Linnæus University
Sweden

Technical Report

Gunfire detection with machine learning on ESP32

ESP32 - TensorFlowLite Micro

Author:	Saleh Shalabi
	Robin Svensson
Email:	ss225bx@student.lnu.se
	rs223dj@student.lnu.se
Professor:	Mehdi Saman Azari
Course:	2DT304
Semester:	VT23

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Abstract

The objective of this project was to create a device that can detect and alert on the occurrence of gunfire using machine learning. A binary classifier model was trained on data that either was labeled as 1 meaning the audio contains gunfire sound, or 0 non-gunfire sound, where it predicts probability of either the sound contains gunfire or not. The data was preprocessed by resampling to 16,000KHz and creating a spectrogram of each sound that was then resized to 128x64 and rounded to two decimal places. A convolutional neural network (CNN) was trained on the preprocessed data, achieving an accuracy of 99% on the test set. The model was then converted to TensorFlow Lite and deployed on an ESP32 microcontroller with a KY-037 sound sensor. When a gunshot is detected, the device sends a LoRa message.

This project provides a proof-of-concept for a low-cost and reliable gunshot detection system that can be deployed in public areas to help reduce gun violence. The system can potentially be scaled up and integrated with existing public safety infrastructure to provide real-time alerts to law enforcement and emergency services, enabling a faster response time. The use of machine learning also opens up possibilities for further improvement and customization of the detection system. However, privacy concerns and false positives must be carefully considered and addressed before wider deployment. Overall, this project demonstrates the potential of technology to address pressing social issues and highlights the importance of interdisciplinary collaboration between computer science and public safety domains.

Abbreviations

Short	Full
LoRa	Long-Range Low-Power
TTN	The Things Network
ADC	Analog-Digital Converter
FFT	Fast Fourier Transformation
STFT	Short-Time Fourier Transformation
CNN	Convolutional Neural Network
GAN	Generative Adversial Networks
RNN	Recurrent Neural Networks
ReLU	Rectified Linear Unit
RTOS	Real-time Operating System
IDE	Integrated Development Environment
MCU	Microcontroller Unit
PCM	Pulse Code Modulation

1 INTRODUCTION

This chapter will go over the background of the project, what sparked the interest and decision to go for a problem in this sector. It will also define the problem and propose a overview of the contents within the report. There will also be an explanation of the purpose and limitations that was taken in regards to the problem definition. Also within the background information a short explanation of the intended target group.

1.1 Background

With a rising amount of gun violence [1] and scared citizens there seemed to be a lack of potential information gathering and alarm oriented solutions to get fast and factual information regarding this issue. Therefore this project idea was created to start working towards as a solution that could help both the general population but also potentially other related authorities as police force, or emergency services by getting immediate information in case such a situation occurs.

1.2 Problem definition

The project aims to address the problem of identifying gunshots in real-time by leveraging machine learning algorithms. The proposed solution involves capturing incoming audio and processing it through a trained model to determine if the sound was a gunshot or something else. Once the model detects a gunshot, the device sends a packet to the TTN LoRa network indicating the occurrence of a gunfire. This information can then be collected and analyzed by a potential server, which can notify end-users and authorities.

The primary goal of the project is to develop an effective and efficient gunshot detection system that can provide quick and reliable information to relevant stakeholders in real-time. The proposed solution offers several advantages over traditional gunshot detection methods, including fast processing, automated data transmission, and cost-effectiveness.

1.3 Limitations

Regardless the proposed solution's potential for usability, effectiveness and impact it could provide, there are some limitations, including the need for sufficient data to train the machine learning model, potential false positives or false negatives, and the reliance on a LoRa network for data transmission. On the hardware side, the device requires sufficient memory to store the model, audio data, and relevant spectrogram for the audio data. Furthermore, there is a need for a good microphone sensor that can collect and record clear data. Despite these limitations, the project has the potential to significantly improve public safety by providing an advanced gunshot detection system that can be easily deployed in various settings.

1.4 Purpose and research question / hypothesis

The purpose of the report is to investigate the possibility of using the existing technology in microcontrollers today to read and classify audio samples in a real time system. By exploring this possibility, the future in the field can do further work and produce a working product that could benefit society. The hypothesis based on prior knowledge in the area indicates that there is a possibility of making a device fit for this purpose.

Research questions:

- Are microcontrollers capable of continuously read audio data, preprocess it in 3-second intervals using overlapped windows of 2 seconds from the previous sample?
- Using 3 seconds audio samples could it be classified without disturbing other processes?

1.5 Target group

While the target group of the project can be characterized as "everyone," a closer examination reveals a potentially significant interest from the police departments, municipalities and emergency services in gaining access to the data for the purpose of reducing response times for emergency vehicles. Moreover, the data could serve as a promising foundation for future efforts aimed at issuing warnings and identifying shooting locations.

1.6 Outline

The report will be organized into several chapters, each of which examines a different aspect of the project. First, a chapter will provide an overview of related works that have been done in this field, and how this project compares to the others. Next, a theory chapter will clarifies background and necessary knowledge to comprehend how the project operates. A hardware chapter will follow the theoretical chapter, detailing the components utilized in the project. The following chapter is about the software, specifying what technologies, languages and libraries employed to build this project. The report will then present the results and include a discussion section. Finally the report will conclude summarizing challenges encountered during working on the project, addressing research questions and outlining potential avenues for future work.

1.7 Project overview

The system is implemented on a Heltec Wireless Lite board, which features an ESP32 microcontroller and provides both LoRa and WiFi capabilities. For this project, only the LoRa functionality is used. A KY-037 sound detection sensor is also employed to capture and process the audio signals. Further details about these components will be provided in the hardware section of the report.

Overview perspective of the device:

- Continuously listen to audio
- Sample three second of audio
- Send the sampled audio data to algorithm
- If gunshot detected, send necessary data through LoRa
- Otherwise, remove first second and take the next sample.

2 RELATED WORK

This chapter will go over the works that have been done using similar techniques but for other hardware and overall feasibility of machine learning on a micro controller compared to stronger devices.

2.1 Others work

2.1.1 *Gunshot Detection from Audio Streams in Portable Devices [2]*

A master thesis written on the feasibility of using machine learning algorithms in portable devices, comparatively with the one used in this project the utilized device was significantly stronger. For example their memory usage ended up being 50 times greater than the amount of RAM that the device for this project had available. This laid the foundation of this project and opening up the possibilities of trying to find a solution that would work on a device that was much weaker.

2.1.2 *Low Cost Gunshot detection using Deep learning on the Raspberry Pi [3]*

The research was conducted utilizing a Raspberry Pi 3, outfitted with an external USB microphone, in an outdoor gun range setting. This preliminary investigation demonstrated promising outcomes, effectively establishing the project's methodological framework. However, the challenge remains to adapt the approach for devices with significantly lower processing power and memory capacity, as well as incorporating an analog microphone instead of the USB version. In this adaptation, it is necessary to work with three seconds of audio data at a 16000Hz frequency, compared to the one-second sample at a 441000Hz frequency employed in the current research.

