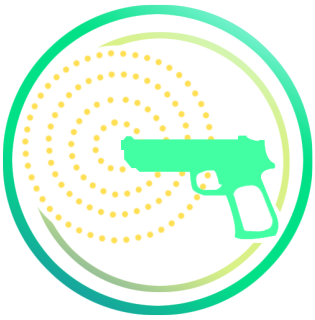
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



PAGD

Privacy-aware Gunshot Detection

Smartphone-based Solution to Distributed Gunshot Detection and Localization using Deep Learning

Bachelor's thesis in Computer Science and Engineering

Martin Engström
Lukas Gartman
Salam Hani
Sven Kellgren
Anas Masri
Tobias Olsson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Bachelor's thesis 2023

Privacy-aware Gunshot Detection

Smartphone-based Solution to Distributed Gunshot Detection and Localization using Deep Learning

Martin Engström
Lukas Gartman
Salam Hani
Sven Kellgren
Anas Masri
Tobias Olsson

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Privacy-aware Gunshot Detection
Smartphone-based Solution to Distributed Gunshot Detection and Localization using Deep Learning
Martin Engström Lukas Gartman Salam Hani Sven Kellgren Anas Masri
Tobias Olsson

Supervisor: Ahmed Ali-Eldin Hassan and Magnus Almgren, Department of Computer Science and Engineering
Examiner: Arne Linde, Department of Computer Science and Engineering

Bachelor's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Icon representing the Privacy-aware Gunshot Detection group

Typeset in LaTeX
Gothenburg, Sweden 2023

Abstract

In this technical report, we describe the design of a gunshot detection system developed for smartphones, covering the basic theory behind the system and look at potential further improvements. We discuss both the broader issue of gunshot detection and explore a specific solution to distributed gunshot detection with the use of smartphones. The system leverages deep learning and GPS technology to provide real-time information about shooting incidents. With the potential to be operational at a large scale and receive input from multiple mobile devices, the system could promptly alert relevant authorities to potential incidents and warn users. The implementation of such a system could contribute to saving lives, reducing harm, convicting criminals, and enhancing public safety. The challenges lie in the complexity in achieving accurate detection in any acoustic condition with a lightweight solution that can operate without compromising the user experience. We further consider the user experience, which imposes stricter processing and energy requirements. Our results show that it is feasible to detect gunshots on smartphones and that it can be a good solution to distributed gunshot detection. However, it also shows that the accuracy of the system is largely dependent on environmental acoustic factors and the data used for evaluation. With this report, we aim to prompt further research and development in this area, as well as encouraging a more critical analysis of the problem at hand.

Keywords: AI, crime prevention, gunshot detection, privacy

Sammandrag

I denna tekniska rapport beskriver vi designen av ett skottavlyssningssystem utvecklat för smartphones. Vi täcker den grundläggande teorin bakom systemet och undersöker möjliga förbättringar. Vi diskuterar både den bredare frågan om skottavlyssning och utforskar en specifik lösning för distribuerad skottavlyssning med hjälp av smartphones. Systemet använder sig av djupinlärning och GPS-teknik för att ge realtidsinformation om skottincidenter. Med potentialen att vara i drift i stor skala och ta emot bidrag från flera mobila enheter kan systemet snabbt varna relevanta myndigheter om potentiella incidenter och varna användare. Implementeringen av ett sådant system kan bidra till att rädda liv, minska skador, fälla brottslingar och förbättra allmän säkerhet. Utmaningarna ligger i komplexiteten att uppnå noggrann detektion under olika akustiska förhållanden med en lättviktig lösning som inte kompromissar användarupplevelsen. Vi beaktar också användarupplevelsen som ställer högre krav på bearbetning och energiförbrukning. Våra resultat visar att det är genomförbart att detektera pistolskott på smartphones och att det kan vara en bra lösning för distribuerad skottavlossningsdetektering. Samtidigt visar det att systemets noggrannhet i stor utsträckning beror på miljöns akustiska faktorer och den använda datan vid utvärderingen. Med denna rapport avser vi att främja vidare forskning och utveckling inom detta område samt uppmuntra en mer kritisk analys av det aktuella problemet.

Contents

Glossary

- **ML**: *Machine Learning*, a type of computer algorithm that "learns" as you give it training data.
- **DL**: *Deep Learning*, a subset of machine learning algorithms that use artificial neural networks, inspired by how the human brain functions. It has been widely used for image and sound recognition among other things.
- **Features**: relevant characteristics of the data derived from transformations of the raw data to serve as input to the deep learning model. An example of a feature in sound or speech recognition is frequency spectrograms.
- **Preprocessing**: the transformation of data into the features of the deep learning model.
- **Concurrent Programming**: a programming paradigm where you separate parts of the program which do not depend on each other, to run in parallel. This greatly increases the performance of software but comes with many challenges and pitfalls.
- **TDOA**: *Time difference of arrival*, a technique to determine the location of a sound source by the differences in time when the sound is first received by two or more microphones.
- **Scalability**: the ability of an application to handle an increasing number of users, requests or other demand.
- **GPS**: *Global Positioning System*, a satellite-based system used by smartphones to determine current position (latitude, longitude, altitude) within some error-margin.
- **CNN**: *Convolutional neural network*, a type of deep learning neural network that is commonly used in image and audio processing tasks. CNNs are designed to automatically learn and extract relevant features from input data, such as images or audio signals, by using convolutional layers.

List of Figures

List of Tables

1
Introduction

Gun violence continues to be a pernicious problem in a contemporary society, resulting in harm and loss of life [2]. Timeliness and accuracy in response to shooting incidents are crucial to minimizing harm and preserving life. CCTV camera systems are commonly used for crime prevention as a measure of deterrence but they are limited in their ability to provide a measure to respond to ongoing threats since they require constant supervision. Meanwhile, the constant surveillance and storage of unused data gives rise to many privacy concerns. This is what drives new research efforts into real-time gunshot detection systems that automatically and quickly detects and reports gunshot incidents with the use of manually installed equipment in specific locations [3] [4]. However, manual installation of specialized equipment can be cost-prohibitive and logistically challenging to implement at a large scale, and is therefore limited in its feasibility.

The advent of mobile technology, efficient enough to handle complex computations and provide location information in real-time, offers a new paradigm for enhancing public safety. The prevalence of mobile devices equipped with microphones and GPS capabilities provides a means for collecting real-time information about shooting incidents on a larger scale than before. In particular, the application of deep learning algorithms on mobile devices to classify and detect gunshot sounds can improve the response time and action of law enforcement by providing critical information about the severity of the threat in the incident of a shooting.

Furthermore, reports of this kind could be used as critical evidence in court cases. Previous attempts to use such reports as evidence have been met with suspicion due to the uncertainty in the accuracy of the system [5, p.10]. Yet, this suggests that while reports generated by gunshot detection systems may not be sufficient evidence on their own, they can provide additional support to a court case if the system is deemed accurate and reliable. Reliability is also of paramount importance in the moderation of the system to ensure that the resources of the law enforcement are utilized to maximum efficiency.

1.1 Previous Work

D. Welsh and N. Roy investigates gunshot detection systems performed on smartphones in [6] where they conclude that the average accuracy of gunshot detection with the use of smartphones can be at least as high as 86.6% by using different sensors of a smartphone together as inputs to a machine learning model. These results show that it is a feasible task to implement mobile phones as gunshot detection

devices. This prompts further research into how smartphones could be used in a distributed system to detect gunshots, which is the intention with the development of the Privacy-aware Gunshot Detection (PAGD) system that will be described in detail further on in this report.

1.2 Requirements

To ensure that the project meets the desired result, a set of requirements were identified for each component of the system prior to the start of development. These requirements outline the necessary functionalities on a higher level that each component must have. The following sections present these requirements, which serves as the foundation for the development of the system.

1.2.1 Server Requirements

In order to localise a gunshot, it was determined that the first requirement to achieve this is to have a central server to collect and process the information sent from the users. Three main requirements for the flow of communication were set: *security*, *accuracy*, and *speed*. The information being sent (which from now on will be referred to as reports) will contain personal user information and it is therefore critical that the reports must be sent securely to the server to ensure confidentiality and integrity of the data. Furthermore, it is very important that gunshots are both accurately localised with few errors and doing so as quickly as possible by processing the reports in real-time and notifying users immediately. This could be a contributing factor to cutting down police response times.

1.2.2 Application Requirements

The two most important requirements of the system is to achieve good accuracy in gunshot detection while having a small footprint on the resource utilization on the smartphone. The accuracy needs to be sufficiently high so that the system can be relied on by law enforcement, but also not be so tasking on the hardware that the user experience is impacted negatively and that the battery life is noticeably degraded. These requirements stand in direct opposition of one another, which is why striking a balance between them is necessary for the system to be useful. Alongside these technical requirements, the system must also foster an environment that champions user-friendliness, privacy, and integrity. It is crucial that the user interface is intuitive and accessible, enhancing the user's interaction with the application.

2
Background

This chapter introduces the signal processing, machine learning, and triangulation algorithms to understand the implementation described further in chapter 3.

2.1 Signal Processing

Signal processing is often the first step in doing any kind of machine learning on audio data. The way the training data as well as the data used for prediction is processed will influence the features that are extracted from the signals and affect the accuracy of the system. It is therefore important to have a good understanding of the basic theory behind signal processing before working with machine learning. The theory necessary to understand the pre-processing of signals for this implementation is described below.

2.1.1 Sampling Signals

Audio can be represented as a continuous signal with any segment of it containing infinitely many data points. In order to represent this signal with a computer, you have to take a number of discrete sample points of the continuous signal in the interval of a second, known as the sample rate. The sample rate limits the amount of data obtained on account of bandwidth and storage limitations, which directly affects the resolution of the information that the signal can be represented by. According to the Nyquist theorem, the sample rate should be higher than at least double the largest frequency component observed in a signal to have an adequately high resolution to represent it with a discrete sequence of amplitudes [7, pp. 267-285]. This means that to get the frequencies in the interval [0 Hz , 4 kHz] of a signal, it is necessary to sample it with a minimum rate of 8 kHz and filter out frequency components above 4 kHz to avoid aliasing of the signal.

2.1.2 Spectrogram Conversion

Once the discrete sample array of the signal is obtained, it needs to be processed to the format intended to be used as an input for the ML model. In theory, it is possible to either use the raw signal that has been sampled, or process it further by taking the Fourier Transform of the signal. In the first scenario, the ML model would only get time-domain information, neglecting the frequency components of the signal. In the second scenario, the ML model would only get frequency-domain

information and lose all time-domain information. An improvement is to generate spectrograms which utilize both time-domain and frequency-domain information.

A spectrogram can be described as a diagram of an audio signal in respect to frequency and time. The way this is achieved is by quantifying the audio signal into equally sized segments called frames, and then performing the Fourier Transform on each frame. Each segment represents a part of the signal's frequency components in a specified period of time, this retains some information about what frequency components were seen in the time of the frame. This is usually combined with first applying a windowing function, such as the Hanning Window, to the frame. The windowing function is meant to minimize spectral leakage that occurs due to the discontinuities at the end and start of the finite signals [7, pp 248-256]. Since some data is lost in this process, it is useful to overlap the frames, which will amplify the lost data and return the signal to its original state. The number of samples that frames are overlapped with is commonly called "frame step" and is conventionally set to one fourth of the frame size. The size of each frame is what determines the frequency contra time resolution. Increasing the size of a frame causes the frequency spectrum to be more accurate but decreases the time spectrum information, and vice versa. This entire process is what defines the Short Time Fourier Transform that generates a spectrogram.

2.2 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a type of neural network that is often used in image and audio processing classification tasks. The key feature of CNNs is that they can learn and extract hierarchical representations of input data by using convolutional layers.

Hierarchical representations refer to the idea that complex objects or concepts can be represented as a hierarchy of simpler components or features. For example, in image classification tasks, the input image can be represented as a hierarchy of features, where each layer of a CNN extracts more abstract and high-level features than the previous layer. The first layer may extract simple features like edges and corners, while subsequent layers may learn to recognize more complex patterns such as shapes and textures, and eventually, high-level concepts like objects or scenes.

2.2.1 Conv2D Layer

The Conv2D layer is the core layer in a CNN that performs the convolution operation on the input data. It applies a set of learnable filters to the input data and extracts features from it. The output of a Conv2D layer is a set of feature maps that represent different aspects of the input data.

A Conv2D layer has several parameters, including *filters*, *kernel_size*, *padding*, *strides*, and *data_format*, among others. Among these parameters, the most important ones are the number of filters, the kernel_size, and the activation function. The *filters* parameter determines the number of kernels that are applied to the input volume, and each of these convolutions generates an activation map, as shown in Figure 2.1. The activation map created by the filters provides information about

which features of the input data are relevant for a particular task. By analysing the patterns in the activation map, the CNN can make decisions about how to classify or process the input data. For example, in image recognition tasks, the activation map can highlight specific regions of the image that are important for identifying a particular object or feature.

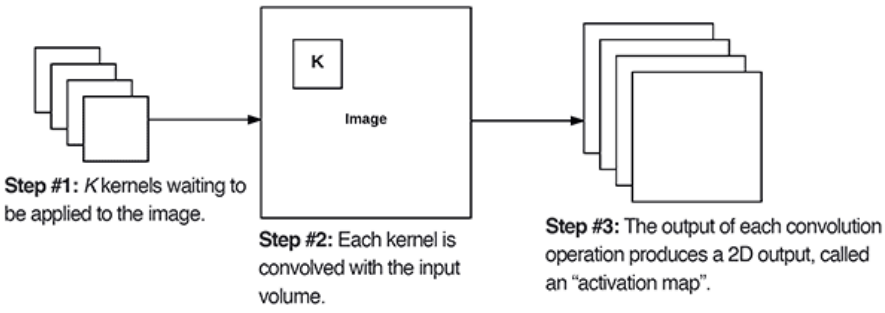


Figure 2.1: The Keras deep learning Conv2D parameter, filters, determines the number of kernels to convolve.

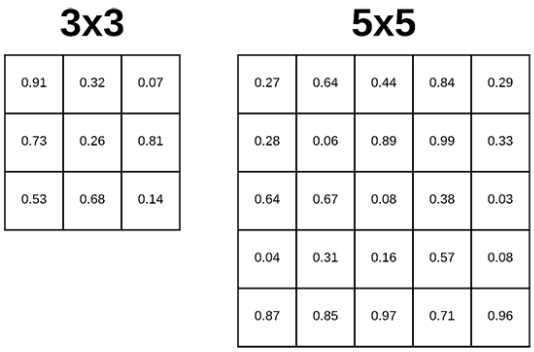


Figure 2.2: The *kernel_size* parameter in Keras deep learning's Conv2D determines the dimensions of the kernel. In the image on the left-hand side, the kernel size is 3x3, while on the right-hand side, it is 5x5.

Overfitting occurs when a machine learning model is trained too much on a particular dataset, causing it to perform poorly when presented with new, unseen data. Essentially, the model becomes too specialized and starts to learn noise in the training data rather than the underlying patterns that would help it make accurate predictions on new data.

2.2.2 MaxPooling Layer

MaxPooling is frequently employed alongside Conv2D layers in CNNs to reduce the spatial dimensions of the feature maps. These layers perform a downsampling operation by selecting the maximum value within each pool of adjacent values. For

a visual representation, refer to Figure 2.3. By focusing on the most important features, this technique can help to reduce the computational complexity of the model and prevent overfitting.

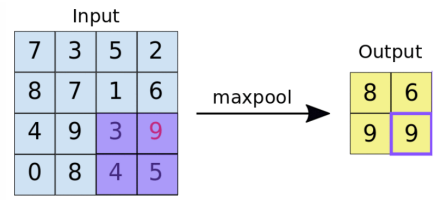


Figure 2.3: The process of reducing an input matrix into a smaller image using a MaxPooling operation. The input matrix is presented on the left-hand side of the figure, while the smaller output image is displayed on the right-hand side.

