validation data and would in reality achieve slightly worse test scores.

A last practical limitation of our work arises from our libraries. Due to the absence of LFCCs on OpenSoundscape, our pipeline does not leverage the best preprocessing method, but only the second best. This suggests that our results might have been even better if OpenSoundscape had supported LFCCs or if the choice of data augmentation techniques was as extensive on Torchaudio as it is on OpenSoundscape.

6.2 Future work

Looking ahead, this work opens up multiple paths for further research. In particular, three key directions appear promising: a theoretical bound for our network's architecture, exploration of alternative preprocessing methods and model designs, as well as a comparative study of TinyML microcontrollers.

Firstly, future work could focus on the creation of a bound for our gunshot approximation network. Given the current modelling of a gunshot signal, most bounds on the network's architecture cannot be applied [28]. Despite the smoothness of our function and its derivatives, its high frequency excludes it from the unit ball $F_{1,1}$ of the Sobolev space $\mathcal{W}^{1,\infty}([0,1])$ (c.f. Appendix A.1). Using recent work on function approximation and optimisation could therefore allow us to design better architectures by providing guarantees on the generalisation capacity of our network and convergence analyses through bounds on the error reduction per training iteration.

Secondly, a study of different architectures and machine learning models could reveal useful to further optimise our gunshot detection pipeline. Regarding preprocessing methods, another option that could be explored is the use of raw spectrograms in their complex-valued form. Current investigations in field of complex neural networks suggest that these architectures could reveal useful in the case of spectrograms [25]. This process notably carries more information than power spectrograms, since squaring the magnitude of the Fourier transform output can eliminate subtle information present in the original signal.

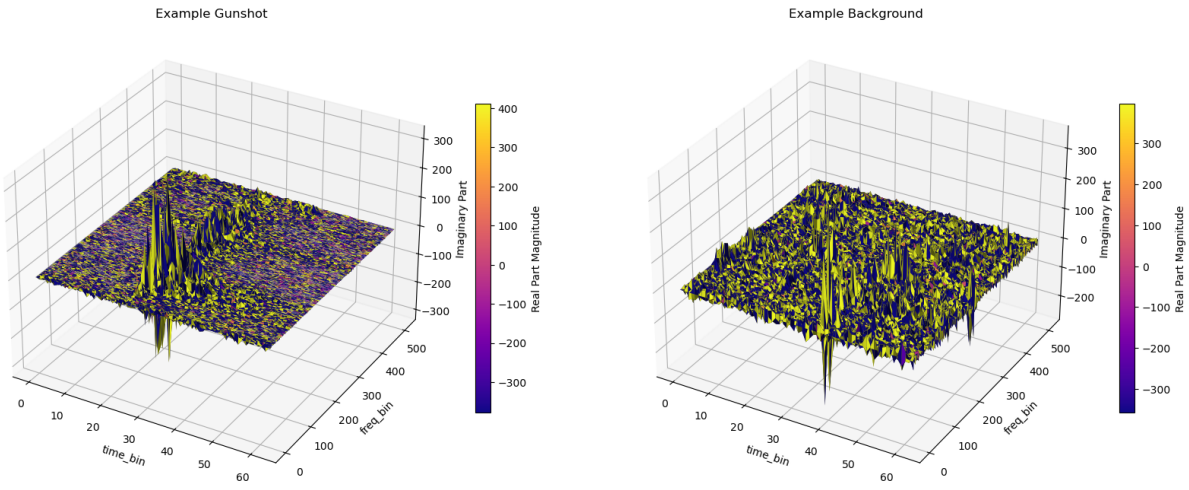


Figure 16: Complex-valued Spectrogram Representation of a Gunshot and Background Sound

Lastly, exploring alternative devices for our TinyML models could prove beneficial in future work. In the interest of minimising costs, finding the cheapest processor capable of running our TinyML models would be essential. This requires a thorough study of the current microcontroller market and an efficient way to approximate the memory required to run and store our models. This study would also allow us to focus on wide-scale products such as AudioMoth that would be a better fit for a commercial application [11].

7 References

[1] Jakub Bajzik, Jiri Prinosil, and Dusan Koniar. Gunshot detection using convolutional neural networks. In *2020 24th International Conference Electronics*, pages 1–5. IEEE, 2020.

[2] Angelo MCR Borzino, José A Apolinário, Marcello LR de Campos, and Carla L Pagliari. Gunshot signal enhancement for doa estimation andweapon recognition. In *2014 22nd European Signal Processing Conference (EUSIPCO)*, pages 1985–1989. IEEE, 2014.

[3] Lorenzo Brigato and Luca Iocchi. A close look at deep learning with small data. In *2020 25th international conference on pattern recognition (ICPR)*, pages 2490–2497. IEEE, 2021.

[4] Shuxiao Chen, Edgar Dobriban, and Jane H Lee. A group-theoretic framework for data augmentation. *Journal of Machine Learning Research*, 21(245):1–71, 2020.

[5] Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[6] Wei Dai, Chia Dai, Shuhui Qu, Juncheng Li, and Samarjit Das. Very deep convolutional neural networks for raw waveforms. In *2017 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 421–425. IEEE, 2017.

[7] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Tiezhen Wang, et al. Tensorflow lite micro: Embedded machine learning for tinyml systems. *Proceedings of Machine Learning and Systems*, 3:800–811, 2021.

[8] Sita Purnama Dewi, Anggunmeka Luhur Prasasti, and Budhi Irawan. The study of baby crying analysis using mfcc and lfcc in different classification methods. In *2019 IEEE International Conference on Signals and Systems (ICSigSys)*, pages 18–23. IEEE, 2019.