2.1.3 *Machine Learning for Microcontroller-Class Hardware: A Review [4]*

This paper was related more to the theoretical approach of using machine learning on micro controllers and how it could be done. It went over some information about speech recognition but that was only based on 10 samples widths compared to the 128 sample widths that was required for the sampling rate of the audio clips in this project. This turned out to be a big problem regarding RAM usage later in the project.

2.2 Comparison

Upon examining the existing literature in this field, it becomes apparent that there are limited projects available for review. Among those that were found, almost all used 25 ms to 50 ms worth of sound for classification. Comparing to this project requiring three seconds window to obtain a reliable evaluation of the model. The similar projects as "Gunshot Detection from Audio Streams in Portable Devices" or "Low Cost Gunshot detection using Deep learning on the Raspberry Pi" required more robust hardware capabilities than this project's device. Although a few projects have successfully conducted this type of classification on mobile devices, there are scarce examples of microcontrollers employing audio recognition.

3 THEORY

This chapter will discuss the theoretical background necessary to comprehend the decisions and thought processes involved in the project. The discussion will be divided into two main sections: audio and machine learning.

3.1 Audio

3.1.1 *Audio waves*

The microphone used in this project provides analog values, making it necessary to understand how to measure the characteristics of audio waves. The project primarily focuses on working with the frequency and amplitude of the received audio waves.

Amplitude refers to the magnitude of an audio wave relative to its reference value, measured within a single period of the wave. The peak value of the wave represents the amplitude that has been recorded.

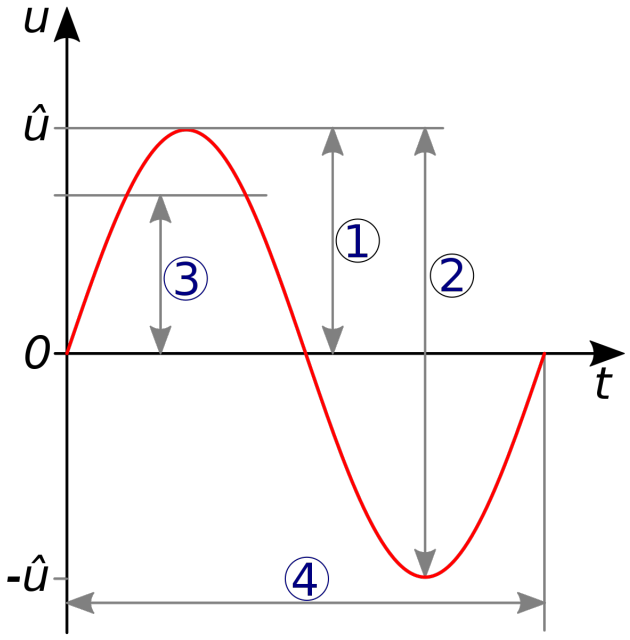


Figure 1: Peak-Peak Amplitude

Frequency is measured by time between periods, where the frequency is calculated by equation 1 to understand how this is used in the project, it is necessary to understand the frequency bands that is frequently used and which ones are important for gun detection. According to data from the Noise Monitoring Services states that most gunshot lay within in the 2-3KHz band of frequencies [5]. This information establishes the requirements for the microphone and micro-controller in the project.

$$f = \frac{1}{T} \qquad \text{Frequency} \qquad (1)$$

From previous information it should be understood that the device needs to be able to have a *sampling rate* that is higher than that frequency band. However, due to the nature of waveforms and their periods, the required sampling rate must be twice as high as the maximum frequency being recorded, as calculated using the Nyquist rate *see equation 2*. This is explained by Nyquist's theorem that states the need of twice as high sampling rate than the frequency due to the nature of waves having positive and negative parts in their periods. If the sampling rate is lower than twice the maximum frequency, it will not accurately capture the peaks of each period.

$$f_s > 2 \cdot f_{max} \qquad \text{Nyquist rate} \qquad (2)$$

3.1.2 Microphone

For the project to work it requires a continuous stream of recorded sound data. This is achieved by a sound detection sensor that records the sound using a microphone then converting it to a digital signal that is relative to the voltage recorded by the sensor. The conversion is performed by an Analog-Digital Converter that measures the strength of the incoming analog signal and generates a corresponding digital signal. This process is affected by multiple variables like Signal-Noise ratio, sampling rate and bandwidth.

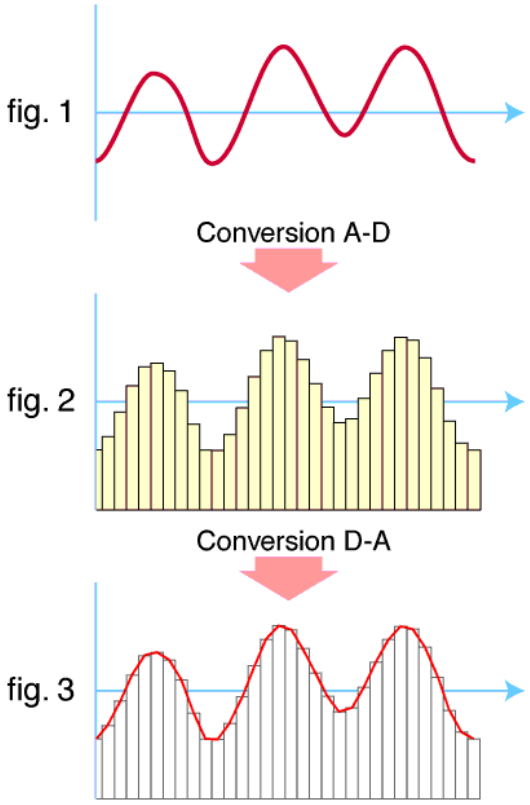


Figure 2: Conversion AD - DA

Using the KY-037's ADC it transmits the digital value converted to a 12-bit resolution, allowing for values between 0-4095 to be read from the program. These values later need to be calibrated to align with the values that is required by the software. The sound data is then sampled using an FFT to obtain the frequency bands and magnitude of the recorded audio.

3.1.3 FFT

In order to transform the recorded data into frequency bands, it is necessary to perform sampling on the data. For this project, Fast Fourier Transform (FFT) is used to compute the Discrete Fourier Transform, which is achieved by calculating the values of a sample set of data. The FFT utilizes Equation 3 to determine the number of points in each frequency band.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-2\pi kn/N} \quad k = 0,\ldots,N-1 \quad \text{Discrete Fourier Transformation} \tag{3}$$

Before computing the FFT, the data is first processed to remove the relative current value received from the ADC. This is achieved by calculating the mean value of all the data points and subtracting it from each individual value. Following this step, the FFT is computed for each value.

To mitigate the effects of using FFT, a technique called windowing is applied, which multiplies each value based on a scale that enables a smoother transition between sample sets. In this project, the Hamming window [6] is used, as shown in Figure 3.