2.2.3 Dense Layer

The Dense layer, also known as a fully connected layer, is a basic layer that connects every neuron in one layer to every neuron in the next layer. It is commonly used in neural networks to perform the final classification task. The Dense layer applies a linear transformation to the input data followed by a non-linear activation function, such as ReLU. More about activation functions can be found in Section 2.2.6.

The number of neurons in a Dense layer is a critical parameter that is specified during model construction, as it determines the dimensionality of the output space. By increasing or decreasing the number of neurons in a Dense layer, the model can capture more complex or simpler representations of the input data, respectively. During training, the weights, and biases of the Dense layer are learned, allowing the network to make accurate predictions or classifications for new input data. Refer to Figure 2.4 for a visual depiction of a Dense layer in a neural network.

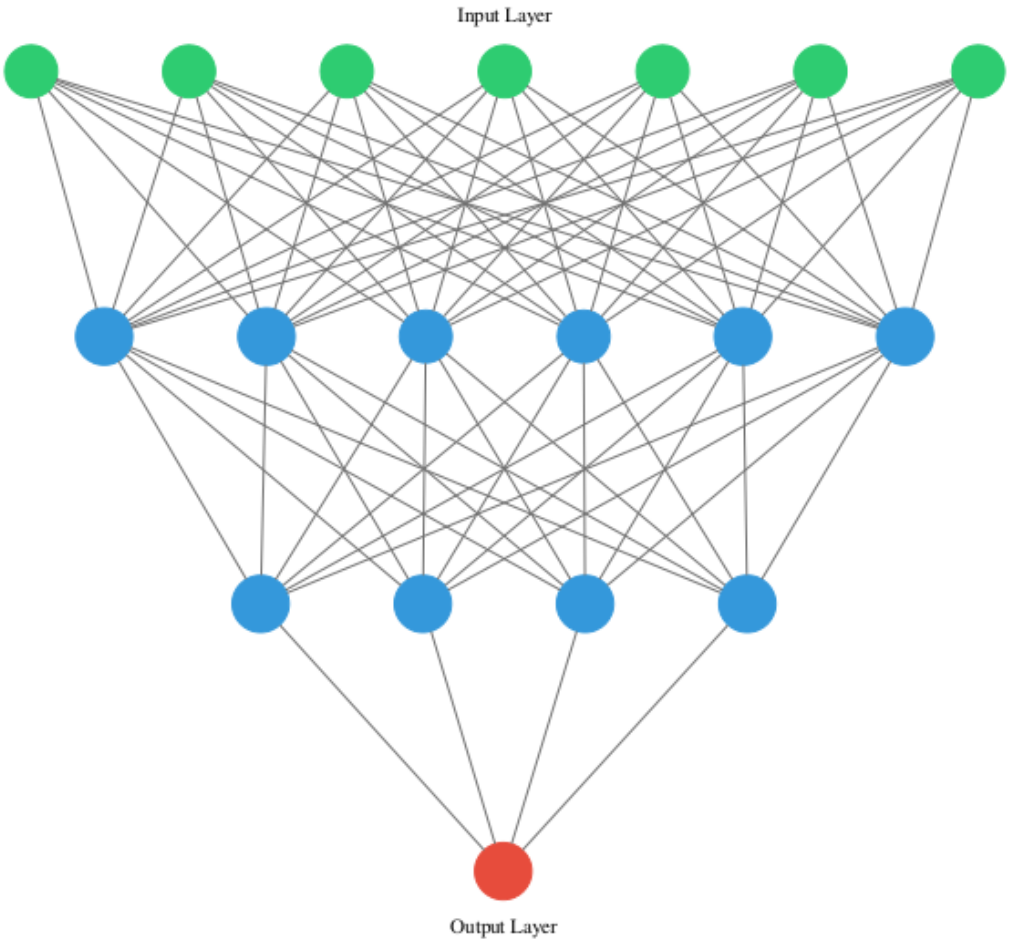


Figure 2.4: The blue layers in this figure are Dense layers, fully connected to the subsequent layer. The number of neurons in each layer is specified through the 'units' parameter, which is set to 6 and 4 in the first and second layers from the top, respectively.

2.2.4 Flatten Layer

The Flatten layer is a simple layer that is used to convert a multidimensional tensor into a one-dimensional tensor. It is often used in conjunction with convolutional layers in CNNs to flatten the output of a convolutional layer into a 1D tensor that can be fed into a fully connected layer.

2.2.5 Dropout Layer

The Dropout layer is a regularization technique used to prevent overfitting in neural networks. During training, it randomly drops out a given percentage of nodes in the layer, which helps prevent the network from relying too heavily on any one node or set of nodes and decreases the chance of overfitting. This encourages the network to learn more robust and generalized representations of the data. Dropout layers are typically added between Dense layers in neural networks. For a visual

representation, refer to Figure 2.5.

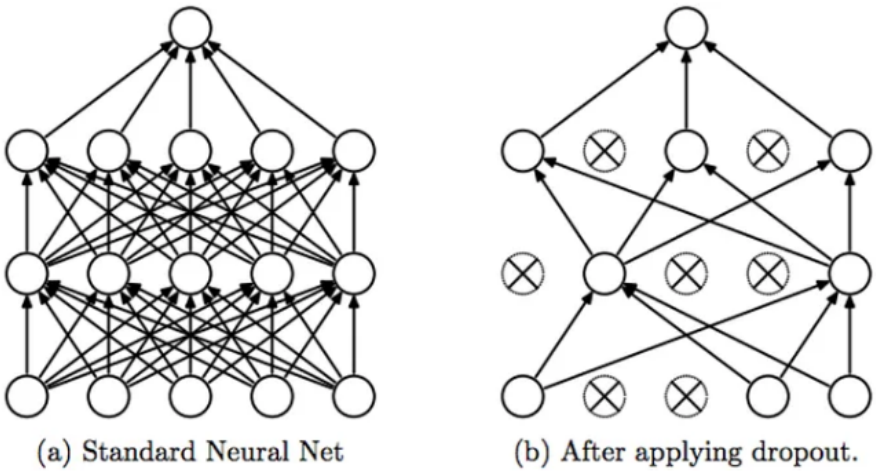


Figure 2.5: The figure depicts the neural network before and after the application of the dropout layer, in (a) and (b), respectively.

2.2.6 Activation Functions

Activation functions are an essential component of a neural network, as they introduce non-linearity into the output of a neuron. Non-linear activation functions, such as the commonly used ReLU and sigmoid functions, allow the network to learn more complex patterns and relationships in the data, making it more powerful and better able to handle real-world problems. By selecting an appropriate activation function, we can improve the performance of a neural network and enable it to learn more complex and meaningful representations of the data.

2.2.7 Hyperparameters

During the training phase of a DL model, it is important to change the hyperparameters to find a configuration that gives the best results. The hyperparameters involved depends on the structure of the model itself, but there are a few parameters that are always used. These are the learning rate, batch size, and number of epochs used for training.

The learning rate determines how quickly the model learns and how much it adjusts its weights during training. If the learning rate is too high, the model can converge quickly but may overshoot the optimal weights. If the learning rate is too low, the model may take longer to converge and may get stuck in a local minimum.

The batch size determines how many samples are used in each iteration during training. A smaller batch size can make the training process more stable and prevent the model from getting stuck in local minima. However, a smaller batch size may also increase the training time. On the other hand, a larger batch size can speed up the training process, but it may cause the model to converge to a suboptimal solution.

The number of epochs is the number of times the model goes through the entire training dataset. It is important to find the right balance between underfitting and overfitting. If the model is underfitting, it may not have enough epochs to learn the patterns in the data. On the other hand, if the model is overfitting, it may have too many epochs and may memorize the training data without generalizing well to new data.

2.3 Time of Arrival

Time of Arrival (TOA) is a technique utilized to determine the precise position and timestamp of a signal's transmission. The technique involves measuring the arrival time of a signal at multiple receivers while also knowing the positions of the receivers. A general formulation of Time of Arrival can be expressed as follows:

Consider a signal was transmitted at time $\boldsymbol{t}$ with its initial position given by $\vec{\boldsymbol{P}}$. Furthermore, consider $\boldsymbol{n}$ receivers with distinct position vectors given by

$$\vec{P}_0, \vec{P}_1, ..., \vec{P}_i, ..., \vec{P}_n$$

that receive the signal at arrival times given by

$$t_0, t_1, ..., t_i, ..., t_n.$$

Assuming a fixed signal propagation speed $\boldsymbol{c}$, one can then formulate a set of $\boldsymbol{n}$ equations relating the transmitter's position and transmission time with receiver $\boldsymbol{i}$'s position and signal arrival time:

$$\mathrm{d}(\vec{P}, \vec{P}_i) = c \cdot (t_i - t), \tag{2.1}$$

where $\mathbf{d}(\vec{\boldsymbol{P}}_\mathbf{1}, \vec{\boldsymbol{P}}_\mathbf{2})$ denotes the Euclidean distance between two points.

The set of $\boldsymbol{n}$ non-linear equations can then normally be solved for unknowns $\vec{\boldsymbol{P}}$ and $\boldsymbol{t}$ using non-linear optimization techniques. The solution set may, depending on the number of receivers $\boldsymbol{n}$, consists of one, multiple or an infinite number of solutions. Generally, to find a non-infinite solution set one would need one more receiver than the number of dimensions $\boldsymbol{m}$ of the position vectors. This can be derived from the fact that the total number of independent unknown variables is $\boldsymbol{m}+1$ ($\boldsymbol{m}$ unknown variables in position vector and one unknown variable as timestamp). To solve for $\boldsymbol{m}+1$ independent unknown variables, we would need $\boldsymbol{m}+1$ independent equations, thus at least $\boldsymbol{m}+1$ receivers. However, having $\boldsymbol{m}+1$ receivers does not in theory guarantee that a solution can be found, since there are edge cases where the set of equations are not all linearly independent. An example of this is the case where the receivers are located on a straight line, as seen in figure 2.6. This would yield an infinite number of solutions, since a signal sent from any point to the right of the receivers could have produced this result.

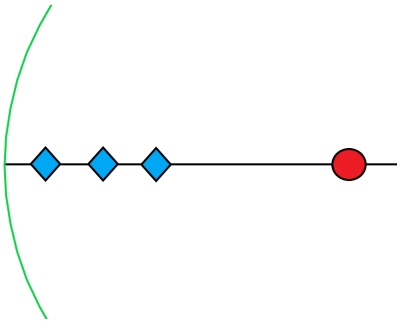


Figure 2.6: Example of a situation where position cannot be determined despite having enough receivers. Knowing when the receivers (shown as blue diamond) received the transmitted signal (shown as green), any position of signal transmission (shown as red circle) to the right of the receivers could have yielded a sound wave that would have produced these results.

Time of Arrival localization is sensitive to measurement errors in both receiver positions and arrival times. Furthermore, the assumption that the signal propagates at a constant speed is not always true in the physical world, since it often depends on what material it is passing through. An example is that sound travel approximately ten times faster in concrete than in air [8]. Bearing this in mind, it is important to consider the cost function to minimize when numerically solving the above equations.

Another method of determining signal position and transmission time is Time Difference of Arrival. It is very similar to the aforementioned technique, except that the equations are set up as the difference in arrival time of two receivers to determine the difference in distances to transmitter. This can similarly be solved for unknowns using non-linear optimization techniques. However, a report by Kaune [9] shows theoretically "no difference in TOA and TDOA localization", whilst TDOA bears more complexity.

3
Implementation

To achieve the requirements set for this system, we have identified four different problems that need to be addressed. First, the system has to be able to detect gunshots accurately. Second, the presence of gunshots has to be able to be reported to relevant authorities and the users of the system. Thirdly, the users have to be able to start recording audio for the ML model and give the necessary permissions to the system. Fourthly, the reports of the gunshots have to be processed to determine the reliability of the reports and to approximate the location of the gunshot.

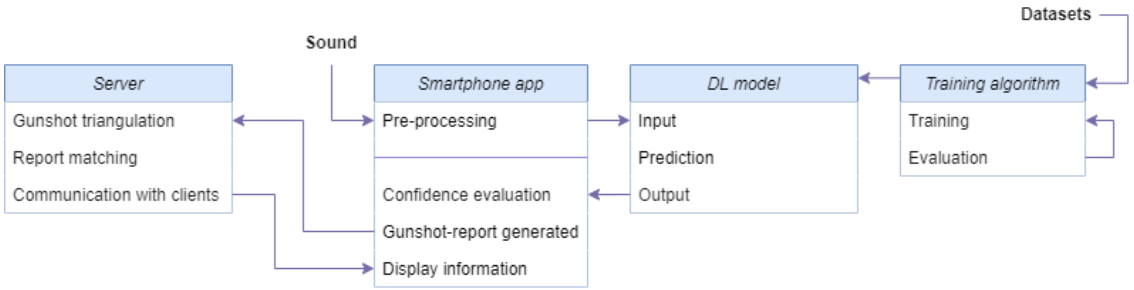


Figure 3.1: Overview of the implemented system with arrows describing the flow of information between system-components.

3.1 System Components

The requirements for the system's different components will be described next.

3.1.1 Android Application Components

The Android application for the Privacy Aware Gunshot Detection (PAGD) system requires several critical components. Foremost, it needs an effective audio recording mechanism to supply the AI model with necessary audio data. A user-friendly map interface is also essential to intuitively inform users of potential nearby gunshots. The application must maintain a reliable network connection for transmitting data to a central server, which handles gunshot reports from users. Most importantly, considering the application continuously records ambient audio and tracks the user's location to estimate the gunshot's origin, it must uphold rigorous privacy standards to protect this sensitive information. Each component plays a crucial role in the system's overall functionality and performance, demanding careful design and execution.

3.1.2 Sound Localisation Components

To achieve the goal of sound localisation, it was found that two key sub problems had to be solved. The first is that of correlating reports, to differentiate which reports originated from which shooting event in the case of multiple shooting happening simultaneously. This was done by splitting reports into events, further described in section 3.6.1. Secondly, one needs to determine the position where these shootings took place. This was implemented using Time of Arrival (described in section 2.3) which was seen as the only viable option from the limited information gathered aside from the very similar Time Difference of Arrival.

3.1.3 Server Components

The server design consists of several components. The first component is the communication between the server and the application, which happens through an API that was built to act as an interface for a seamless data exchange in a standardised way. The API is not meant to be used by everyone for security purposes, therefore it is necessary to also incorporate a token system to authenticate users to control who is allowed to access the server data and for how long. This also opens up the ability to better differentiate users from each other. The next component of the server is designed to handle grouping of incoming reports that belong to the same gunshot event, allowing for multiple reports from different places in the world to come in simultaneously without interfering with each other, thus providing more accurate results. This component is tightly coupled with the localisation component of the server that analyses the data from the reports to determine the origin of the gunshot. In order to store and manage the incoming reports, the detected gunshot events, and all of its associated information, a database was designed to keep this data together for efficient retrieval and processing.

3.2 Functionality of the Mobile Application

The Privacy Aware Gunshot Detection (PAGD) application is developed utilising Kotlin, a statically-typed programming language that is officially recommended for Android development. This app implements the Model-View-ViewModel (MVVM) architectural pattern, recognised for its efficacy in maintaining a clean and comprehensible code structure.

Endorsed by Android developers, the use of MVVM facilitates adherence to the single responsibility principle, an integral concept for clean architecture [10]. This principle asserts that each component or module should have one, and only one, reason to change, thereby enhancing the clarity, maintainability, and testability of the application.