[9] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. In *Low-power computer vision*, pages 291–326. Chapman and Hall/CRC, 2022.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[11] Andrew P Hill, Peter Prince, Jake L Snaddon, C Patrick Doncaster, and Alex Rogers. Audiomoth: A low-cost acoustic device for monitoring biodiversity and the environment. *HardwareX*, 6:e00073, 2019.

[12] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[13] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[14] Lydia Katsis, Andrew Hill, Evelyn Piña-Covarrubias, Peter Prince, Alex Rogers, C Patrick Doncaster, and Jake Snaddon. Tropical forest gunshot classification training audio dataset, March 2022.

[15] Lydia KD Katsis, Andrew P Hill, Evelyn Pina-Covarrubias, Peter Prince, Alex Rogers, C Patrick Doncaster, and Jake L Snaddon. Automated detection of gunshots in tropical forests using convolutional neural networks. *Ecological Indicators*, 141:109128, 2022.

[16] Douglas A Lyon. The discrete fourier transform, part 4: spectral leakage. *Journal of object technology*, 8(7), 2009.

[17] Alex Morehead, Lauren Ogden, Gabe Magee, Ryan Hosler, Bruce White, and George Mohler. Low cost gunshot detection using deep learning on the raspberry pi. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 3038–3044. IEEE, 2019.

[18] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[19] Daniel S Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D Cubuk, and Quoc V Le. Specaugment: A simple data augmentation method for automatic speech recognition. *arXiv preprint arXiv:1904.08779*, 2019.

[20] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668*, 2018.

[21] Udo Schickhoff, Maria BOBROWSKI, I Offen, and Suraj MAL. The biodiversity crisis in the anthropocene. *Geography and the Anthropocene*, pages 79–111, 2024.

[22] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[23] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[24] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI global, 2010.

[25] Chiheb Trabelsi, Olexa Bilaniuk, Ying Zhang, Dmitriy Serdyuk, Sandeep Subramanian, Joao Felipe Santos, Soroush Mehri, Negar Rostamzadeh, Yoshua Bengio, and Christopher J Pal. Deep complex networks. *arXiv preprint arXiv:1705.09792*, 2017.

[26] Wenxiao Wang, Wei Chen, Yicong Luo, Yongliu Long, Zhengkai Lin, Liye Zhang, Binbin Lin, Deng Cai, and Xiaofei He. Model compression and efficient inference for large language models: A survey. *arXiv preprint arXiv:2402.09748*, 2024.

[27] Shengyun Wei, Shun Zou, Feifan Liao, et al. A comparison on data augmentation methods based on deep learning for audio classification. In *Journal of physics: Conference series*, volume 1453, page 012085. IOP Publishing, 2020.

[28] Dmitry Yarotsky. Error bounds for approximations with deep relu networks. *CoRR*, abs/1610.01145, 2016.

[29] Khalid Zaman, Melike Sah, Cem Direkoglu, and Masashi Unoki. A survey of audio classification using deep learning. *IEEE Access*, 11:106620–106649, 2023.

A Appendix

A.1 Sobolev space $\mathcal{W}^{1,\infty}([0,1])$ and its unit ball $F_{1,1}$

To understand the mathematical framework of the $\mathcal{W}^{1,\infty}([0,1])$ Sobolev space, let us first recall the definition of sets of measure 0 and L^∞ spaces.

A set is said to be of measure zero if it can be covered by intervals of arbitrarily small total length. Formally, a set $S \subseteq \mathbb{R}^n$ has Lebesgue measure zero if $\forall \varepsilon > 0$, there exists a countable number of open sets $\{U_i\}$ such that

$$S \subseteq \bigcup_{i=1}^{\infty} U_i \quad \text{and} \quad \sum_{i=1}^{\infty} \text{vol}(U_i) < \epsilon.$$

where $\text{vol}(U_i)$ denotes volume. In our example, $S \subseteq \mathbb{R}$ implies that U_i's volume is its length on the real line.

Furthermore, a function is said to be essentially bounded if $\exists M \in \mathbb{R}$ such that $|f(x)| \leq M$ for every $x \in [0,1]$ except on a set of measure zero. We denote essential supremum (ess sup) the smallest M satisfying this property. Then, a function f is measurable if, $\forall \alpha \in \mathbb{R}$, the set $\{x \in [0,1] \mid f(x) > \alpha\}$ is Lebesgue measurable, meaning it can be constructed from open sets using countable unions, intersections, and complements. Unlike continuity, measurability allows for discontinuous functions if they are well-behaved with respect to measure.

Combining these notions, $L^\infty([0,1])$ is the space of essentially bounded measurable functions $f : [0,1] \to \mathbb{R}$.

Now, in Section 2.2, to be able to apply our bound, we have to consider the Sobolev space $\mathcal{W}^{1,\infty}([0,1])$ defined as

$$\mathcal{W}^{1,\infty}([0,1]) = \Big\{ f : [0,1] \to \mathbb{R} \;\Big|\; f \text{ is weakly differentiable, } \|f\|_{L^\infty([0,1])} < \infty, \text{ and } \|f'\|_{L^\infty([0,1])} < \infty \Big\}.$$

We also define the unit ball $F_{1,1}$ lying in that space as $F_{1,1} = \{f \in \mathcal{W}^{1,\infty}([0,1]) : ||f||_{\mathcal{W}^{1,\infty}([0,1])} \leq 1\}$ which uses the norm defined as

$$\|f\|_{W^{1,\infty}([0,1])} = \max\Big\{ \text{ess sup}_{x\in[0,1]} |f(x)|,\ \text{ess sup}_{x\in[0,1]} |f'(x)| \Big\}$$