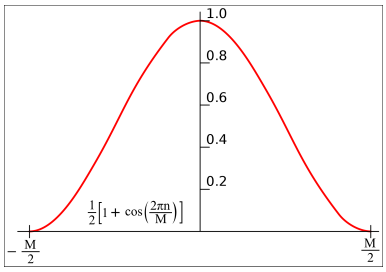


Figure 3: Hamming Window

After applying the Hamming window, it is necessary to calculate the magnitude of each complex frequency band to determine the corresponding frequency magnitude. The primary method used involves calculating the square root of the real and imaginary values. However, since this computation is resource-intensive for the CPU, the same result can be achieved using an approximation method [7].

$$X_{r_n} = \sqrt{X_{r_o}^2 + X_{im}^2} \quad \text{Original method} \tag{4}$$

$$X_{r_n} = 0.95 \cdot \max_{X_{r_o}, X_{im}} + 0.39 \cdot \min_{X_{r_o}, X_{im}} \quad \text{Approximation of magnitude} \tag{5}$$

3.1.4 Spectrogram

A spectrogram provides visual representation of the frequency content of a signal over time. It give a possibility to analyze and visualize the distribution of energy across different frequencies as they change over time. Spectrograms are widely used in audio signal processing, speech recognition, and music analysis, as they allow better understanding of the characteristics of audio signals [8]

In a spectrogram, the x-axis represents time, the y-axis represents frequency, and the color or intensity at each point (x, y) corresponds to the amplitude or power of the signal at that particular frequency and time. This visualization makes it easier to identify patterns which is useful for audio recognition tasks because they provide a compact and informative representation of the audio signal. By transforming the raw audio data into a time-frequency representation, it becomes easier to analyze and extract features that are relevant for various recognition tasks, such as gunshot detection. Furthermore, the use of machine learning techniques can be applied to these features to improve the accuracy and efficiency of recognition algorithms.

A spectrogram is generated using the Short-Time Fourier Transform (STFT), which involves applying a sliding window to the audio signal and computing a Fast Fourier Transform (FFT) for each window. The STFT is represented as:

$$\mathrm{STFT}x(t)(m,\omega) = \sum_{n=-\infty}^{\infty} x(n)w(n-m)e^{-j\omega n} \tag{6}$$

Where x(t) is the input signal, w(n) is the window function, m is the time index, and is the frequency. [9, 10]

3.2 Machine Learning

Machine learning is a subbranch of artificial intelligence that aims to develop algorithms that can learn patterns from data [11]. It enables computers to make decisions or predictions without being explicitly programmed for specific tasks. This section will provide an overview of the main types of machine learning and some common specialized architectures.

3.2.1 Types of Machine Learning

There are three primary types of machine learning: supervised learning, unsupervised learning, and reinforcement learning [11].

- *Supervised learning*: involves training a model using labeled input-output pairs. The model learns to map inputs to the correct outputs based on the provided training data. Common tasks include classification and regression.
- *Unsupervised learning*: involves training a model without labeled data. The model learns patterns or structures within the data, such as clustering, dimensionality reduction, or density estimation.
- *Reinforcement learning* involves training a model to learn a policy by interacting with an environment. The model learns to take actions that maximize cumulative rewards over time, based on feedback from the environment.

Although the aforementioned types of machine learning constitute the primary categories, it should be noted that additional specialized types and variations exist within the field.

3.2.2 Deep Learning

Deep learning is a subfield of machine learning that focuses on neural networks with multiple layers, enabling the learning of increasingly complex representations of data [12]. Common deep learning architectures include Convolutional Neural Networks (CNNs), Generative Adversarial Networks (GANs), and Recurrent Neural Networks (RNNs).

- *CNN*: Designed for processing grid-like data such as images, CNNs use convolutional layers to automatically learn spatial hierarchies of features from the input.
- *GAN*: Composed of two neural networks, a generator and a discriminator, GANs learn to generate new data samples that resemble the training data through an adversarial training process.
- *RNN*: Designed for processing sequences of data, RNNs use recurrent layers to maintain an internal state, allowing them to capture temporal dependencies in the data.

While the deep learning architectures discussed are among the most prevalent, numerous alternative architectures have been designed for specific tasks and applications.

3.2.3 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a class of deep learning models that are specifically designed for image and signal processing tasks [12]. They have been proven highly effective in various applications, including image classification, object detection, and speech recognition. CNNs consist of multiple layers, including convolutional layers, pooling layers, and fully connected layers. Some examples of these layers are:

- *Convolutional Layers*: which are the primary building blocks of CNNs. This layer is designed to detect local patterns within the input data, such as edges or textures in images. The layer applies a set of filters or kernels to the input data, which are small matrices that slide over the input data and perform element-wise multiplication followed by a sum. Each filter is responsible for detecting a specific feature or pattern, and the output is a set of feature maps that capture the presence of these patterns in the input data.

- *Pooling layers*: Another component of CNNs. They are used to reduce the spatial dimensions of the feature maps, which helps to decrease the number of parameters and computational complexity of the network, thus reducing the risk of overfitting. The most common type of pooling operation is max pooling, which selects the maximum value within a local region of the feature map

- *Dropout layers*: employed to help prevent overfitting by adding a regularization technique during training. It randomly sets a fraction of the input neurons to zero at each training step, effectively "dropping out" those neurons from the network. This prevents the model from relying too heavily on any single neuron, promoting the learning of robust features and improving the generalization capabilities of the network. The dropout rate, a hyperparameter that defines the fraction of neurons to be dropped out, is typically set between 0.2 and 0.5

- *Fully connected layers*: Used towards the end of the network to combine the learned features from the convolutional and pooling layers and produce the final output. Each neuron in a fully connected layer is connected

to every neuron in the previous layer, and the output is typically passed through an activation function to produce a probability distribution over the possible classes.

It is important to acknowledge that, apart from the key components of CNNs presented, other types of layers and variations may be employed depending on the particular problem being addressed.

3.2.4 Activation Functions

Activation functions are essential components of neural networks, introducing non-linear properties to the model [12]. They determine the output of a neuron based on its input, enabling the network to learn complex relationships between input features and the target variable. Some common activation functions include:

- *Sigmoid*: A smooth, S-shaped function that maps any input value to a range between 0 and 1. It is commonly used in the output layer for binary classification tasks.
- *Rectified Linear Unit (ReLU)*: A piecewise linear function that outputs the input value if it is positive and 0 otherwise. ReLU has become the default activation function in most neural networks because of its computational efficiency and ability to mitigate the vanishing gradient problem.
- *Softmax*: Used in the output layer for multi-class classification tasks, the softmax function normalizes the output of each neuron to form a probability distribution over the possible classes.

The activation functions detailed are extensively utilized in neural networks; however, a wide range of alternative activation functions exists, each with its own unique properties and use cases.