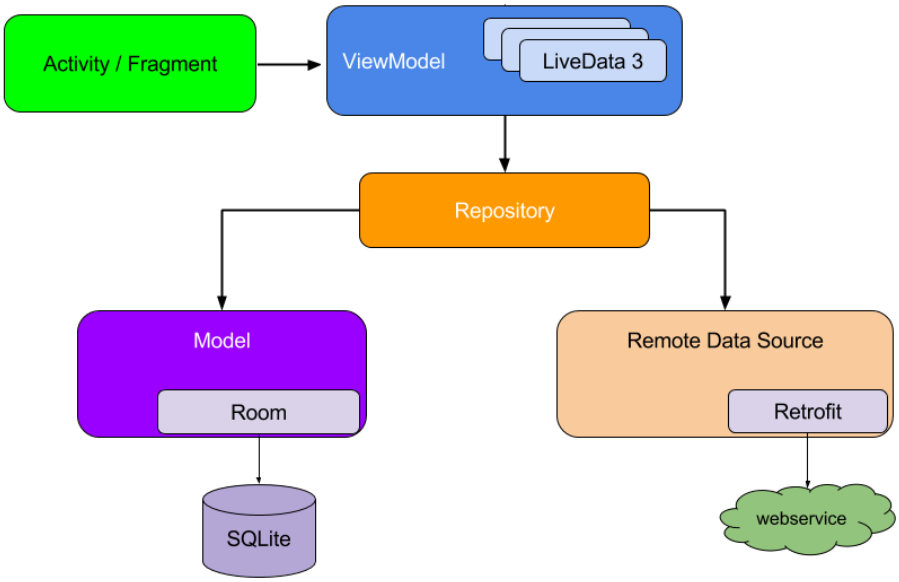


Figure 3.2: Basic structure of MVVM. From [1], CC BY 2.5

The diagram 3.2 provided depicts the basic structure of the MVVM architecture, and the similar structure is shown for the PAGD application in the diagram 3.3 down below.

3.2.1 MVVM Architecture

In the Model-View-ViewModel (MVVM) architecture, when a user interacts with the View component of an application, the View delegates commands to the View-Model. Subsequently, the ViewModel retrieves data from the Model or data domain. Upon receiving the data, the ViewModel, which serves as a data holder, exposes the updated information to the View. As the View observes changes in the data stored within the ViewModel, it responds accordingly and presents the updated data to the user.

The main concept in this architecture is that, when developing mobile apps, a developer must consider and handle the different life cycles of the components that make up an app. As a user navigates through an app, they primarily interact with a main Activity class, which typically contains various Fragments. Activities and Fragments display UI elements to the user. When a user navigates between different Fragments within the same Activity or performs a screen rotation, the current Fragment is destroyed and then recreated if the user navigates back to it. This process can cause the displayed data to be lost and may require refetching. To avoid this issue, a ViewModel is used, as it retains the data and is scoped to the specific Activity's lifecycle. This allows for the preservation of the data even when the underlying UI components, such as Fragments, are destroyed and recreated.

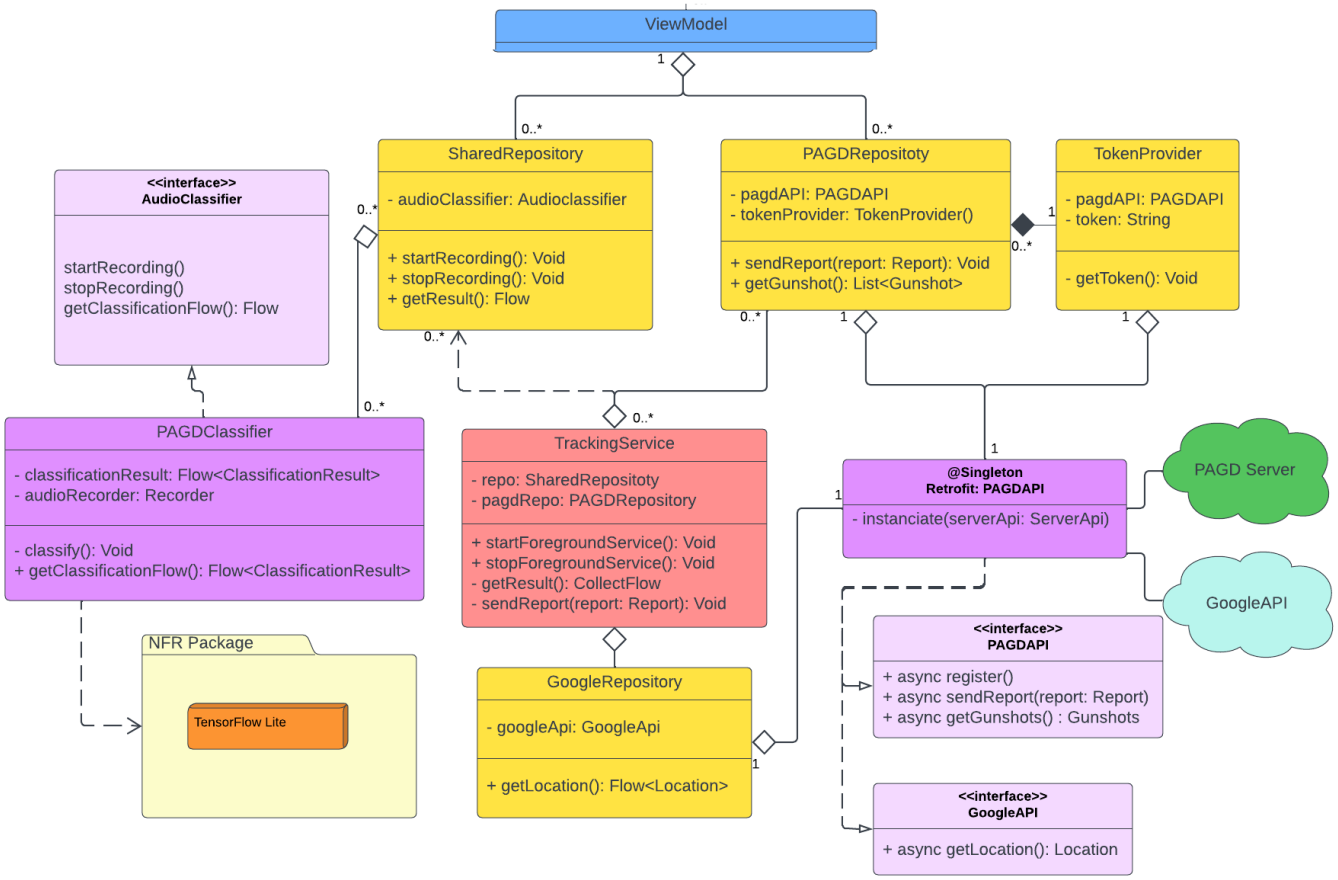


Figure 3.3: Class diagram of the PAGD app

3.2.2 Specific Requirements

Unfortunately, the standard approach is still insufficient for this specific project, as the application's intended functionality requires continuous operation, recording the user's surroundings and obtaining their location regardless of active user interaction. Due to security constraints within the Android framework, applications are not permitted to utilise the microphone or access the user's location while running in the background. To address this requirement, the application employs a foreground service from the Android framework, which is capable of executing long-running tasks in the background, provided that the user is notified of the service's operation.

3.2.3 Flow of the App

The primary responsibility of the foreground service is to obtain the results from the audio classification model and the user's location and transmit the report to the PAGD server while retrieving potential gunshot events from the same server. Upon the user activating the recording functionality (assuming the necessary permissions have been granted), the foreground service initiates and continues to operate in the background, even if the user terminates the app through the task window.

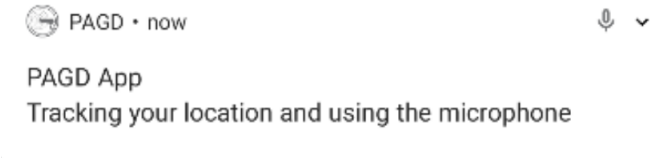


Figure 3.4: Notification from the foreground service

This design decision considers situations where a user might unintentionally close the app, thus losing the benefits of its functionality. In order to preserve security and integrity, the user is unable to dismiss the notification until they deliberately stop the service.

3.2.4 Reporting Gunshots

Upon activation, the foreground service initiates the operation of the PAGDClassifier, thereby commencing the audio recording process. The PAGDClassifier model, representing the essential functionality of the application, processes the captured audio through the AI model. This processing generates a probability score indicative of the likelihood of a detected gunshot based on the recorded audio data.

If the derived probability score exceeds a pre-established threshold, the respective data is relayed to the foreground service for further action. Upon receiving this data, the foreground service executes an asynchronous call to the GoogleAPI with the aim of fetching the user's geographical location.

Successful acquisition of the location data triggers the creation of a comprehensive report, encompassing vital details such as the geographical longitude, latitude, and altitude, along with a timestamp corresponding to the occurrence. This report is subsequently transmitted to the PAGD Server, facilitating further analysis and responsive action.

Both the Google API calls and classification process are executed within a coroutine, allowing the code to run concurrently with the app's other components. It is worth noting that the result from the PAGDClassifier class are emitted through a flow, a Kotlin framework built on the language's coroutines library. This enables working with asynchronous data streams that the foreground service can collect from.

```
private suspend fun sendReport() {    // asynchronous
    serviceScope.launch mainLaunch@{
        sharedRepository.getActiveClassifierResultFlow()
            .collect { result -> // Collecting from the flow
            serviceScope.launch {
                 val location = withContext(Dispatchers.IO) {
                    locationClient.getLocation(3) // asynchronous
                } ?: return@launch

                val report = Report(
                     result.timestamp,
                     location.latitude.toFloat(),
                     location.longitude.toFloat(),
                     location.altitude.toFloat(),
                     result.gun
                  )
     val result = pagdRepo.addReport(report) // asynchronous
            }
        }
    }
}
```

Listing 3.1: Asynchronous method to collect result from the AI-model and send reports to the server

Flows also facilitate the implementation of buffers, which are added to the flow's producer, in this case, the PAGDClassifier and the GoogleRepository. Since the foreground service is responsible for continuously sending reports to the server upon detection, it may not have sufficient time to complete the API call before another detection occurs. Fortunately, due to the buffer, all emitted results from the classification model are stored and collected by the service once it is ready to gather results again.

3.2.5 Network Implementation

The connection between the PAGDRepository and the PAGD server is facilitated using the Retrofit library, a high-level HTTP client. As the endpoints to the PAGD server necessitate an authorisation token for each call, the app implements a token provider that retrieves the token upon the application's startup and stores it on the app's storage device. This token is then supplied for each call to successfully access the API endpoints. Given that a token is valid for only 30 days, the app verifies its age upon startup and requests a new token if the current one is older than 30 days. This approach prevents the app from making superfluous calls to the PAGD server and only retrieves a token when required. Since the app is designed to operate indefinitely in the background and is never intended to be shut down, the token verification process occurs solely at the app's startup. To ensure a robust construction, a callback function is implemented for each API call. If the server returns a 401 (not authorised) response code, the callback is triggered and a new

token is acquired.

3.2.6 Receipt and Interaction with Gunshot Alerts

The application harnesses the potential of Firebase's event messaging system for the delivery of notifications related to gunshot detection. This system triggers an alert broadcast to all devices subscribed to the Firebase database when a gunshot event is identified.

When such a notification is received, the foreground service immediately attempts to ascertain the user's current geographical position. It then computes the spatial distance between the user and the site of the reported gunshot incident.

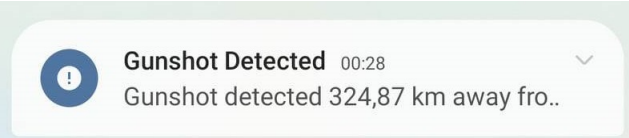


Figure 3.5: Notification of gunshot

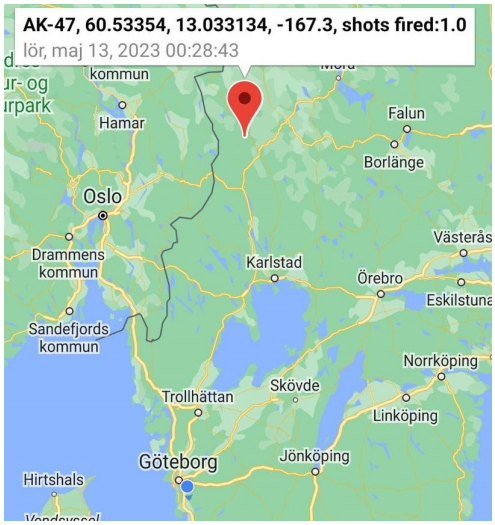


Figure 3.6: Gunshot displayed on map

Regardless of the user's ongoing interaction with the application, actioning the notification prompts the app to launch automatically. Upon launch, the app focuses on the precise location of the gunshot incident on a built-in map, providing a visual marker of the event's origin. This functionality ensures immediate notification and comprehension of the situation, whether the user is actively engaged with the app at the moment of alert or not, thereby facilitating prompt awareness and potential response.

3.3 Deep Learning Model

The deep learning model is a fundamental component of the mobile-based gunshot detection system. This section outlines the model's architecture, layering, and structure used to accurately identify gunshot sounds in real-time. By optimising the deep learning model's performance, the system aims to improve the effectiveness and reliability of gunshot detection and reporting. For the background behind the chosen architecture, review Section 2.2.

The deep learning model uses convolutional and deep layers, allowing the model to capture spectral features of spectrograms of gunshot sounds, increasing its ability to distinguish gunshots from other sounds.

3.3.1 Layering and Structure

To start building the convolutional neural networks (CNNs) we defined a 2D convolutional layer with a varying number of filters between 16 and 64 depending on the model used, with kernel size 5x5 to extract local features from the input spectrogram. We then included one or more Conv2D Layer with 3x3 filters to process the output of the first layer, which reduces the output size to the subsequent layer. Before the dense layers, the model uses a flatten layer to flatten the output of the convolutional layers from a 3D tensor into a 1D tensor in order to be used in the fully connected layer that consists of 128 or 256 neurons that uses RuLe as an activation function.

The last layer used was a Dense layer with a single neuron that produces a probability output which tells if the output is a gunshot audio or not. For the models used in the project, dropout layers were used which ignores 10% to 50% of the neurons, in a random fashion. This helps to prevent overfitting on the training data.

Finally, to minimise the loss, the models make use of an optimisation algorithm called Adam (Adaptive Moment Estimation). This algorithm is a crucial component, responsible for updating the model's parameters while training. This function determines how much loss the predictions will result in, based on how much the predictions of the model deviates from the expected value derived from the samples' category. The final models can be found in Appendix A, and will be studied in Section 5.1.

3.3.2 Datasets

Datasets are a crucial component of machine learning models, as they provide the training data necessary for the models to learn from. The quality and size of the dataset can significantly impact the performance of the model.

Cleaning the dataset was done through filtering and normalising the data, which involved removing any irrelevant or duplicate data points, and scaling the data to a similar range, respectively. This can help improve the accuracy of the model. This will be described further in Section 3.4.1.

Additionally, it is important to ensure that all recordings in the dataset are of

equal length when processing it. This is because most CNNs require inputs of fixed sises, and varying lengths of input can cause errors in the training process. One common approach to handle this is to take the longest recording in the dataset and set it as the standard length, padding the shorter recordings with zeros to match the length of the longest recording. However, it is important to note that padding with zeros can also introduce some problems. For example, if the padding is too long, it can introduce a large amount of unnecessary data, which can increase the computational cost of training the network. We decided to extract the gunshots from the audio recordings which significantly reduced the amount of padding necessary, this process will also be described in detail in 3.4.1.