3.2.5 Optimization Functions

Optimization functions, also known as optimizers, are algorithms used to adjust the weights and biases of neural networks during the training process [12]. They aim to minimize the loss function, which measures the difference between the network's predictions and the true target values. Some common optimization functions include:

- *Gradient Descent*: A basic optimization algorithm that updates the model parameters by moving in the direction of the negative gradient of the loss function with respect to the parameters.
- *Momentum*: An extension of SGD that incorporates a momentum term, which accelerates the optimization process in the direction of the negative gradient and dampens oscillations.
- *Adaptive Gradient (AdaGrad)*: An optimizer that adapts the learning rate for each parameter individually based on the accumulated gradients. This can lead to faster convergence, especially for sparse data.
- *Adaptive Moment Estimation (Adam)*: A popular optimization algorithm that combines the benefits of both momentum and AdaGrad. It computes adaptive learning rates for each parameter and maintains an exponentially decaying average of past gradients.

The highlighted functions are among the most popular optimizers, but it should be noted that multiple of other optimization algorithms with distinct properties and advantages does exsists.

4 HARDWARE

In this chapter, the hardware components used in the project will be discussed, focusing on their specifications and the rationale behind their selection. The chosen hardware components are crucial in ensuring the effective implementation of the machine learning algorithms and real-time processing required for this application. The primary hardware components discussed in this chapter include the embedded board, sensors, and communication modules.

4.1 Embedded Board

The embedded board selected for this project is the Heltec Wireless Stick Lite V2 [13], which is based on the ESP32 [14]. The choice of this board was primarily driven by its dual-core architecture, which is essential for meeting the real-time and concurrency requirements of this project. The Heltec Wireless Stick Lite V2 is equipped with 4MB of flash memory and 320KB of DRAM, which will be utilized for data storage, sound processing, and predictions. These two capabilities form the foundation of the project. Additionally, the board features a LoRa radio module, allowing for the transmission of gunfire confirmations over the internet.

Table 1: Heltec Wireless Stick Lite V2 Specifications

Parameter	**Value**
Microcontroller	ESP32
Processor	Dual-core Tensilica LX6
Clock Speed	Up to 240 MHz
Flash Memory	4 MB
SRAM	520 KB
Wireless Connectivity	Wi-Fi, Bluetooth
LoRa Radio Module	SX1276 chip, 433/868/915 MHz frequency bands
I/O Pins	10 digital, 1 analog, 2 DAC
Interfaces	USB, JST connector
Power Supply	3.7V LiPo battery, USB
Power Consumption	70 mA (average), 0.5 mA (sleep mode)
Power Supply	5V DC (via USB Type-C) or 3.3V DC (regulated)
Development Tools	Arduino IDE, ESP-IDF framework

4.1.1 Operating system

The project uses the FreeRTOS kernel [15] to enable real-time scheduling, which is important for maintaining concurrency control within the device. This ensure continuous sound recording without interruptions due to data sampling and classification being important for the system's effectiveness. Thus, the use of FreeRTOS allows for the implementation of various tasks running simultaneously on separate cores.

4.2 Sensor

4.2.1 Microphone

The KY-037 sensor[16] is a small module designed to detect sound intensity and convert it into an electrical signal. It is commonly employed in home made electronics projects, such as development of sound-activated lighting systems or alarms. This sensor was used due to the accessibility. Although it may not provide the highest quality data, it serves as a proof of concept, demonstrating that with a larger budget and more time, the project's performance could be improved.

Table 2: Specifications of the KY-037 Microphone

Parameter	**Value**
Operating voltage	3.3V to 5V DC
Signal output	Analog, 0V to 5V, up to 15mA
Detection range	50dB to 100dB
Sensitivity adjustment	Potentiometer
Module size	30mm x 15mm x 10mm
Frequency response	20Hz - 20KHz
Operating temperature range	-10°C to 70°C
Humidity range	Affected by high humidity

4.2.2 LoRa

The device incorporates a built-in LoRa radio module to enable data transmission in the event of a gunfire detection. The module operates in the 868MHz frequency mode. Since the Heltec board already includes a built-in LoRa chip, specific designated pins are utilized for sending data over LoRa. Through these pins, the software can create a packet message and transmit it using the device ID, application ID, and application key. This system employs the LoRaWAN protocol [17], which offers long-range communication, low power consumption, and license-free operation while having a limited payload size and transfer rate. Therfore, the LoRaWAN protocol is used to send data upon the true detection of gunfire.

4.3 System hardware design

4.3.1 Power consumption

As the project heavily depends on a continuous stream of input data, there are limited possibilities of optimizing the power usage of the device. Transmissions using LoRa is only done when the prediction returns as gunfire. Therefore minimizing the amount of power used to transmit data. However, it is important to note that the device initiates a connection to LoRa upon startup and remains connected throughout its operation, which may impact power usage.

Below is the baseline power consumption that is provided from the datasheet

Table 3: Board power consumption

Power Consumption	**Value (mA)**
Sleep mode	0.1
Idle mode	4
Transmitting at +20 dBm	120
Receiving at 150 Mbps	70

4.3.2 Circuit

The following figure shows the circuit schematic and how the sensor was connected to the board:

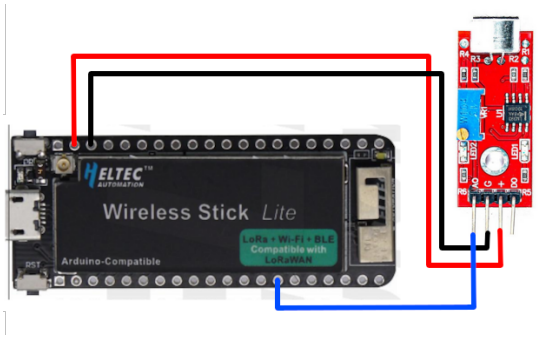


Figure 4: Circuit Diagram

4.3.3 Casing

The device with the microphone was cased with a 3D printed case found on the internet at the page Printables [18].

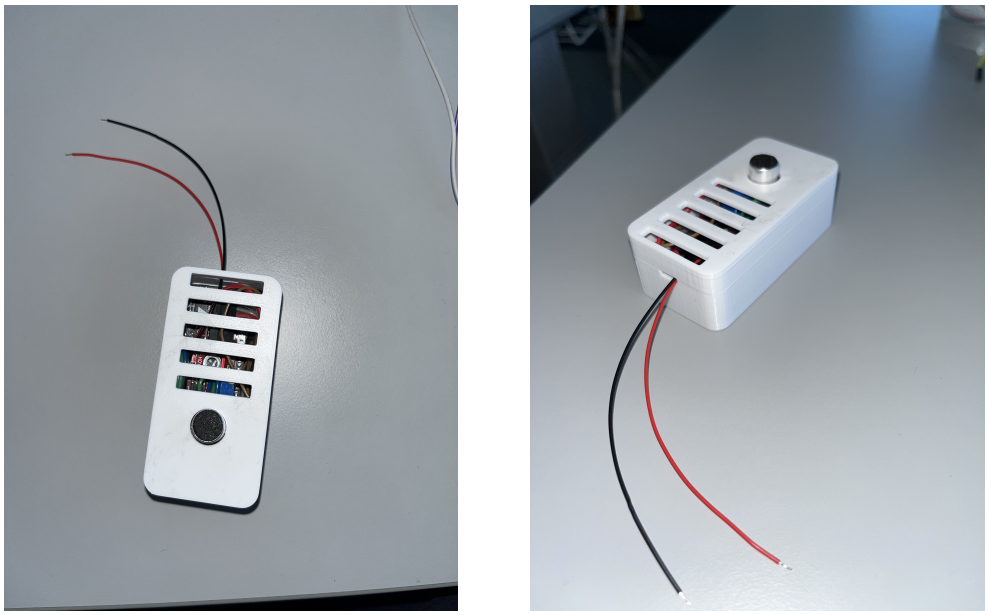


Figure 5: Product Prototype

5 SOFTWARE

This chapter will go over the different software design choices that had to be made during the project. It will go over choice of programming language and why that was chosen, how real-time constraints could affect the software, if any security issues might be apparent and the third party libraries that was used in the project.

5.1 Programming Language

Given that the Heltec Wireless Stick Lite device was designed to be operated with the Arduino Integrated Development Environment (IDE), the decision was made to utilize C++ due to its capacity to regulate memory usage and support of necessary libraries for performing the calculations and predictions required for the project.

Alternatives to C++, such as MicroPython, C, and Assembly, were evaluated but found to be unsuitable for the project constraints. Specifically, MicroPython was dismissed due to its lack of robust support for a Real-Time Operating System (RTOS), which was necessary for the project's programming requirements. While C could have been a viable option, the absence of relevant libraries such as TensorFlowLite micro and Fast Fourier Transform (FFT) in C would have resulted in significant time investment to develop a functional system. Similarly, the decision to not use Assembly language was based on the complexity and volume of code required to be developed from scratch within the project's limited timeframe.

5.2 Security

The project does not raise any significant security concerns, although it is theoretically possible to manipulate the system to produce continuous false alarms by running simulated audio clips as gunfire in close range, or by accessing the device physically.

5.3 Libraries

The application software utilize multiple libraries for various tasks, including creating and training the machine learning model, as well as handling audio sampling and establishing LoRa connections on the device. In this chapter, these libraries will be introduced and explained in terms of their usage and integration within the project.

5.3.1 TensorFlow

TensorFlow is an open-source machine learning framework developed by Google [19]. It has become a popular choice for implementing deep learning models since it provides a comprehensive ecosystem of tools, libraries, and resources that allows for the easy development and deployment of machine learning models. For this project, TensorFlow was used to create, train, and deploy the neural network model responsible for classifying the sounds captured by the microphone.

TensorFlow's flexibility and high-level APIs enable the creation of custom architectures and the use of pre-trained models. In this project, TensorFlow was used to build the Convolutional Neural Network (CNN) model for sound classification, which was trained on a dataset of audio samples. The trained model was then deployed on the embedded board, allowing for real-time analysis and classification of incoming audio data. TensorFlow's support for various hardware platforms and its efficient computation capabilities made it a suitable choice for this project, as it allowed for the integration of the machine learning model into the Heltec device.

5.3.2 TTN_ESP32

The project uses TTN ESP32[20] library to facilitate communication between the device and TheThingsNetwork via LoRa. This library requires the application, device, and eui-ids to be connected to LoRa, after which it attempts to establish network connectivity. Upon detection of a gunshot, the library is used to transmit a "HIT!" message.

5.3.3 *ArduinoFFT*

The ArduinoFFT[21] library is used for processing the recorded microphone samples. This library was specifically designed to perform a Fourier transformation on the data, allowing for the extraction of frequencies and magnitudes. The library supports a set of functions to interface with the sample data. The library is initialized with the real and imaginary data, sample size, and frequency read.

The sample data undergoes a DC removal process to eliminate any existing noise. Then, the windowing settings are fine-tuned to enable more precise frequency band averaging for the FFT values. Lastly, the FFT values are computed, and the magnitudes of the frequency bands are determined.

5.4 Dataset

The pursuit of a high-quality and diverse dataset for the project proved to be challenging, as there were limited available datasets containing gunfire sounds. After thorough research, three main datasets were selected:

- The primary source of positive sound samples was the Gunshot Audio Dataset from Kaggle [22], which contains a collection of gunfire sounds.
- An additional audio dataset containing gunfire sounds was obtained from Zenodo [23]. However, this dataset required considerable preprocessing, as each sound was repeated multiple times with varying sample rates or decibel levels.
- The Environmental Sound Classification 50 dataset from Kaggle [24] was used as the primary source of negative sound samples. This dataset was manually inspected and cleaned, with any gunfire sounds identified and transferred to the positive sound samples.

Given the nature of the collected data, considerable effort was dedicated into structuring the datasets. All gunfire sounds from the various sources were combined into a folder labeled "positive" indicating the presence of gunfire. Conversely, non-gunfire sounds were placed in a folder named "negative". This structuring goal was to simplify data processing and model training, as it facilitate easier access to relevant audio samples during the model creation step.

5.5 Implementation

As previously discussed in the Programming Language section of this chapter, the primary languages employed in this project were C++ and Python. The subsequent sections will elaborate on the specific use of these languages in the project's various stages.

5.5.1 Model Training

Python offers a wealth of libraries specifically designed for deploying machine learning models, which greatly simplifies the implementation of such tasks. The language proved to be an invaluable tool in reading and preprocessing the data, as well as building and training the model. The Python version utilized for this project was the latest at the time, version 3.10.9. The development environment used was PyCharm 2022.3.3 (Community Edition).

Both TensorFlow and TensorFlow I/O libraries were employed, with TensorFlow I/O serving as a complementary library to TensorFlow. It provided additional functionalities that were beneficial for certain preprocessing steps, such as audio resampling at different rates than the original.

The development of the model involved three primary stages:

DATA PREPROCESSING The initial step involved reading and listing the data from both the positive and negative directories, where labels were assigned to each file. Files in the positive directory were given a label of 1, while those in the negative directory were assigned a label of 0.

Subsequently, all audio .wav files were resampled to 16000Hz and compressed to a single channel due to the board's limitations in processing 44100Hz sound data. After resampling each sound, a spectrogram was generated. Since spectrograms are inherently two-dimensional, it was necessary to expand their dimensions to three and resize them to fixed dimensions of (128, 64) before reducing the dimensions back to two. This step ensured consistent input shape for the model, which was chosen based on the achieved shape that was generated on the board out of *2* seconds audio data. Furthermore, each spectrogram was rounded to two decimal places, as the board could not support more accurate numerical depths.

An additional aspect requiring careful consideration involved expanding or reducing the length of each sound to ensure consistency when creating the spectrogram. However, this step was ultimately omitted due to potential complications. The intended length of the sound files was 48000 (equivalent to 16000 * 3, or 3 seconds), which was the mean length of all audio files. However, the original audio data varied in length, particularly for the more critical gunfire sounds. Some recordings had the shot at the beginning, some at the end, and others in the middle. Cropping the audio could reduce the trustworthiness of whether a cropped audio file indeed contains a gunfire sound or not. Padding shorter audio files would not have a significant impact in this case, as the reshaping process already addresses this concern.