Finally, it is important to carefully curate the dataset to avoid bias and ensure that it is balanced across different categories. In this implementation, this involved keeping a balance between the gunshot sounds that were used as positive data, and all other sounds used as negative data. A well-curated dataset can greatly improve the accuracy and generalisation of the model, making it more effective for real-world applications. The method of creation of datasets are described in Section 4.1

3.3.3 Hyperparameters

Hyperparameters are a crucial aspect of training machine learning models. They are the parameters that determine how the model learns and how it performs. It is essential to tune these hyperparameters to achieve the best performance possible, review Section 2.2.7. Some of the most important hyperparameters to consider when training a model include the learning rate, the batch size, the amount of neurons used in each layer which affects the complexity of the model, and the number of epochs.

The hyperparameter tuning can be a time-consuming and iterative process, requiring multiple training runs to find the optimal values. One common approach is to use a grid search or a random search to explore different combinations of hyperparameters and evaluate their performance on a validation set. Bayesian optimisation is another popular technique that can optimise the hyperparameters more efficiently in a more automatic manner. However, it was decided for this project that the optimisation would be done manually to get a better and more thorough understanding of the solution.

In our implementation we started with just a few epochs and increased it as we got more complex models, until it reached 50 which gave us the best results. More epochs could have been used, but at some point overfitting is introduced, which is why more epochs were not used. For the batch size, it was found that 150 and 200 was a good size, giving good results while training the AI model. The learning rate chosen for the models ranged between 0.1 and 0.0001. Different values were chosen to find the optimal learning rate for the model, in order to optimise the model performance.

On several deep learning problems, Adam has been demonstrated to perform better than other optimisation algorithms, especially when it comes to overfitting problems. Besides, it is renowned for its quick convergence and strong generalisation abilities.

3.4 Signal Processing

Audio used for training has to be pre-processed to extract the necessary features to accurately detect gunshots. For this implementation, the muzzle flash could be detected by camera and used for the gunshot detection, but this would be inefficient if the smartphone is out of view of the flash, as well as posing some additional privacy concerns. Therefore, only the muzzle blast is considered, due to it being detectable from a longer range. However, the features can differ between gun models and type of ammunition, making the problem more complex. This is why a large set of data containing multiple weapon types and calibres has been gathered to train the model, see Table 4.1. The algorithm used for training was developed together with the signal processing due to its reliance on the data-structure.

3.4.1 Pre-processing

Pre-processing audio for machine learning applications is commonly done by transforming the signals into spectrograms, Mel spectrograms, or Mel-frequency cepstrum coefficients [MFCC]. Mel spectrograms and MFCC are variations of spectrograms meant to amplify specific features of the data, designed to better represent human hearing for music and speech recognition. It is unclear if these alternative methods will result in better gunshot recognition without testing and comparing each method, which is why regular spectrograms are used for this implementation.

In a spectrogram such as in Figure 3.7 a) the gunshot appears as a column of densely packed frequency-domain information. According to a paper on the acoustical characterisation of gunshots, the signal of a gunshot should be detectable for approximately 3ms, but according to the training-data collected for this project the characteristics of the gunshot tends to remain present for up to 200ms [11]. It seems reasonable to assume that this is a product of the environment where the gunshots are recorded and how it was processed, and that a majority of the data is repetitively captured audio introduced by echo. The accuracy of detection could therefore be more or less effective depending on where the gunshot is recorded. Another factor that affects the accuracy is the presence of background noise, such as someone clapping their hands together, see Figure 3.7 b).

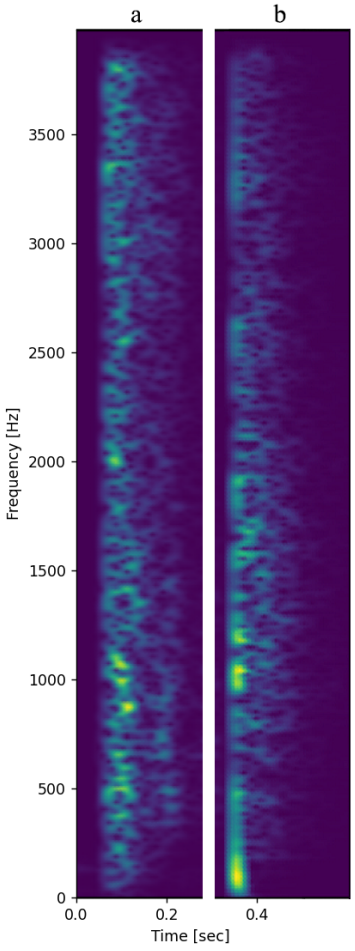


Figure 3.7: Overlaying of spectrograms a) Gunshot,
b) Hand-clap

Considerable noise will be present in real use of the system, that is why a set of data containing noise from different sources has been gathered to train the deep learning model to exclude these sounds from the gunshot detection. For an overview on how the datasets were created, see Section 4.1. Although deep neural networks can train on both positively and negatively labelled data to recognise noise, they have limited capabilities of detecting noise that has entered the positively labelled data. In this implementation, much of the positive training data contained significant noise that caused most of the signal's energy to be concentrated in the 0 to 1 kHz range. To address this, the data needed to be filtered to remove the noise. A 5th order Butterworth band-pass filter was used with the high-pass cut-off frequency between 0.6 kHz and 1 kHz, and the low-pass cut-off frequency set to the 4 kHz Nyquist frequency of the down-sampled signals, view Section 2.1.

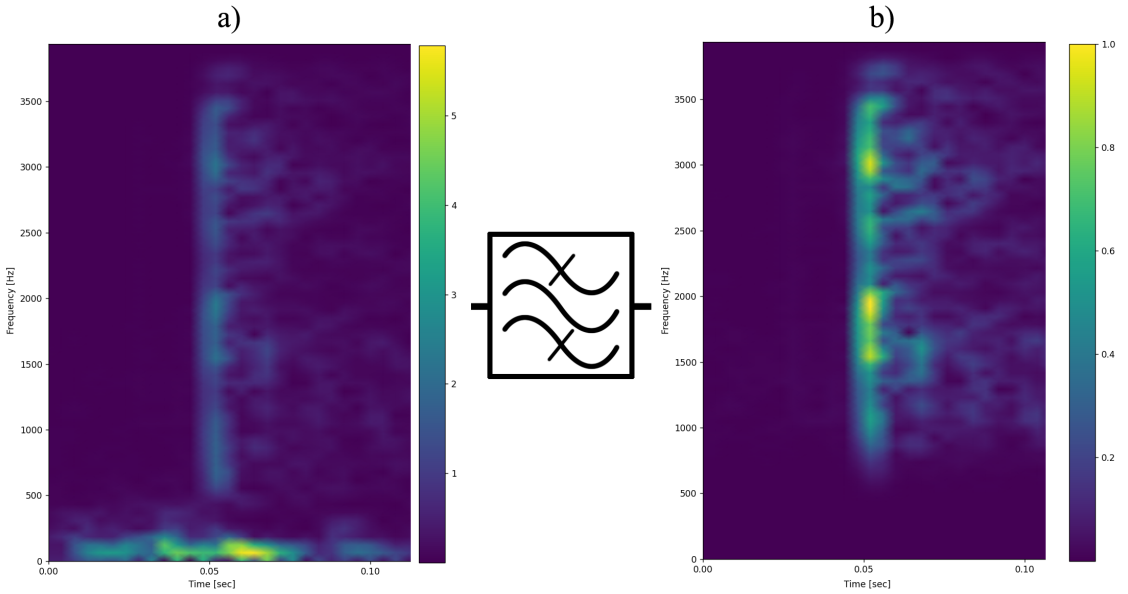


Figure 3.8: Spectrogram comprising unfiltered gunshot and wind sounds, as well as a filtered version that utilise a band-pass filter with amplitude normalisation, in a) and b) respectively. Note that the figures have different scales, which represents the normalisation of the amplitudes. The spectrograms have been cropped to fit the figure.

The differences between the raw and filtered signal in figure 3.8 shows how the noise from the wind has been removed, resulting in cleaner data that better represents the features of the gunshot. Also note that the different scales of the figures indicates a normalisation of the spectrogram to the amplitude of 1 with respect to the maximum amplitude of the gunshot, which was achieved through amplitude-normalisation after filtering. These factors contribute to the efficiency of the gunshot detection by constraining the amount of noise that can be captured to the frequency range where the gunshot has most of its energy, as well as keeping the amplitudes of the spectrograms more consistent. This helps the model to learn the defining characteristics of a gunshot by ignoring variations in amplitude caused by changing recording conditions, which is something that otherwise could negatively affect the

training of the DL model due to the large variation present in the positive data sets that were used.

Finally, the signals were shortened from the duration of approximately 2 seconds that the original audio clips had to 250 ms by finding the approximate location of the gunshot in the signal and extracting the surrounding area. This had two positive effects on the training of the model. Most significantly, the gunshot covers more of the area in the resulting spectrograms which decreases the opportunity for the machine learning algorithm to detect the absence of noise as an indication of a gunshot, as well as for the gunshot to appear at different locations in the spectrogram. Secondly, the training of the DL model was made approximately 8 times faster and lighter due to the decrease in memory requirements, which enabled a more efficient evaluation process. The negative effects of the length of the spectrogram could be further remedied with data-augmentation, which in this application would mean to digitally move the gunshots around in the spectrogram. This technique was attempted for a part of the evaluation process, and its results will be discussed further on in section 5.

3.4.2 Signal Processing Module Functionality

The processing of data and training of the DL model was performed on a computer using the programming language Python for its efficiency and simplicity. Due to the large set of data that needed to be processed every time a model was trained, a function was created that pre-processes and exports all data to JSON so that it can easily be imported upon training, view preProcessAudio() from Listing 3.2. In the pre-processing stage, the individual audio files were loaded and down-sampled to 8 kHz. This is the lowest sampling rate that most smartphones can capture, which happens to be suitable for this application as it keeps the battery consumption low but can still accurately represent the frequency range of gunshots that have most of its energy located in the 4 kHz bandwidth range. The signals were then processed differently depending on if it was supposed to be negative or positive data. In the case of it being positive data, the approximate location of the gunshot would be located and extracted to a normalised length of approximately 250 ms. In the case of it being negative data, the entire audio clip would directly be segmented into a set of 250 ms signals. The spectrogram(s) would then be processed as described in Section 3.4.1, with the frame size and frame step set to 128 and 32 samples respectively, which seemed like a good compromise between frequency resolution and time resolution, see Section 2.1.2. The spectrogram(s) were then appended to data to be exported into separate folders depending on if it was positive or negative.

The training of the model was then carried out through loading all JSON files from both the positive and negative training data sets, see trainModel() in Listing 3.2. 5% of all data would be reserved for the evaluation step and the data would be zipped together with a 0 or 1 indicating if it was negative or positive data. The training data was then shuffled to minimise bias and placed into smaller batches and cached for efficiency. The loaded model was then trained with the data, saved, and finally evaluated with the test data. It is also possible to evaluate the model further with other data not located in the training data sets.

```
# Pre-processes audio into spectrograms and exports to JSON files
def preProcessAudio(paths):
    for folder in paths:
        data = []
        for path in folder:
            #1 Load audio as a discrete signal
            #2 Down-sample signal to required sample rate
            if data is positive:
                #3.1 Filter out 1 kHz from signal
                #3.2 Find index of gunshot
                #3.3 Extract 250 ms with gunshot from signal
                #3.4 Band-pass filter and normalize signal
                #3.5 Transform to spectrogram and add to data
            else:
                #3.1 Divide signal into 250 ms chunks
                #3.2 Band-pass filter and normalize signals
                #3.3 Transform to spectrograms and add to data
        #4 Save data to JSON file
    return

# Trains a model and evaluates its performance
def trainModel(positiveData, negativeData):
    filePaths = positiveData + negativeData
    testData = []
    trainingData = []
    for i in filePaths:
        #1.1 Load spectrogram from positive & negative data
        #1.2 Take 5% of data and add to testData
        #1.3 Insert the rest of the data into trainingData
    #2 shuffle trainingData
    #3 Cache and batch trainingData and testData
    #4 Set class-weights to 3:1 for negative and positive data
    #5 Train model with trainingData and class-weights
    #6 Save model
    #7 Evaluate accuracy of model with testData
    #8 Evaluate model further with other data
    return
```

Listing 3.2: This pseudo-code illustrates how some of the data pre-processing, DL model training, and evaluation was performed.

3.5 Server Implementation

A server is needed to gather reports of gunshots reported by clients in order to determine where the gunshot possibly came from. An API was developed to handle this communication with the server and the clients using the Python web framework Flask [12]. The server was built with security and modularity in mind, and it consists of multiple parts, which will be discussed in more detail.

3.5.1 Design Principles and Patterns

One of the key aspects of the server implementation was to ensure it was built in a modular, flexible, and maintainable manner by following some of the common design principles and patterns. In particular, the SOLID principles were applied, which are arguably the most important ones for object-oriented programming. SOLID is made up of five principles:

- **S**ingle responsibility principle: Each task of the server has been divided into its own class, being responsible for only one thing and thus having only one reason to change. By doing this, each class has a clear and well-defined purpose, making the code maintainable and easier to understand.
- **O**pen-closed principle: The classes have been written to be open for extension but closed for modification, meaning that functionality can be added without changing existing code. This makes the code more flexible and less prone to errors.
- **L**iskov substitution principle: Superclasses have been designed to be replaceable by its subclasses while keeping the same functionality. For example, an instance of PagdDB does not change the functionality of its parent class Database. Adhering to this principle makes the design more robust and less likely to encounter unexpected behaviour when new code is added.
- **I**nterface segregation principle: The interfaces have been designed to only include the methods that are actually needed by the modules that use them. Interfaces that are designed this way makes the code more maintainable and easier to understand.
- **D**ependency inversion principle: The classes were designed to depend on abstractions (their corresponding interface) instead of on concrete implementations. By doing so, the implementation of a class can be changed without affecting the rest of the code that depends on. This is because dependencies do not need to care how something is implemented, only that an implementation exists.

Apart from the SOLID principles, more design choices were made to ensure modularity and flexibility. One of them being the implementation of the observer pattern for notifying the sound localiser of incoming reports. This design choice was necessary in order to separate the two applications, preventing the server from becoming tightly coupled and maintaining high cohesion. The observer pattern consists of a subject (the API) and one or more observers (the sound localiser). The role of the subject is to keep track of observers by allowing them to subscribe to its updates. The observers do not know how or where the updates are coming from, all they are

interested in is when there is new data to fetch. When the API receives an incoming report, the subject calls its notify method, which loops over all of its subscribed observers and publishes the report to them. Thereafter, the observer processes the data and passes it on to the sound localiser for further processing.

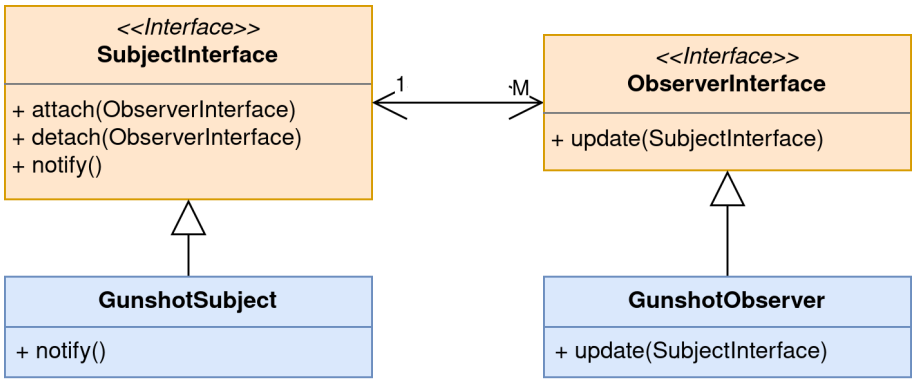


Figure 3.9: UML diagram of the observer pattern

3.5.2 Security

Handling sensitive user data over the Internet and storing it on a server puts a significant responsibility on developers and app owners to maintain its integrity by keeping it safe from intruders. Therefore, it was vital to take certain security measures into consideration. These measures include encrypting the communication over the web by using the HTTPS protocol signed by the trusted certificate authority *Let's Encrypt* [13]. Moreover, API calls are required to hold a valid token in the *Authorization* request header, which can be acquired after registering. The tokens, more specifically the *JSON Web Token* (JWT), are encoded and decoded using a generated 256-bit private key which is stored separately from the server on the Linux machine. The JWT tokens allows the API to not only verify that the client is allowed to make a request, but to also store necessary information in them, such as an expiration date and a unique client ID. The client ID is a generated *Universally unique identifier* (UUID), which is a 128-bit string used to differentiate clients from other clients.

3.5.3 API Endpoints

Creating an API involves defining different endpoints that users use to access certain resources. These endpoints were designed according to the RESTful [14] standard style in order to be simple and intuitive to use. One aspect of this design standard is to make use of HTTP methods. In this API, the GET and POST methods are being used for fetching existing data and adding new data respectively. Another part of the RESTful standard includes endpoint naming requirements for creating a clear and consistent interface. To achieve this, names must be descriptive and follow a consistent pattern by using plural nouns and hierarchical naming with the parent resource being listed first (see below for examples of the endpoints defined in the API).

GET	**/register**	Retrieve a JWT token used to authorise API calls.
POST	**/api/guns**	Add a gun to the database
GET	**/api/guns**	Search for a gun in the database
POST	**/api/reports**	Add a report to the database
GET	**/api/gunshots**	Search for gunshots based on time or location

Table 3.1: Defined API endpoints

The use of Flask provided an easy way to define these endpoints. Flask uses Python decorators, namely @app.route("<URL>"), to define URL routing by specifying the URL pattern and the function that should be called when a request matches said pattern. When a request comes into the server, Flask parses the URL and extracts the endpoint and any arguments that may have been passed in. After parsing, Flask matches the endpoint against the defined routes. If a match were found, the attached function takes care of the request and returns a value to the user along with an HTTP response status code 200 signalling that no errors occurred. If no match were found, Flask aborts the request with a 404 Not Found error.

HTTP response codes are a useful way of informing clients about the status of their request in a standardised way. There are many different status codes, all of which belong in different categories. This API make use of the 200 (informational), 400 (client error), and 500 (server error) level categories [15], all of which can be seen along with its explanation down below.

200	**OK**: no errors occurred
400	**Bad request**: missing required parameters
401	**Unauthorized**: invalid token
500	**Internal Server Error**: something went wrong processing during of the request

Table 3.2: HTTP response status codes used in the API

3.5.4 Database Structure

A database was required for reliable and efficient data storage to keep track of incoming gunshot reports and storing confirmed gunshot events. MariaDB [16] was chosen for this purpose as it is known for its high performance and strong security. Being able to rapidly store and process reports is vital for cutting down the response time after a gunshot has been detected, and security is a necessity when storing sensitive information such as people's GPS coordinates.

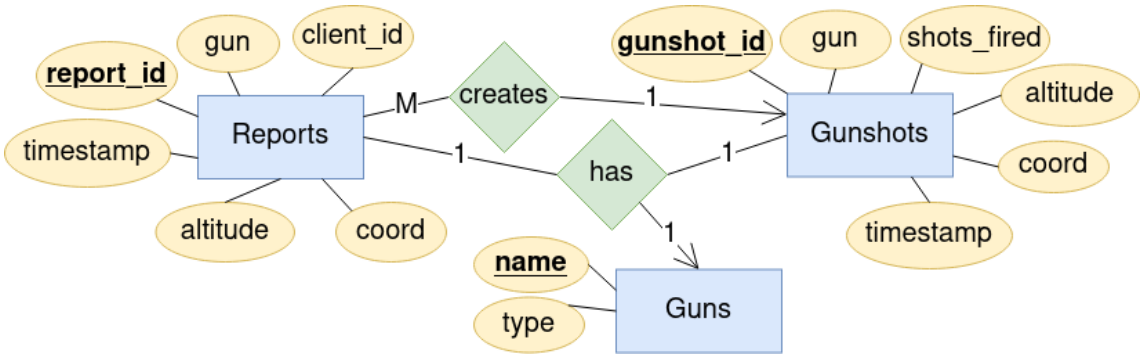


Figure 3.10: An entity relation (ER) diagram of the database structure

The database consists of three tables Reports, Gunshots, and Guns. Reports stores the information sent from the app after a gunshot was detected. Gunshots stores the estimated information that the sound localiser calculated based on a number of different reports. Guns serves as a dictionary of gun data that is known to the app. An entity relation diagram can be seen in figure 3.10 describing the table relations and their columns.

3.6 Sound Localisation

Localising a gunshot involves determining the location of the sound source based on reports from multiple distributed smartphones. This task can be divided into two sub-problems:

- The first sub-problem consists of correlating reports from different sensors to determine which reports are likely to be from the same gunshot event. This is crucial to be able to mathematically determine gunshot position and reduce errors.
- The second sub-problem comprises using the correlated reports to determine the position of the gunshot. This was achieved using mathematical methods such as Time of Arrival described in chapter 2.3.

3.6.1 Correlating Reports

In our implementation, only three pieces of information could be gathered from a gunshot report sent to the server. Firstly the GPS position of the reporting smartphone, secondly a timestamp in milliseconds of when the gunshot was detected on the smartphone, and lastly a unique ID for each smartphone client to differentiate clients. With the limited information to work with, a set of rules were set up to correlate which reports belong to which gunshot events. The parameters of these rules were based on a general parameter $distance_{max}$ acting as the maximum distance a gunshot can be detected from. If two reports uphold these rules, they will be assumed to be related to the same event:

- The GPS position of the two reports are within a set distance of $2 \cdot distance_{max}$. This is based on the fact that a gunshot that can be heard by clients from

a maximum radius of $distance_{max}$ cannot have two clients within that radius further away from each other than $2 \cdot distance_{max}$.

- The timestamp of the two reports are within a set amount of time of $\frac{distance_{max}}{v_{sound}}$ where v_{sound} is the speed of sound. This is based on the fact that a sound traveling at v_{sound} only able to be heard for a distance of $distance_{max}$ can only be heard for a time span of $\frac{distance_{max}}{v_{sound}}$ seconds.

$distance_{max}$ was for this implementation set to 1 kilometre. Using these rules, reports were grouped into events based on correlation. Reports in events were only "loosely" correlated, meaning a report only has to be correlated according to all above rules to one other report in event to be a part of it. This "loose" correlation allowed an event to encapsulate multiple consecutive firings by the same perpetrator, which is often the case. Since an event produced by this algorithm will contain reports of several firings and thus multiple sound waves, only the first report by each client is used for determining position and timestamp.

3.6.2 Determine Position and Timestamp

Once reports had been grouped into events, a position and timestamp of the gunshot firing could then be calculated. Each report contained a GPS position consisting of latitude, longitude, and altitude over sea level in meters. Timestamp of report was given in milliseconds. Since the dimension of position vector is three, four reports were needed to determine position (see section 2.3). Four unknown variables were then set up, latitude, longitude, altitude and timestamp. With these four unknown variables, an error function could be formulated for each report $\boldsymbol{i}$ using equation 2.1 in section 2.3:

$$\mathrm{e}_i(\vec{P}, t) = \mathrm{d}(\vec{P}, \vec{P}_i) - c \cdot (t_i - t). \tag{3.1}$$

Distance $\boldsymbol{d}$ was estimated by first calculating the geodesic distance between the pairs of latitude and longitude coordinates, and then estimating the total distance by calculating the hypotenuse between the geodesic distance and the latitude difference. Whilst this is not entirely accurate, it is believed to be accurate for the relatively small distances worked with here (less than one kilometre). With a set of error functions for each report, a loss function was formulated using the common "sum of squares" often used for regression problems.

$$\mathrm{loss}(\vec{P}, t) = \sum_i \mathrm{e}_i(\vec{P}, t)^2 \tag{3.2}$$

An estimation for $\vec{\boldsymbol{P}}$ and $\boldsymbol{t}$ could then numerically be found by minimising the loss function using the Nelder-Mead algorithm. The algorithm, as Dr. Saša Singer describes it "is one of the best known algorithms for multidimensional unconstrained optimisation without derivatives" [17] which is exactly the type of problem faced. After finding a solution, loss of solution could then be analysed to see if estimation is plausible. In this implementation, a loss of 10000 or more was seen as an invalid solution.

4
Evaluation

The evaluation of the system was carried out during the period of several weeks where all the components of the system were tested to evaluate if they met the functional requirements and expectations set on quality in the beginning of the project. The evaluation of the deep learning model, the localisation of gunshots, and the system as a whole will be covered next.

4.1 Creation of Datasets

The negative datasets used to train the DL model were created in a combination between gathering data from online sources containing sounds from anything that is not a gun, and recording audio of things that were noticed to trigger the models during the evaluation, such as handclaps and other varied noises one might produce in the home. Since any data that is not a gunshot is considered negative data, the only requirement posed on it was that it needed to be varied and not contain any gunshots. While class-imbalance was of some concern, it was evaluated that a small amount of class-imbalance in favour of negatively labelled data was to be desired to achieve higher precision. Although this has a negative effect on the accuracy in classifying positive data, the classification can not be trusted if the precision does not meet expectations.

Of higher importance was that the positively labelled data was of good quality, being systematically obtained and capturing the characteristics of different gun models. However, during the creation of the datasets it was noticed that the majority of publicly available data of real gunshot sounds was haphazardly obtained and of poor quality with significant background noise. The dataset that would come to make up all of the positive data used for training was captured in the NIJ Grant 2016-DN-BX-0183 project, which fulfilled all the previously specified requirements on most of the samples [18]. It includes approximately 300 samples from each weapon type listed in Table 4.1. However, a considerable amount of data in this dataset either poorly captured the gunshot characteristics or contained noise from wind, coughs, police radio noises, and speech. Therefore, the best audio from the dataset was carefully selected to only include audio that had distinctively audible gunshots with minimal noise. The negative data mainly consisted of audio recorded for this implementation, with some contributions from various online datasets.

Handguns	Caliber	Rifles	Caliber
Colt 1911	-	Bolt action	.22
Glock	.45, 9mm	M16	5.56mm
HK USP	-	MP40	9mm
Kimber	.45	Remington 700	-
Lorcin	.38	Win M14	-
Ruger	.22, .357	WASR	-
SIG Sauer	9mm		
SpKing	.22		
SW22	.22		
SW38sp	.38		

Table 4.1: This table lists the weapon-type and caliber combinations that the system has been trained on (Unknown: -).

4.2 Evaluation of the Deep Learning Model

To evaluate the accuracy of the models and to ensure that they were properly optimised during the training phase, an extensive evaluation process had to be performed that covered many different DL model architectures, with different methods used for processing the data and configuring the training parameters. Three experiments that made up the evaluation process will be explained next, along with the general setup of the experiments.

4.2.1 Metrics

To analyse the resulting models, four different metrics were used: Loss, precision, recall, and F1 scores. The loss metric describes how much the predictions deviate from the expected predictions (0 or 1) on the tested samples. The threshold for predictions was set to 50%, which means that even if precision and recall is 100%, some deviation from the expected prediction can still be present. For example, if it is a positively labelled sample being predicted on and the prediction is 0.7 units, then it means that it was a correct classification but deviated from the target by 0.3 units. Precision is a metric that describes how many true positives have been observed out of all the positive predictions, which indicates how well the model classifies negative data. Recall describes how many of the positive samples that the model classifies correctly by dividing all the true positives by all the true positives and false negatives. The F1 score is commonly used for determining the accuracy of models trained on unbalanced datasets by balancing the recall and precision scores. It is calculated by the following formula: $2 * (Precision * Recall)/(Precision + Recall)$. These metrics together defined the view of accuracy that permeated the evaluation phase, with slightly higher requirements placed on precision than on recall.

4.2.2 Setup of Experiments

During the evaluation process, the signal processing steps, DL model architecture, as well as the hyperparameters were changed to study the effects that different configurations had on the accuracy of the models in an attempt to find a good solution. The method by how these changes were made was to progressively increase the complexity of the design while also taking the performance limitation of smartphones into consideration, balancing accuracy with processing speed. Models that were fundamentally different were trained on spectrograms obtained from different duration of audio processed into spectrograms, ranging from 2 seconds to 0.1125 seconds long. The spectrograms were performed with frame sizes 512, 248, and 128 samples, with the overlap of frames being either 32 or 16 samples. Since the amount of different configurations to test grows exponentially with the number of hyperparameters in each layer, the entire evaluation process took a long time before favourable results were achieved. To make the evaluation more efficient, the data was pre-processed separately to the training.

4.2.3 Main Model Evaluation

The main evaluation consisted of observing the training phase and evaluating it with test-data after the training was complete. It was useful to observe the training phase to see how the model improved for each epoch, as this helped to determine what the underlying cause could be if something went wrong during training. For example, if the learning-rate was set too high then the model would make too large changes to its network of neurons which causes unpredictability and a poor convergence to the minimum loss. On the other hand, if the learning rate was too small then it would take unnecessarily long to train the model as more epochs were needed to reach the best potential results for a specific setup. During the training phase, a part of the data sets that the model was trained on was removed for testing, ranging from 5 to 30% of each set. After the model was trained, it was evaluated using the test-data which produced loss, precision, and recall scores, along with the calculated F1 score which were then compared to previous runs.

4.2.4 Secondary Model Evaluation

A second evaluation was performed on the DL models with predicting on new positive and negative datasets that were not part of the training phase, producing precision, recall, and F1 scores. This was done to rigorously evaluate how well the model had generalised to new data that it was not specifically trained for. The positive data used was from different gun models and different recording environments than those in the training datasets to gauge how well the model could detect gunshots by other gun models than those that were trained for, as well as how it adapted to audio that had been recorded with different hardware. During this evaluation, the new negative data that was used was recorded on one of the smartphones that the system would be installed on. It should be noted that the secondary evaluation was less encompassing than the main evaluation, with the main intent of giving a rough, yet concrete, estimation of how well the model performs on unseen data. This

evaluation was used in conjunction with live testing to further validate the observed results in an attempt to minimise biases that may arise due to human intervention.

4.2.5 Live Testing the Model

After good accuracy was achieved, the models were imported into android studio for live testing through an android emulator. The emulator captured audio from a MXL 770 microphone being sampled at 8 kHz and made predictions using the evaluated models. The predicted values were printed on the console which were studied to determine how well the model reacted to real world scenarios with more varied negative data, as well as samples of gunshots that were played back on speakers. Since the emulator used a different type of microphone than smartphones have, this was only done to roughly and quickly assess the performance of the model. Once a model reacted well to this type of testing, we installed them on a smartphone and performed the same tests.

4.2.6 Verifying the Implementation Specifics

To make sure that the implementation specifics did not affect the evaluation process of the model, it was evaluated by manually importing audio to the emulator and running it through the signal processing and model prediction stages. The output was then compared to the expected output from the python program to see if they differed in any significant way.

4.3 Evaluation of Gunshot Localisation

To evaluate whether the server fit the requirements concerning receiving reports and accurately determining the position where the shooting happened, a module for simulating gunshots and reports was implemented. It served as a test of all key components of the server as well evaluating accuracy of localisation.