The final step in the data processing stage involved randomly shuffling the data to prevent large consecutive stacks of positive or negative samples, which could hinder the model's ability to learn effectively. Following this, the entire dataset was divided into 80% for training and 20% for testing. Furthermore, the 80% allocated for training was further divided, with 80% being used for actual training and the remaining 20% serving as a validation set.

MODEL ARCHITECTURE AND TRAINING The chosen architecture for the sound classification model was a Convolutional Neural Network (CNN), implemented using TensorFlow's Sequential model. The architecture consists of:

- An Input layer with a shape of (128, 64), corresponding to the reshaped spectrograms.
- Three Conv1D layers, with 4, 8, and 16 filters respectively, and a kernel size of 3. Each of these layers uses the ReLU activation function.
- A MaxPooling1D layer with a pool size of two, which reduces the spatial dimensions of the feature maps.
- A Dropout layer with a rate of 0.25, added to prevent overfitting by randomly dropping a fraction of the input units during training.
- A Flatten layer to convert the pooled feature maps into a one dimensional array.
- A Dense layer with 16 units, ReLU activation function, and an L1 regularization term with a weight of 0.001. The regularization term helps to prevent overfitting by adding a penalty to the model's loss function based on the absolute values of its weights.

- A final Dense layer with 1 unit and a sigmoid activation function, which outputs the probability of the input sound being a gunshot.

The total number of parameters in the model was approximately 17,600. While a larger model might yield better results, the limited memory capacity of the board necessitated a reduction in the model's size as much as possible.

The model was compiled using the Adam optimizer, Binary Crossentropy loss, and metrics such as Binary Accuracy, Recall, and Precision. It was then trained for 100 epochs, with validation data provided to monitor the model's performance during training. After the training process, the model was saved for further use.

MODEL EVALUATING The final step was to evaluate the model's performance using the 20% portion of the dataset reserved for testing. This allowed for an assessment of the model's ability to generalize and perform well on unseen data.

5.5.2 Model Converting

The model conversion process was divided into two main steps. The first step involved converting the saved TensorFlow model to TensorFlow Lite format, and the second step involved generating a header file for integration with the C++ project and subsequent deployment onto the target hardware.

TENSORFLOW TO TENSORFLOW LITE CONVERSION To convert the TensorFlow model to TensorFlow Lite format, a simple Python script was used. The script loaded the previously saved model and used the TensorFlow Lite Converter API to convert it. The converter was set to use default optimizations, which are designed to reduce the size of the model and improve its performance on resource-limited devices. Once the model was converted, it was saved in the tflite format for further use.

GENERATING HEADER FILE The TensorFlow Lite model was then converted to a header file, which enabled its integration into the C++ project. This conversion was performed using the 'xxd´ command-line utility, which transformed the tflite model file into a C array. The generated header file was subsequently included in the C++ project, allowing for seamless integration with the rest of the code and enabling the model to be built and flashed onto the device.

5.5.3 Audio Recording & Sampling

As mentioned in the background of this project, it works by reading audio data that is being gathered by the KY-037 sensor, this part will be described in the follow paragraphs.

RECORDING OF AUDIO DATA The program starts by initializing parameters pertaining to the quantity of samples and the duration for audio reading. The data is then read in two-second increments before being sent for sampling. In order to obtain data that matches the dataset, it is necessary to preprocess the values collected from the Analog-to-Digital Converter (ADC). This is achieved by mapping the values to match the PCM values when the wav files are read, followed by normalizing the values to fall between 1.65 and -1.65.

SAMPLING OF AUDIO To compute the mapped values for the sampling task, the first step is to remove the average values to eliminate the baseline noise. Next, a windowing function is applied, and the data is looped over to select 256 values at a time, with 64 overlapping values from the previous and 64 of the next sample set. This process results in the creation of a spectrogram with the shape of (128x64).

The code implemented for audio sampling drew heavily from existing examples found online, but was customized to meet the unique requirements of the project. These constraints required substitution of the existing square root formula for an approximation formula that was described in the theoretical chapter.

PRDICTION At the start of device operation, the model is initialized to ensure its functionality and to obtain the necessary input and output buffers.

The sampled spectrogram's values from each loop are directly assigned into the input buffer, as a complete spectrogram cannot be created at once due to RAM limitations. Once the data sampling loop is complete, a new task is created to invoke the model and obtain the prediction result. If the prediction value is higher than 0.5, indicating a 50% or higher chance of a gunfire sound, a task is triggered to transmit a message to the LoRa network.

5.5.4 Device-Software Deployment

The implementation of the software was executed through utilization of the Visual Studio Code extension, PlatformIO. This particular extension is designed to facilitate the creation of software for embedded systems. This enables the seamless building and shipping of software code onto the device, in addition to offering access to libraries and a straightforward approach to integrating dependencies. Consequently, the codebase incorporates the requisite configuration file necessary for compiling and exporting the software code to the target board.

To establish a LoRa connection using TTN_ESP32, it was necessary to remove two file classes from the library - TTN_BLE_esp32 and TTN_Cayenne. The reason for this removal was due to a coding error within the library, which attempted to access a Bluetooth module that was not present on the board being used.

6 RESULTS

This chapter will go over the results from the project, it will discuss the outcome and provide images of the product and data feedback.

6.1 Machine Learning

The evaluation of the model was carried out using the 20% testing data that had been previously set aside. The model achieved 99.5% accuracy, successfully classifying all positive samples and all but two negative samples. Additional manual testing was performed using audio clips sourced from YouTube containing both positive and negative sounds. In this test, the model was able to correctly classify 18 out of 20 negative sounds and 10 out of 12 positive sounds.

On the target hardware, due to its limited memory capacity, it was not feasible to test using audio files. Instead, more clips from YouTube were used to evaluate the performance of the model. The model was mostly successful in correctly classifying positive and negative sounds. However, it occasionally exhibited unexpected behavior, providing predictions above 1, even though the expected output range should be between 0 and 1. The cause of this anomaly could not be definitively determined. It is unclear whether the issue was caused by the microphone providing unusual input data to the model or if the model itself was at fault, given that the TensorFlow Lite Micro library is relatively new, still missing some features, and may still contain bugs.

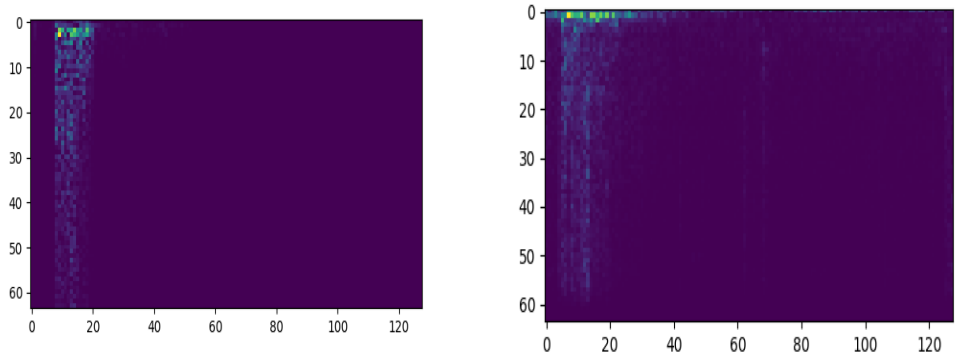
Please refer to Appendix A for a visualization of the model's performance during the training phase.