The module simulated a gunshot firing at a randomly selected set of coordinates in Sweden. For each gunshot simulation, a set of $\boldsymbol{n}$ clients randomly located 50 to 500 meters away were simulated. For each client, a connection was set up between the simulation module and the server was set up over HTTPS. Each client generated a token from the server and then sent reports to the server. These reports consisted of a timestamp as the calculated timestamp when the sound wave of the gunshot should have reached the client, as well as an additional timestamp error between 0 and $\boldsymbol{k}$ milliseconds. The reports also contained their generated position as well as a random GPS error randomly distributed between 0 and 15 m, as per the GPS Performance Standard [19] which reports a 95% global average vertical position error of 15 meters. When reports had been sent, a request for a gunshot within that timestamp was sent to the server. If the localisation algorithm converged on an answer, error in meters could then be calculated from the difference of estimated position and real position of the simulated gunshot. This also allowed for statistical probability of convergence, as seen in figure 4.1.

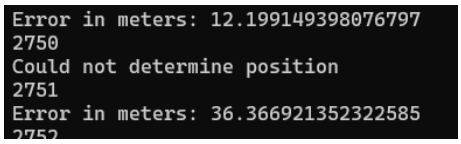


Figure 4.1: Simulation script running. Here we see that the localisation algorithm on the server failed to converge on an estimate of gunshot position

Parameter $\boldsymbol{n}$ was tested as 4, 6 and 8. Parameter $\boldsymbol{k}$ was tested as 10 and 100. For each combination of parameters, 10000 gunshots were simulated and results recorded.

4.4 Evaluation of the App

The application's architecture employs a modular design, facilitating the effortless replacement of components with minimal modification to the overall system. This approach proves advantageous when incorporating updated AI models; as long as the new model conforms to the Audioclassifier interface (see 3.3), it can be easily integrated, ensuring the application remains up-to-date with the latest technological advancements.

4.5 Evaluation of the PAGD System

To conclude this project, we performed live testing on a shooting range by predicting on gunshots at 10 m from the source of the gunshots. The predictions triggered reports to be generated that were sent to the server and reported back to the client. During this evaluation, the entire system was tested to verify that all the components worked in unison. The models D and E were used for testing, as well as an additional model that utilised an identical structure to models B and C.

5
Results

5.1 Deep Learning Model

After an extensive evaluation process over hundreds of DL models, we will describe the results from the evaluation process of a few of the more interesting models that were created, review Sections 4.2.3 and 4.2.4. The models selected for this discussion are the models A, B, C, D, and E, ordered from lowest to highest complexity. These models are described in the deep learning model evaluation report located in Appendix A. Please note that models B and C are based on a shared architecture, and models D and E also share the same architecture. The evaluation report contains all the results discussed in this section. We will then continue to discuss these results from a broader perspective in the next chapter.

5.1.1 Model Performance

A line chart in Figure 5.1 displays the historical values of each epoch for model E. Analysing the line chart, it can be observed that both precision and recall approached a value of 1 as the number of trained epochs increased. This trend holds true for all the mentioned models, indicating that the chosen models were capable of effectively learning the gunshot features. However, relying solely on this information may not accurately reflect the models' performance, as shown by the main and secondary evaluations in Table 5.1.

Upon closer examination of model E, it is evident that precision remains close to the expected value depicted in the line chart. However, recall drops to approximately 80%. This discrepancy could suggest mild overfitting, wherein the model starts to recognize patterns specific to the training data rather than general gunshot characteristics. To mitigate this issue, dropout layers were incorporated into all models. These layers randomly set a fraction of the neurons' learned values to zero, reducing bias and promoting a more balanced and generalized learning process.

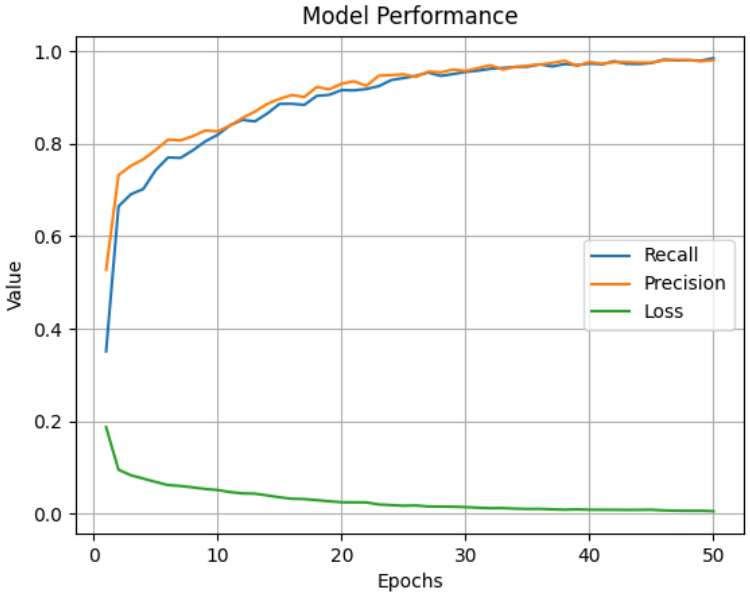


Figure 5.1: Progression of recall, precision, and loss scores of model E through 50 epochs of training

Model	Loss	Precision	Recall	Precision	Recall
A	0.0908	0.9543	0.9216	0.9736	0.9160
B	0.0278	0.9727	0.8699	0.9819	0.8202
C	0.0465	0.9561	0.7385	0.9834	0.8097
D	0.0273	0.8750	0.9484	0.9781	0.9228
E	0.0319	0.9847	0.8322	0.9988	0.8160
	Main evaluation			**Secondary**	**evaluation**

Table 5.1: Performance metrics of different models during the main and secondary evaluations; detailing loss, precision, and recall scores

The table reveals that model D achieved higher recall than precision in the main evaluation when class weights (0:1.2, 1:1) were used. These class weights were selected to prioritize precision over recall by assigning greater weights to incorrectly classified negative data. In contrast, model E, evaluated with equal class weights (0:1, 1:1), achieved higher precision than recall, with higher loss. This discrepancy contradicts the intended effect of class weights and the number of epochs used, and is likely attributed to the specific training path taken by the model. As the data is shuffled and the values of all neurons are randomized at the beginning, moderate variations can occur during training, even with identical configurations.

The table also indicates that model C did not outperform model B in the evaluation, despite utilizing twice as much positive training data, where the additional data for model C was selected using the same method as the initial data for model B. This outcome could be a result of the need to increase the number of training epochs due to the increased amount of positive data. Although it was believed that increasing the number of epochs from five to seven for model C adequately addressed

this, it is possible that doubling the number of epochs would have been necessary to account for the data change. Another possibility is that the newly added training data was of lower quality compared to the initial data.

5.1.2 F1 Scores

In general, comparing models based on their F1 scores can provide a more balanced overview of their overall accuracy, as depicted in Figure 5.2. The figure demonstrates the result of a significant drop in recall when using more data and epochs for training, as models C and E achieved much lower scores in one of the evaluations than their counterparts. This diagram is valuable when precision and recall are equally desirable, thus we have included it for readers interested in the overall accuracy of the system. However, for the evaluation of a system like this, we believe that the loss, precision, and recall scores better convey the results.

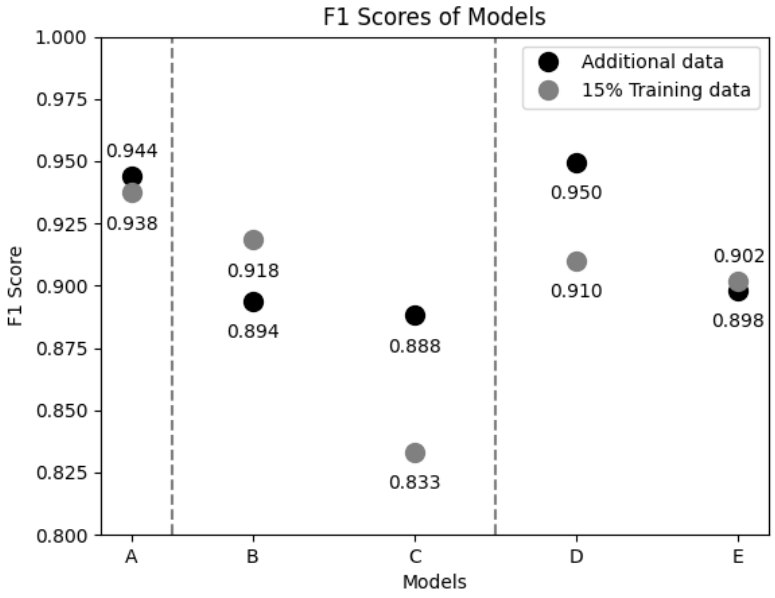


Figure 5.2: F1 scores for 5 different models evaluated with datasets from the training and additional data, making up the main and secondary evaluations, respectively. Note that the models on the same side of the vertical dashed borders made use of identical model structures

Model A was the highest-scoring model among the models that utilized a less advanced training and evaluation setup than what was used for models B through E. Figure 5.2 demonstrates that this model achieved impressive F1 scores in both the main and secondary evaluations, but note that the results for model A are somewhat misleading and will be used for the discussion of the results in the next chapter.

5.2 Android Application

The fundamental basis of the application is a simple interface that enables the user to view their position (if permission was granted) on a Google Map, which also displays information about any potential gunshots that may have occurred.

Upon opening the application, the user is presented with the option to allow the app to use their current location, recording device, and the possibility of receiving notifications. The user has the option to deny any of these permissions, which will limit the functionality of the app. However, to ensure the best user experience, the user can still access important functionalities such as alerts from gunshots, even if some permissions are denied.

In case the user decides to use the app again after denying some permissions, a rationale is displayed to provide additional information on why these permissions are needed.

Figure 5.3: When opening the app

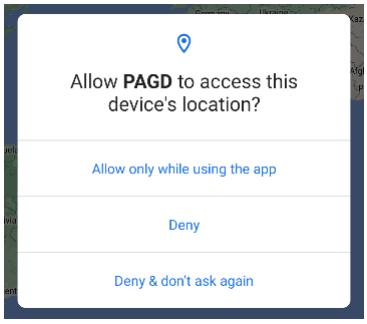


Figure 5.4: When opening the app

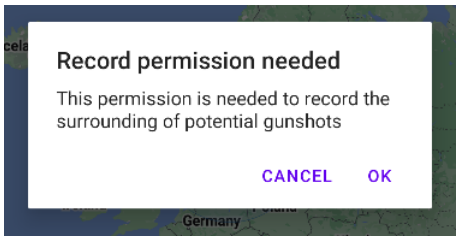


Figure 5.5: If permission is denied

5.3 Gunshot Localisation

The simulation module described in section 4.3 was run 10 000 times for each combination of parameters, and results are shown in table 5.2 and 5.3. Tables show how in how many of the simulations the algorithm converged on a result and statistics on the estimation error when converging on an estimation.

	4 Clients	6 Clients	8 Clients
Converged	82.8%	94.2%	97.8%
Median error	23.7 m	7.88 m	6.33 m
Mean error	70.1 m	17.8 m	8.31 m
Standard deviation of error	118 m	46.7 m	19.0 m
95th percentile error	320 m	39.8 m	15.9 m

Table 5.2: Test results with timestamp error between 0 and 10 ms

	4 Clients	6 Clients	8 Clients
Converged	82.4%	94.4%	98.2%
Median error	42.2 m	18.7 m	14.1 m
Mean error	84.6 m	28.8 m	16.9 m
Standard deviation of error	115 m	45.8 m	19.3 m
95th percentile error	319 m	71.6 m	33.7 m

Table 5.3: Test results with timestamp error between 0 - 100 ms

5.4 Results of the Evaluated PAGD System

In the evaluation of the PAGD system, all components worked together without any issues. The gunshots were discovered and reported to the server, which pinged the smartphone to alert that a gunshot was detected. No false reports were generated during the test, which further confirms the results in Section 5.1.1, view Table 5.4 for the results of this test.

Model	**Average prediction on gunshots**
Additional model	0.791
D	0.765
E	0.878

Table 5.4: Results of the DL models during testing on a shooting range

6
Discussion

This section delves into an analytical exploration of the findings and implications arising from the research and development of each individual component of this project. It aims to provide a comprehensive interpretation of the results, pinpoint the encountered limitations, and suggest potential avenues for future research and improvements. The discussion integrates insights pertaining to both the AI model and the application's architecture, their functionalities, and the computational resources necessitated by their operation, in addition to the ethical considerations associated with the application's deployment. This section serves as a platform for a detailed examination of the multifaceted aspects of the PAGD application, scrutinising the practical implications of the technology while critically evaluating the societal, legal, and ethical contexts within which it functions.

6.1 Deep Learning Model

Signal processing, dataset creation, and model training and evaluation, all played major roles in the final results of the deep learning models that were expressed in Section 5.1. Many changes were done throughout the evaluation process to all of these areas to address any issues found with the deep learning models before we obtained reliable and favourable results.

6.1.1 The Process of Machine Learning

Initially, for model A we used a low resolution for the spectrograms and included all data from the positive dataset. Despite the impressive results obtained from the evaluation, during live testing it was noticed that the model reacted positively to almost any input. For a description of the live testing procedures, see Section 4.2.5. It was discovered that instead of learning the features of gunshots, the model learned to give an optimal constant value for all predictions, which minimised loss, and maximised recall and precision.

After this realisation, we tried a variation of methods to try to improve the results. The first attempt was to increase the complexity of the model, but this gave worse accuracy in the evaluations, many times giving recall and precision of 0. Next, we turned our attention to the data and made use of band-pass filtering and data augmentation that decreased the noise in the training data, and varied the position of gunshots in the spectrograms. This had little effect on the evaluation of models with the same structure as model A. While both of these methods should

have increased the accuracy of the model, and certainly made a positive impact on future models, it did not address the core issue.

Initially, a thorough analysis of the positive data used for training had not been conducted, as we relied on the negative data to compensate for any potential negative impact from the presence of subpar data in the positive dataset. Moreover, we recognised that the resolution of the spectrograms, set at 64 by 65 pixels, might have been insufficient to capture the complex features of the gunshots. Upon realising these limitations, we performed a manual audit on the positive dataset and increased the resolution of the spectrograms by approximately four times the original amount. The increase in resolution had a noticeable effect on the efficiency of the model, taking longer to process each spectrogram, which has a negative effect on the resource utilisation of the application. However, with these changes in place we noticed a drastically improved correlation between the results from the evaluation, and the results observed during live testing. This shows that the evaluation process is every bit as important as choosing a good deep learning model, as well as the importance of auditing the data that is used for training.

6.1.2 The Resulting Deep Learning Models

Out of the five DL models presented, the model E best matched our expectations and requirements, achieving the highest precision during all the evaluations steps and giving good results on the shooting range, review Sections 5.1.1 and 5.4. However, there are still improvements that could be made.

First off, the recall could be significantly improved by including more high quality data in the training. This could be achieved by recording a large amount of gunshots specifically for the intended implementation. This was done by E. Grane and L. Bokelund in [20], studying gunshot detection performed on portable devices where they collected more than 3000 samples for their training data that significantly increased the performance of the model. With a relatively simple DL model structure, they achieved F1 scores ranging from 0.931 to 0.952 with the use of the new data, which was a significant improvement over the F1 score of approximately 0.43 for the model solely being trained on data collected from free online sources. It would also be beneficial if the newly collected data was gathered from a larger number of weapon types in order to make the model generalise better to new types of firearms.

Second, during the live testing where the models were tested on a variety of negative data, some false positives were discovered. Specifically, the sounds produced by hand clapping and a pen dropping on a table caused the model to output large positive predictions. It was difficult to make the hand clapping sound right for the model to give high values, which shows that the model did recognise to some degree that hand claps were not gunshots, but when performed in a certain way the model made predictions up to 85% confidence. The pen drop gave more consistent results, causing the model to make a false prediction with a maximum value of 92% confidence. It should be noted that if the threshold for predictions were set above 92%, then none of these sounds would be reported as gunshots. The negative dataset could be extended with more data to prevent false classifications that are discovered

during the evaluation phase, but this shows the complexity of the problem at hand.

The objective of a system like this extends beyond accurately detecting gunshots; it also involves reliably identifying non-gunshot sounds. Achieving this goal entails incorporating a wide range of potential sounds that the system may encounter in real-life scenarios during both training and evaluation. Since this system is designed to be used on smartphones by the general public, the sheer magnitude of scenarios that needs to be taken into account becomes enormous. Consequently, addressing this challenge presents a difficult problem with a complicated solution, as any oversight in accounting for audio types that could trigger false positives could significantly impact the system's reliability.

The solution to this issue might consist of a more complex deep learning model used, trained by experienced professionals making use of more extensive datasets. While developing a more complex deep learning model is one aspect of the solution, it is also worth to point out that resolving the issue may extend beyond the realm of machine learning. Incorporating statistical methods may play a pivotal role in achieving a comprehensive solution that minimises false reporting. One aspect of a statistical solution will be explored further in the next section, discussing a statistical approach to solve false reporting in regard to the base-rate problem.

6.2 Base-rate Problem

As with any crime detection system, the amount of criminal activity tends to be relatively small compared to the amount of tested data. This is not unique to this application, yet this certainly poses a problem to the efficiency of the system by the cheer volume of false reports that could be generated if it is not properly handled. This problem is sometimes referred to as the base-rate fallacy problem [21, pp. 280-281]. To illustrate how it affects the application, the following example is provided:

It was reported in the year of 2022 that 391 shootings took place in Sweden; what will happen if a single user is assumed to have the application and be present at every incident of these gunshots? For simplicity, it is assumed that during every shooting, only one gunshot was recorded. With the current configuration of the application, the user would process recorded audio every other second, which would amount to 15 768 000 spectrograms being processed in a year that has to be correctly identified. Let us also assume that the accuracy of the gunshot identification process is highly accurate with 99.9% accuracy.

A = 99.9% : Accuracy
G = 391/15768000 = 0.00248% : Spectrograms containing a gunshot
N = 99.99752% : Spectrograms not containing a gunshot
R : Report indicates gunshot

Using Baye's Theorem the statistical probability of a report being a false positive, as well as how many reports in the given example are expected to be false and correct can be calculated:

Probability of a report being false **P[N|R]** = P[R|N]P[N]/P[R] =
P[R|N]P[N]/(P[R|G]P[G] + P[R|N]P[N]) = (1 - A)*N/(A*G + (1 - A)*N) =
(1 - 0.999)*0.9999752/(0.999*0.0000248 + (1 - 0.999)*0.9999752) = **97.5823%**
Reports = (0.999 * 0.0000248 + (1-0.999) * 0.9999752) * 15678000 = **16066**
False reports = 0.975823 * 16066 = **15678**
Correct reports = (1 - 0.975823) * 16066 = **388**

As seen from this example the application almost detects every gunshot, however, despite the high accuracy, it can still be expected that 15678 false reports will be made by one person using the application. In reality most users are likely to not experience a shooting in a year which means the number of false reports will be even higher, and in here lies the problem. With so many false reports, the moderation of the system will be infeasible. To rectify the situation, a set of variables that can influence the number of false positives have been identified:

- **Number of devices reporting a gunshot:** Impacts the probability of a false report being taken seriously
- **Geographical distance between reports:** Limits the amount of devices that together can produce false reports
- **Difference in time between reports:** Tighter constraints limits the possible amount of spectrograms to be processed in a given time-frame
- **Estimated population density:** Gives the ability to calculate the probability of a false report being generated and adjusting the allowed geographical distance and time between reports as necessary
- **Confidence threshold for reporting:** Affects the accuracy of the system

An example of how the previous example can be improved on with these variables is shown next, where N = Population density * radius$^2 * \pi$ is the number of devices within the radius that a gunshot can be detected and Nt is the number of devices required to report a gunshot for it to be considered valid. Using the binomial theorem, the possibility of a false report being considered valid at a given time by N devices located in the same radius can be calculated with the following formula: $N!/((N - Nt)! * Nt!) * (1 - A)^{Nt} * A^{(N-Nt)}$.

Assume that at a given time a group of 1000 devices with the application installed are located within a defined radius and that no gunshot takes place during this time, without any restriction put on considering the validity of reports it can be expected that the group will generate at least 1 false report: $(1 - A) * 1000 = 0.001 * 1000$. When the condition that Nt = 5 is applied, it can be expected that approximately 0.003 false reports will be made: $1000!/((1000-5)!*5!)*(1-0.999)^5*0.999^{(1000-5)} = 0.00305$. This has effectively reduced the probability of a false report to be marked as valid by 99.695%.

These simplified examples underline the problem and solution to non-malicious false reporting, and in conclusion shows that the reliability of the system can be made adequately high with statistics on average population density of devices with the application installed, which could very easily be obtained with more or less

privacy concerns depending on the method used, and thorough enough constraints. Furthermore, this statistical approach might also be part of a solution to decrease false reporting in general.

6.3 Analysis of Battery Drain by the Application

The application's influence on the battery lifespan of a device is a critical consideration for its overall performance. To evaluate this, the application was tested under idle conditions for several hours on a Samsung Galaxy S20, during which it constantly monitored ambient sounds for potential gunshots. The observed battery consumption data included:

- 4 hours - 8.4% battery usage
- 3 hours 41 minutes - 7.6% battery usage
- 2 hours 49 minutes - 6.9% battery usage

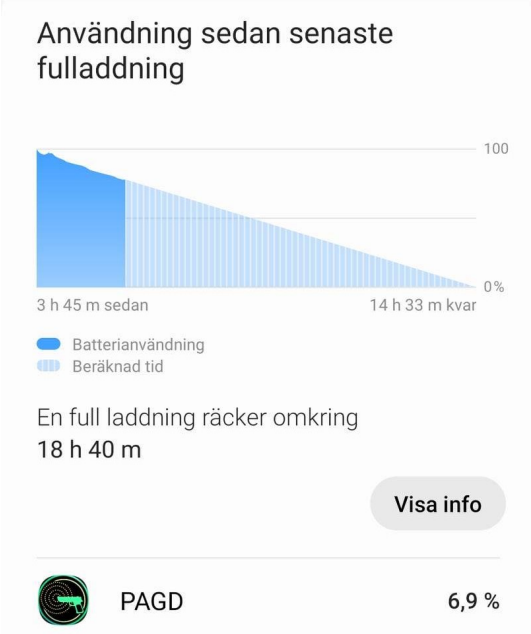


Figure 6.1: A graph depicting the approximated battery time left while using the app

An analysis of this data indicates an average battery consumption rate of 2.21% per hour. Given the 24-hour operational design of the application, this translates into an estimated daily battery usage of approximately 53.04%.

Despite this significant consumption, which utilises over half of the device's battery life, the application's continuous operation remains feasible. Taking into account that a typical adult remains awake for approximately 15 to 17 hours per day, as per the recommended sleep duration of 7 to 9 hours [22], the device can be expected to operate without charging during these awake periods. Consequently, this lowers the battery usage to a more sustainable range of 33.15% - 37.57%, predicated on the assumption that the device is recharged during sleeping hours.

However, it is important to note that these figures are derived under ideal conditions, not accounting for the power consumption of other applications that the user might be operating concurrently on the device.

Addressing these key aspects can significantly improve the application's functionality and overall user experience.

6.4 Gunshot Localisation

Results of the evaluation of the gunshot localisation systems shows the potential of a distributed gunshot detector system as implemented. The results show a median error of 7.88 meters when timestamp error is less than 10 milliseconds and 6 clients report the gunshot. This is comparable to the 1999 study of the widely implemented Shotspotter system shown in National Institute of Justice's research in brief [23] where their median error is 25 feet or 7.6 meters when .38 calibre pistols were fired. However, the results of this study is limited by the fact that the evaluations only consisted of simulations of a component of the system.

What must be taken into account is the fact that the simulations do not reflect the impact of false reports and conflicting reports (two gunshot firings close in time and space). Since our AI model has been shown to be prone to giving false positives, and the fact that our sound localisation system has no method of eliminating false reports indicate that this may be a big issue not reflected in results.

Another problem is that, as indicated in the results, at least six clients need to report the gunshot to get a reasonable estimate of position. But as shown in field trials, the AI model often fails to detect gunshot when more than a few hundred meters away. This then requires the detectors to be densely distributed, which is harder to implement and further increases the risk of false reports impairing results.

6.5 Privacy and Ethics

In a system handling sensitive personal information, it is of paramount importance to consider the ethical concerns of the goal with the system and the privacy concerns that arise from the implementation details.

6.5.1 Privacy

This implementation is affected by the General Data Protection Regulation (GDPR) which is applicable to all countries in EU [24]. The main principles of the GDPR for protection of privacy in regard to personal information use applicable to this application include:

- **Lawfulness, fairness, and transparency** — Processing must be lawful, fair, and transparent to the user. This means that the user has to be notified about what personal information is being collected, how it is used and what choices the user has in regard to the system's use of this information
- **Consent:** The user has to give explicit consent to collection of their personal information. The user should also be able to revoke their consent to collection

of their personal information and all information previously stored and have any previously collected information removed.
- **Purpose limitation** — You must process data for the legitimate purposes specified explicitly to the data subject when you collected it.
- **Data minimisation** — You should collect and process only as much data as absolutely necessary for the purposes specified.
- **Accuracy** — You must keep personal data accurate and up to date.
- **Storage limitation** — You may only store personally identifying data for as long as necessary for the specified purpose.
- **Integrity and confidentiality** — Processing must be done in such a way as to ensure appropriate security, integrity, and confidentiality (e.g. by using encryption).
- **Accountability** — The data controller is responsible for being able to demonstrate GDPR compliance with all of these principles.

For an application of a sensitive nature such as this, it is crucial to proceed with caution and maintain transparency with the end-user. Therefore, it is imperative to provide a comprehensive description of the data being collected and how it will be used. The implementation of this system aims to capture the minimum necessary information about the user's GPS coordinates and timestamp only in the event of a gunshot being detected. This approach ensures that the description remains concise and relevant to the user's needs, without making unwarranted use of their personal data. This will in turn ensure that the user can make an informed decision regarding the use of their personal information.

It is also important to maintain accuracy in the system's capability to classify gunshots, as this will help to avoid unnecessarily obtaining personal data from users in the case of false positives. In addition, it is important to delete any data that has been left unused for a prolonged period of time. In order to achieve this, the timestamp of each report saved in the database has to be evaluated to ensure that it has been validated by other reports within a reasonable time-frame. In the case that a report can not be validated, it has to be removed from the system.

It is also important for legal reasons, as well as the functioning of the system, to verify the accuracy of the reports. A principle that was followed when creating this system is that collected user-data should not reveal the user's identity. However, falsified data such as falsified GPS coordinates or timestamps could disrupt the system's operations, rendering it unusable. Therefore, user authentication is essential to counter any repudiation issues.

In summary, by adhering to the guidelines mentioned above, the system should comply with all GDPR requirements, ensuring transparency and maintaining the end-users' privacy.

6.5.2 Ethics

The ethical implications of an application that persistently records ambient audio to detect gunshots are multifaceted and require careful examination. Foremost among these is the potential invasion of privacy, as the audio recording could unintentionally capture private conversations or sensitive data. It is critical that these potential

privacy breaches are mitigated through robust data handling policies and practices. Informed consent is a vital component of this process. Users must be thoroughly briefed on the continuous audio recording nature of the application, and their explicit consent should be procured prior to any recording activity. The ability to revoke this consent should be available to the users at any time. Beyond this, the security of the data is of utmost importance. There should be measures in place to protect the collected data against unauthorised access or leaks that could jeopardise users' privacy. Furthermore, the AI system might misinterpret other loud sounds as gunshots, leading to false positives. Such a possibility necessitates a rigorous evaluation of the system's accuracy and reliability.

Legal implications also demand attention. As laws around recording audio without explicit consent vary across jurisdictions, it is imperative to ensure the application's operation complies with all relevant laws and regulations. Furthermore, the potential psychological impact on users, who may experience increased anxiety or fear knowing an application is continuously monitoring for gunshots, should be considered.

The application's role in the event of a detected gunshot presents another significant ethical aspect. Determining who should take responsibility for the subsequent response, whether it is the user, the application developers, or law enforcement, is a complex task. This responsibility allocation requires careful deliberation.

The application also needs to be sensitive to issues of bias and discrimination, particularly if its usage is concentrated in areas with higher crime rates. Ensuring transparency in how the application functions, what data it collects, how that data is used and stored, and who can access it, is crucial to maintaining user trust.

6.6 Further Work

The project has yielded a moderately successful prototype of the Privacy Aware Gunshot Detection (PAGD) system. However, it is important to acknowledge that with additional time and resources, multiple aspects of this system could see substantial improvements.

Immediate future steps could involve large-scale deployment of the application for more comprehensive testing. This would not only allow for fine-tuning of the algorithms and optimisation of parameters, but would also provide a much larger dataset to enhance the accuracy and reliability of gunshot detection.

A considerable amount of security implementation is also required. Protecting user data is of utmost importance, given the application's continuous audio recording capabilities. Measures must be taken to ensure secure data encryption, transmission, storage, and stringent access controls.

A possible improvement to the gunshot localisation system would be to utilise the method of cross correlation. Cross correlation is a technique to determine how similar two signals are, at a specified time offset from one another. Given that a report from a receiver would also contain a couple seconds of audio of the gunshot, one could then also determine a more accurate timestamp of the gunshot. As seen in the results, a more accurate timestamp gives much more accurate estimations. Furthermore, one could potentially also use cross correlation to determine if the two

reports concern the same gunshot to reduce the risk of false or conflicting reports.

Another possible improvement would be if on the application side additional information could be estimated from the sound received, such as distance to gunshot or what type of weapon was used. This could possibly allow for better correlation of reports and lesser clients needed to get a good estimate.

An additional crucial area that merits further investigation is the system's battery consumption. While the current usage rates are feasible for continuous operation, further optimisation could significantly enhance the application's usability and acceptance amongst users.

The potential for false positives is another area for development. Minimising false reports is critical for maintaining user trust and ensuring appropriate resource allocation, making it essential to improve the AI model's ability to distinguish gunshot sounds from other loud noises.

In essence, while the current system serves as a promising foundation, its evolution into a fully optimised, secure, ethically sound, and broadly applicable tool will necessitate continued research, refinement, and rigorous testing.

7
Conclusion

Based on our findings, the development of a privacy-aware gunshot detection system on smartphones appears feasible, offering a distributed solution for crime detection and prevention. Although the prototype developed in this project has scope for reliability improvements, it performed within the expectations considering the limited resources available. Furthermore, the Privacy-Aware Gunshot Detection (PAGD) system demonstrates substantial potential for application in law enforcement and military contexts. For example, the technology could enable real-time alerts of gunfire incidents, thereby enhancing response times and potentially saving lives. Additionally, integrating the system with citywide surveillance cameras would create a comprehensive detection network, significantly enhancing public safety.

Given the various potential uses of the system, it is imperative to address ethical considerations regarding privacy and responsibility allocation. Transparency regarding data usage and granting users control over their data are paramount. Clear delineation of roles and responsibilities in the event of a detected gunshot is essential. By carefully addressing these ethical considerations, the PAGD system can operate in a responsible and trustworthy manner, maximizing its societal benefits while respecting privacy concerns.

Bibliography

[1] "Final-architecture." [Online]. Available: https://developer.android.com/static/topic/libraries/architecture/images/final-architecture.png (accessed: 2023-04-27).

[2] Gun Violence Archive (GVA), "Gun violence archive," 2023. [Online]. Available: https://www.gunviolencearchive.org/ (accessed: 2023-02-24).