6.2 Audio Sampling

From the research questions asked in the report, results indicate that the system is capable of reading and sampling audio data up to 2 seconds without disrupting the data readings for the new sample set. This was achieved by optimizing the system to sample the data fast enough to prevent memory address conflicts arising from concurrent use by multiple functions.

Due to RAM limitations, the sampling process was restricted to a maximum of 2 seconds of data at a time. To enhance the efficiency of the system, we chose to revise the existing arduinoFFT library, substituting the original square root calculation with an approximation calculation requiring less computation power. This modification resulted in a 13% increase in efficiency of the sampling process.

Furthermore, the sampling process applies an STFT by using overlapping samples of data, producing a smoother spectrogram. The sampled data was then transformed into a one-dimensional array, which represents a spectrogram for input into the model.



(a) Recorded audio **(b)** Dataset

Figure 6: Spectrograms

7 CONCLUSION

This chapter will go over the results and conclusions that was drawn from the project, it will involve challenges, research answers, future work and responsibility areas of the authors.

7.1 Challenges

As the project is based on a micro controller, there is an inherited challenge in creating a program that runs on very limited resources. This was one of the biggest hurdles in this project due to the size of recordings that was needed to have a good prediction on the model. The first iterations was based on a three second sound sample that was reran every second. This did not work due to RAM and concurrency issues, it took to long for the sampling of 3 seconds to be ran every second.

Due to this it was decided to go down to three second recording samples, which worked very well in terms of sampling and having concurrency but proved to be a problem when the model had to be implemented on the micro controller. As the model itself takes up around 70kbit of RAM to be initialized. Forcing the reduction to 2 seconds of sampled audio to be predicted by the model. This is the choice that was made for the final product, however it also provide some challenges as it can overlap the gunshot such that it doesn't fully classify it as a gunshot.

Another challenge that became apparent was the usage of a cheap microphone sensor, it was chosen way to late to test listening to the audio that was sampled from this and it proved to have a lot of noise in the audio it recorded. This have to be considered in the prediction of the data as all the sampled data will have error margins in the data due to noise.

It required a lot of time to get an understanding of the language used, due to it being a lower level language compared to what has been used prior. Requiring research into memory handling and building of a project.

There was another inherited challenge in that the choice of project was originally way out of scope for the amount of time that was allocated for the project,

this showing in the project time spent from both parties. Going way above the 20 hours/week that was allocated to the course.

7.2 Research answers

Are microcontrollers capable of continuously read audio data, preprocess it in 3-second intervals using overlapped windows of 2 seconds from the previous sample?
Due to the statements from the challenges section, with the current RAM limitations and our implementation it would not work. It requires to much data being processed at the same time as recording. With increasingly better hardware it is feasible that this would be a potentiality in the future.

Using 3 seconds audio samples could it be classified without disturbing other processes?
From the experiments done during the project it was concluded that it is possible to classify the audio samples in under three seconds, where the final result ended up with a 460 ms calculation per sampled data.

7.3 Future work

There is some work that could be done in the future regarding this project, some that could be implemented today with the current technology and knowledge level of the authors. This is the localization part of the underlying project statement, only reason it was not finished was due to the time constraints of the course.

With upgraded hardware it could open up potential of having larger models and continuous predicting as this project had planned from the beginning. This also would allow three second clips to be sampled and predicted every new second of recorded audio. Which could remove some of the error margin regarding the prediction.

7.4 Responsibility areas

During the project it was decided that Saleh would start working on the machine learning model and how to implement TensorFlow Micro on the device while Robin started implementing the RTOS tasks and handling sound recording, sampling and sending to TTN using LoRa. As the project went on Saleh started problem solving on the recording together with Robin, who after focused more on setting up the report and writing.

During the project time frame it was recorded that Saleh worked for 450 hours (56 hours/week) and Robin worked for 300 hours (37.5 hours/week).

REFERENCES

[1] Polisen. *Sprängningar och skjutningar - polisens arbete*. Last accessed 2023-03-12
https://polisen.se/om-polisen/polisens-arbete/sprangningar-och-skjutningar/.

[2] E. Grane L. Bokelund. *Gunshot Detection from Audio Streams in Portable Devices*, 2022. Last accessed 2023-03-12
https://lup.lub.lu.se/student-papers/search/publication/9090317.

[3] 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, 2019. Last accessed 2023-03-05
https://ieeexplore.ieee.org/document/9006456.

[4] M. Srivastava S.S. Saha, S.S. Sandha. *Machine Learning for Microcontroller-Class Hardware: A Review*. Last accessed 2023-03-12
https://arxiv.org/ftp/arxiv/papers/2205/2205.14550.pdf.

[5] Noise Monitoring Services. *How loud is a gunshot?* Last accessed: 2023-03-20
https://www.noisemonitoringservices.com/how-loud-is-a-gunshot/.

[6] FREDERIC J. HARRIS. *Handbook of Digital Signal Processing*, 1987. 2023-03-11
https://www.sciencedirect.com/science/article/pii/B9780080507804500084.

[7] Grant Griffin. *DSP Trick: Magnitude Estimator*. Last accesed 2023-03-11
https://dspguru.com/dsp/tricks/magnitude-estimator/.

[8] Center for Computer Research in Music and Acoustics (CCRMA) at Stanford University. *Spectrograms*. last accessed 2023-03-15
https://ccrma.stanford.edu/~jos/mdft/Spectrograms.html.

[9] Center for Computer Research in Music and Acoustics (CCRMA) at Stanford University. *Mathematical Definition of the STFT*. last accessed 2023-03-15
https://ccrma.stanford.edu/~jos/sasp/Mathematical_Definition_STFT.html.

[10] Center for Computer Research in Music and Acoustics (CCRMA) at Stanford University. *The Short-Time Fourier Transform*. last accessed 2023-03-15 https://ccrma.stanford.edu/~jos/sasp/Short_Time_Fourier_Transform.html.

[11] IBM. *What is machine learning?* last accessed 2023-03-15 https://www.ibm.com/topics/machine-learning.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. ***Deep Learning***. MIT Press, 2016. last accessed 2023-03-15 http://www.deeplearningbook.org.

[13] Heltec Automation. *Wireless Stick Lite LoRa Node Development Kit*, 2020. Last accessed 2023-03-11 https://resource.heltec.cn/download/Wireless_Stick_Lite/lite.pdf.

[14] Expressif Systems. *ESP32 Series Datasheet*, 2023. Last accessed 2023-03-11 https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.

[15] *FreeRTOS*. Last accessed 2023-03-11 https://www.freertos.org/.