[3] J. Hansen and H. Boril, "Gunshot Detection Systems: Methods, Challenges, and Can they be Trusted?," in *Convention Paper 10540*, Las Vegas, USA, 2021. [Online]. Available: https://www.researchgate.net/publication/361313583_Gunshot_Detection_Systems_Methods_Challenges_and_Can_they_be_Trusted, Accessed: 2023-05-15.

[4] N. G. La Vigne et al., "Implementing Gunshot Detection Technology," Urban Institute, Washington, USA, 2019. [Online]. Available: https://www.urban.org/sites/default/files/publication/101161/implementing_gunshot_detection_technology_recommendations_for_law_enforcement_and_municipal_partners.pdf, Accessed: 2023-05-15.

[5] State v. Carter, "STATE OF OHIO Plaintiff-Appellee v. CHRISTOPHER CARTER Defendant-Appellant," United States Reports, Court of Appeals of Ohio, Montgomery, USA, N.E.3d 611, 183, 2022. [Online]. Available: https://casetext.com/case/state-v-carter-12152198, Accessed: 2023-02-09.

[6] D. Welsh, N. Roy, "Smartphone-based mobile gunshot detection," in *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. Kona, HI, USA, 2017. [Online]. Available: https://ieeexplore.ieee.org/document/7917566, Accessed: 2023-02-08.

[7] B. P. Lathi, *Signal Processing And Linear Systems: The Sampling Theorem*, international ed., New York, USA: Oxford University Press, 2010.

[8] "Engineering toolbox, solids and metals - speed of sound," 2004. [Online]. Available: https://www.engineeringtoolbox.com/sound-speed-solids-d_713.html, Accessed: 2023-04-27.

[9] R. Kaune, "Accuracy studies for tdoa and toa localization," in *2012 15th International Conference on Information Fusion*, pp. 408–415, 2012.

[10] Robert C. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design*, Prentice Hall, 2017.

[11] R. C. Maher, "Acoustical characterization of gunshots," in *2007 IEEE Workshop on Signal Processing Applications for Public Security and Forensics*, pp. 1–5, 2007. [Online]. Available: https://ieeexplore.ieee.org/document/4218954?arnumber=4218954 (accessed: 2023-04-18).

[12] The Pallets Projects, “Welcome to Flask — Flask Documentation (2.2.x),” 2010. [Online]. Available: https://flask.palletsprojects.com (accessed 2023-02-01).
[13] Internet Security Research Group (ISRG), “Let’s Encrypt,” 2023. [Online]. Available: https://letsencrypt.org (accessed 2023-05-15).
[14] Lokesh Gupta, “What is REST,” 2023. [Online]. Available: https://restfulapi.net (accessed 2023-02-20).
[15] MDN Web Docs, “HTTP response status codes,” 2023. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status (accessed 2023-05-15).
[16] MariaDB Foundation, “MariaDB Server: The open source relational database,” 2023. [Online]. Available: https://mariadb.org (accessed 2023-02-20).
[17] S. Singer and J. Nelder, “Nelder-mead algorithm,” *Scholarpedia*, vol. 4, no. 7, p. 2928, 2009.
[18] R. Lilien, "Development of Computational Methods for the Audio Analysis of Gunshots," NCJRS, Report No. 2016-DN-BX-0183, National Institute of Justice, Washington, DC, USA, 2018.
[19] U. S. government, “Gps standard positioning service (sps) performance standard,” 2020.
[20] E. Grane, L. Bokelund, "Gunshot Detection from Audio Streams in Portable Devices," master thesis, Department of Electrical and Information Technology, Lund University, Lund, Sweden, 2022. [Online]. Available: https://lup.lub.lu.se/luur/download?func=downloadFile&recordOId=9090317&fileOId=9090321.
[21] W. Stallings and L. Brown, *Computer Security Principles and Practice: The Base-Rate Fallacy*, 4th ed., New York, USA: Pearson Education Limited, 2018.
[22] “National sleep foundation’s sleep time duration recommendations: methodology and results summary.” [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352721815000157 (accessed: 2023-05-14).
[23] L. G. Mazerolle, C. Watkins, D. Rogan, and J. Frank, “Random gunfire problems and gunshot detection systems,” *National Institute of Justice - research in brief*, 1999.
[24] Swedish Authority for Privacy Protection, "The GDPR fundamental principles," 2021. [Online]. Available: https://www.imy.se/en/organisations/data-protection/this-applies-accordning-to-gdpr/the-gdpr-fundamental-principles/ (accessed: 2023-04-27).

A

Appendix 1

Deep Learning Model Evaluation

Project name: Privacy-aware Gunshot Detection
Date: 2023-04-20 - 2023-05-13

Author:
Martin Engström

Evaluated models:

MODEL A

Layer (type)	Output Shape	Param #
(Conv2D)	(60, 61, 16)	416
(Batch Normalization)	(60, 61, 16)	64
(Activation)	(60, 61, 16)	0
(Conv2D)	(58, 59, 32)	4640
(Batch Normalization)	(58, 59, 32)	128
(Activation)	(58, 59, 32)	0
(Flatten)	(109504)	0
(Dense)	(128)	14016640
(Batch Normalization)	(128)	512
(Activation)	(128)	0
(Dropout 0.1)	(128)	0
(Dense)	(1)	129

Total params: 14,022,529
Trainable params: 14,022,177
Non-trainable params: 352

MODELS B & C

Layer (type)	Output Shape	Param #
(Conv2D)	(60, 125, 64)	1664
(Activation)	(60, 125, 64)	0
(Conv2D)	(58, 123, 32)	18464

(Activation)	(58, 123, 32)	0
(Flatten)	(228288)	0
(Dense)	(128)	29220992
(Batch Normalization)	(128)	512
(Activation)	(128)	0
(Dropout 0.5)	(128)	0
(Dense)	(1)	129

Total params: 29,241,761
Trainable params: 29,241,505
Non-trainable params: 256

MODELS D & E

Layer (type)	Output Shape	Param #
(Conv2D)	(64, 129, 64)	640
(Activation)	(64, 129, 64)	0
(MaxPooling2D	(32, 64, 64)	0
(Conv2D)	(32, 64, 128)	73856
(Activation)	(32, 64, 128)	0
(MaxPooling2D)	(16, 32, 128)	0
(Conv2D)	(16, 32, 256)	295168
(Activation)	(16, 32, 256)	0
(GlobalAveragePooling2D)	(256)	0
(Dense)	(256)	65792
(Batch Normalization)	(256)	1024
(Activation)	(256)	0
(Dropout 0.5)	(256)	0
(Dense)	(1)	257

Total params: 436,737
Trainable params: 436,225
Non-trainable params: 512

Description:

For all of the models, the metrics precision, recall, and loss scores were used. Binary cross-entropy was used as the loss function and the positive data used was collected from the Gunshot Audio Forensics Dataset.

Model A was trained on the whole dataset with poor quality audio included, while models B through E were trained on manually selected data that contained less noise and generally higher quality recordings. Models B and C had the same architecture but were trained on different amounts of data to see the changes in accuracy based on the size of the datasets. Models D and E also shared architecture and were trained for a different amount of epochs in an attempt to maximize accuracy, see Table 1.

Additional testing was performed on an android emulator on the desktop to verify the reliability of the scores.

Model	Learning rate	Class Weights	Positive portion	Evaluation portion	Batch Size	Positive Data points
A	0.1	(0:3, 1:1)	25%	5%	40	4237
B	0.0001	(0:1.2, 1:1)	3.8%	15%	200	330
C	0.0001	(0:1.2, 1:1)	6.4%	15%	200	689
D	0.00015	(0:1.2, 1:1)	6.3%	15%	150	689
E	0.00013	(0:1, 1:1)	6.3%	15%	150	689

Table 1: Hyperparameters used and information on the datasets used during training

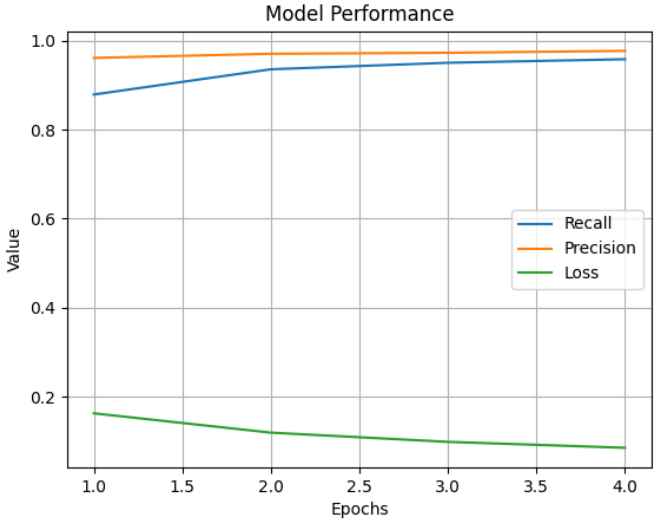
Results:

Model	Loss	Precision	Recall	Precision	Recall
A	0.0908	0.9543	0.9216	0.9736	0.9160
B	0.0278	0.9727	0.8699	0.9819	0.8202
C	0.0465	0.9561	0.7385	0.9834	0.8097
D	0.0273	0.8750	0.9484	0.9781	0.9228
E	0.0319	0.9847	0.8322	0.9988	0.8160
	Evaluation 1			**Evaluation 2**	

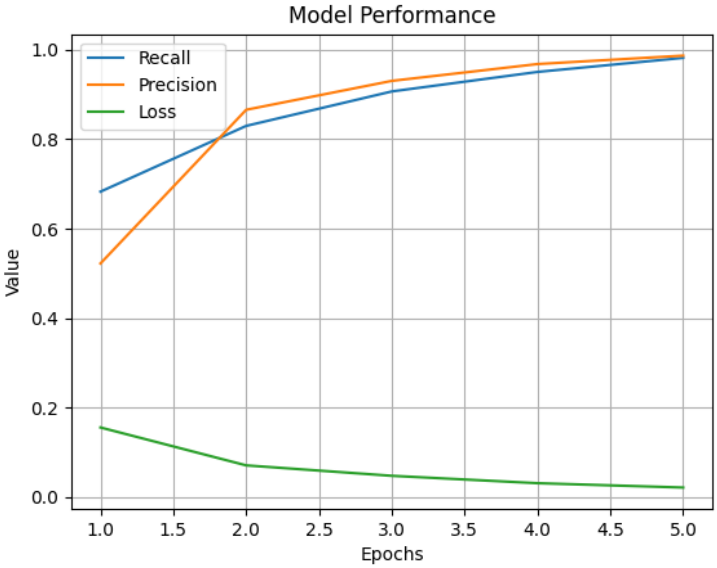
Table 2: Loss, precision, and recall scores from two evaluations for each model

Diagrams showing the evaluated performance of all models during each epoch:

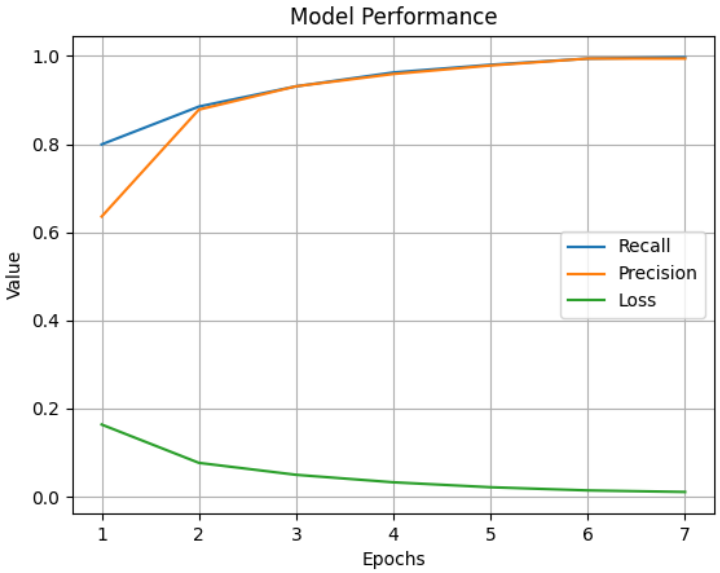
Model A



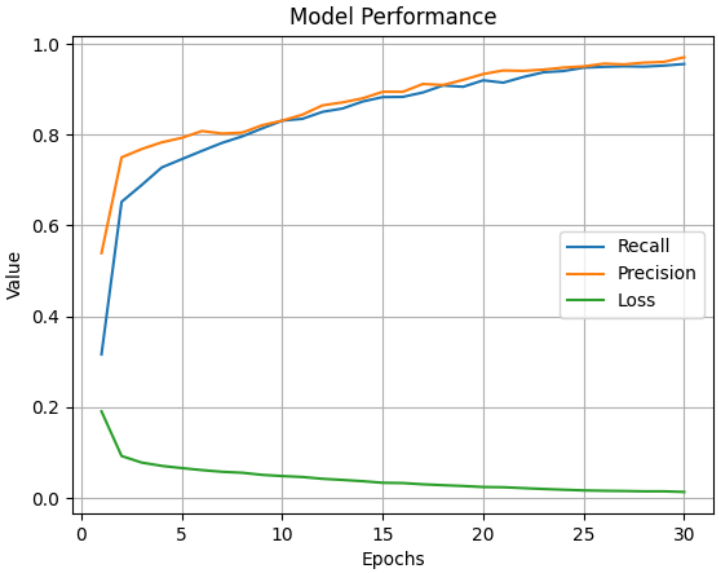
Model B



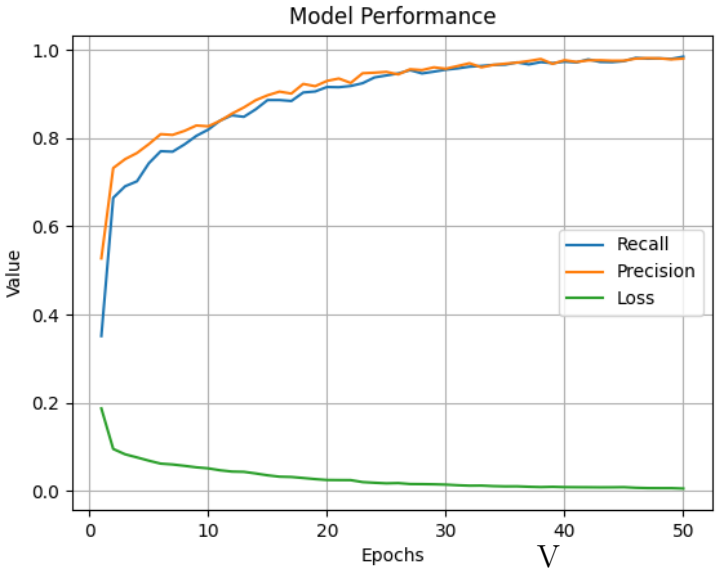
Model C



Model D



Model E



Analysis:

Model A achieved a good score on the evaluation, but gave false positives on mostly anything during live testing on a smartphone and desktop, making the scores for model A unreliable, see Table 2.

After many changes, the problem that affected previous models such as model A was detected to lie in the poor quality of some of the positive data used during training. For models B through E, better data was selected and a more rigorous evaluation process was performed. Although the amount of positive data used for training these models was significantly lower, these models showed evidence that the scores were accurate during live testing and that they had generalized well to new data.

It was interesting how including more than twice as much data between models B and C actually lowered the evaluation scores on the first evaluation, and increased loss. It is possible that this is just a coincidence, however further investigation would be required to find the root cause.

The models D and E only have slightly different hyperparameters, with the largest change being that model D was trained for 30 epochs, and model E was trained for 50 epochs. It makes sense that precision has significantly increased after more training, on the expense of recall. This is a positive development, since precision is most important for reliable positive predictions.