[16] joy it. *KY-037 Microphone sensor module (high sensitivity)*, 2017. Last accessed 2023-03-11 https://datasheetspdf.com/pdf-file/1402047/Joy-IT/KY-037/1.

[17] LoRa Alliance. *A technical overview of LoRa® and LoRaWAN™*, 2015. Last accessed 2023-03-11 https://hz137b.p3cdn1.secureserver.net/wp-content/uploads/2020/11/what-is-lorawan.pdf?time=1678214570.

[18] *ESP32 Case for WLED*. Last accessed: 2023-03-13 https://www.printables.com/model/255431-esp32-case-for-wled.

[19] Google. *TensorFlow*. Version 2.11.0 https://www.tensorflow.org/.

[20] Francois Riotte. *TTN-ESP32*. Version 0.1.3 https://github.com/rgot-org/TheThingsNetwork_esp32.

[21] Enrique Condes. *ArduinoFFT*. Version 1.5.6
https://github.com/kosme/arduinoFFT.

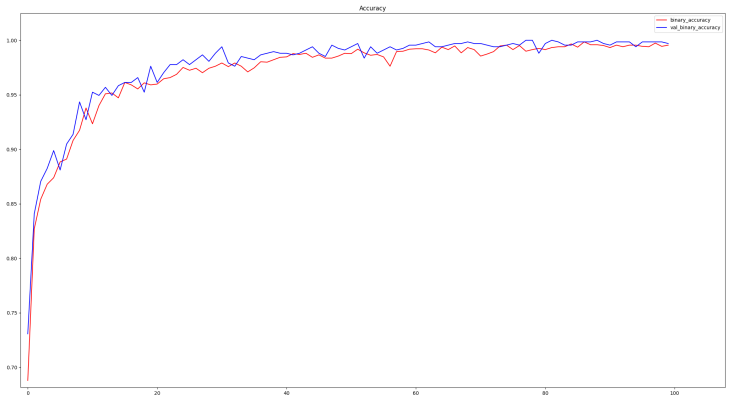
[22] Emrah Aydemir. *Gunshot Audio Dataset*. Last accessed 2023-01-20
https://www.kaggle.com/datasets/emrahaydemr/gunshot-audio-dataset.

[23] Ruksana Kabealo and Steven J. Wyatt. *Gunshot/Gunfire Audio Dataset*, July 2022. last accessed 2023-01-21
https://doi.org/10.5281/zenodo.7004819.

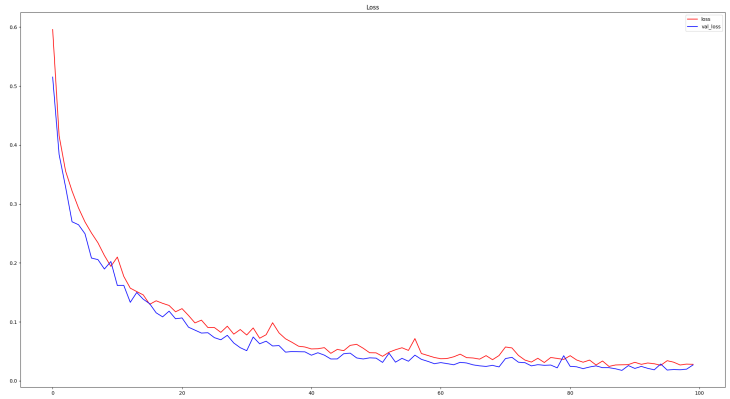
[24] Maxime Moreaux. *Environmental Sound Classification 50*. last accessed 2023-01-20
https://www.kaggle.com/datasets/mmoreaux/environmental-sound-classification-50.

A APPENDIX 1.1

Model Performance During Training.

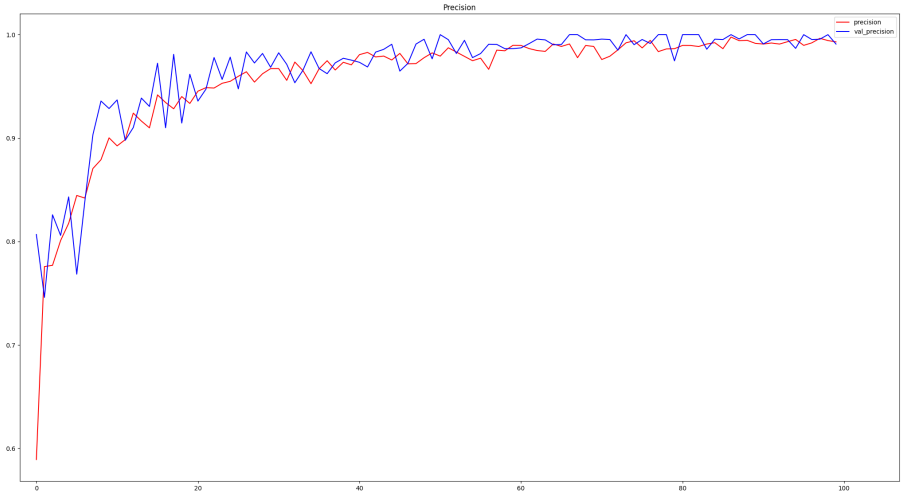


(a) Accuracy

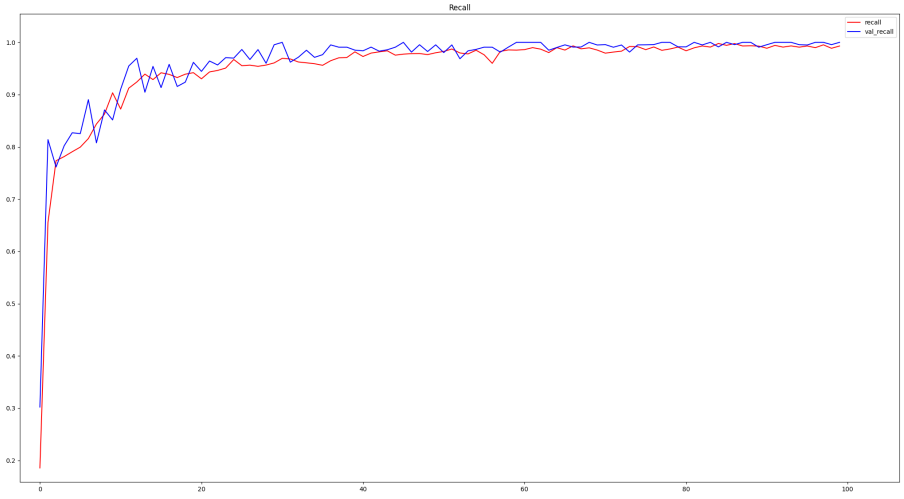


(b) Loss

Figure 7: Accuracy and Loss



(a) Precision



(b) Recall

Figure 8: True Positive and False Negative

B APPENDIX 1.2

Source code provided using gitlab
https://gitlab.lnu.se/ss225bx/2dt304-project-with-embedded-system

C APPENDIX 1.3

Video presentation of project
https://youtu.be/8LgsOIyKHro