Siren Detection and Recognition for Hearing-Impaired Individuals: A Machine Learning Approach

UNIVERSITY OF LINCOLN

Samuel Harwood
25790959

25790959@students.lincoln.ac.uk

School of Computer Science

College of Science

University of Lincoln

Submitted in partial fulfilment of the requirements for the
Degree of BSc(Hons) Computer Science

Supervisor: Mr. James Wingate

May 2024

Acknowledgements

I would like to thank James Wingate, my project supervisor for his support during weekly meetings and especially for dealing with my repeated out of hours questions. I would also like to thank Connor Graves for his assistance in recording data for this project.

Abstract

Since their adoption in the 1970s, sirens have been used by the emergency services to warn other road users of their immediate need for the right of way. Nevertheless, it is far too common that unaware drivers miss the warning signals, and cause delays for the emergency services. The Royal National Institute for Deaf People stated that in the UK alone, “12 million adults are either deaf, have hearing loss or tininitus” (RNID, 2023); therefore, it is no wonder that a siren system is often ineffective at gaining drivers’ attention, especially at busy junctions. This project proposes a solution to this issue, using machine learning to detect siren sounds amongst busy urban streets.

Table of Contents

List of Figures

List of Tables

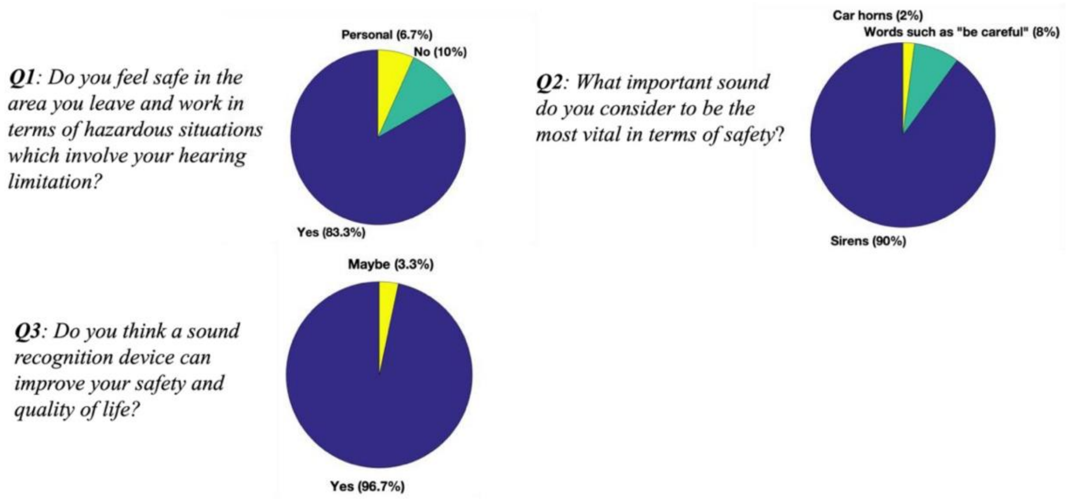
Chapter 1

Introduction

1.1 Motivation

The highway code dictates that in emergency situations, “it is critical to ensure there is suitable access for emergency services to provide a swift and effective response at all times” (The Stationery Office, 1931, 219-255). Other road users are expected to respond appropriately and yield possession of the road so that an emergency service vehicle (ESV) can quickly and safely reach the site of the emergency. To illustrate the immediate need for priority use of the road, ESVs activate blue lights on the vehicle in conjunction with a siren system. A lights and siren (L&S) system provides road users with ample warning of an incoming ESV. Notably, research conducted by (Brown et al., 2000), in Syracuse, New York (pop. 170,000 at time of research) revealed that using a L&S system reduced ambulance response times by an average of 1 minute, 46 seconds. Any road user with a hearing impairment (HI) cannot rely on a siren system to proactively stay clear, and therefore their lack of action could increase the response time of ESVs (Harwood, 2024).

Despite numerous innovations in vehicle safety technologies over the last two decades, there has been little to no evolution benefiting those with HIs. To gauge the importance of this issue, (Hersh et al., 2010) conducted a survey on the possible effects of creating a system which automated the detection of a siren. Three questions were asked to 30 respondents, each with a diagnosed hearing disability. The questions and responses are visualised in the figure below.


Q1: Do you feel safe in the area you leave and work in terms of hazardous situations which involve your hearing limitation?

Q2: What important sound do you consider to be the most vital in terms of safety?

Q3: Do you think a sound recognition device can improve your safety and quality of life?

Figure 1: Hearing-impaired people survey results (Ramirez et al., 2022).

Evidently, there is a strong moral case for the development of an effective siren detection and recognition system to aid hearing impaired drivers (Harwood, 2024).

1.2 Document structure

This report encompasses the entire software development lifecycle of the artefact created for this project, which begins with a comprehensive literature review justifying development. Next, a requirement analysis sets up the primary aims and objectives of this project and defines the scope. This will also include a risk analysis, followed by the methodology for software development and the reasoning for specific machine learning models will be touched on in design and methodology. This will be followed by implementation, which showcases important developments which took place to create a final artefact. To conclude this report, a results and discussion page will present all findings and a final conclusion will provide a summary of completed work, the artefacts limitations, and the scope for future work.

The scope of this project has been refined from the original project proposal in order to achieve the primary aim of the software artefact most efficiently: enhance the awareness and safety of hearing-impaired drivers with a machine learning model to detect ESV siren sounds.

Chapter 2

Literature Review

The primary objectives of an ESV siren system are to: gain attention of the public, identify their presence and obtain a proper reaction by projecting size, distance, speed, and direction of their travel. Sirens are therefore integral to efficient negotiation of traffic and reduction of collision risk (De Lorenzo and Eilers, 1991). Drivers are expected to yield the right of way and safely clear a path for an incoming emergency vehicle. A ESVs light and siren (L&S) system means drivers can anticipate its arrival and pre-emptively move, saving crucial time.

A report by (Murray and Kue, 2017) aggregated multiple published results of response (responding to an emergency) and transport times (such as transporting a patient to a hospital) of ESVs in urban and rural environments.

Citation	**Response vs Transport**	**Environment**	**Mean Non-L&S (sec)**	**Mean L&S (sec)**	**Time Saved (sec)**
(Hunt et al., 1995)	Transport	Urban	406	362	43.5 (10.7%)
(Marques-Baptista et al., 2010)	Transport	Urban	1,026	870	157 (15.3%)
(Ho and Casey, 1998)	Response	Urban	449	268	181 (38.5%)
(Ho and Lindquist, 2001)	Response	Rural	728	511	363 (30.9%)
(Brown et al., 2000)	Response	Urban	399	293	106 (26.5%)

Table 1: Emergency vehicle siren and light response time (Murray and Kue 2017, p. 211).

The results in Table 1 demonstrate significant time savings associated with the implementation of L&S systems. Of course, a hearing impaired driver may not perceive the auditory cues of a siren, thereby contributing to delays. In fact, research by (Picard et al., 2008) found that hearing impaired drivers with a hearing loss ranging from 16 to 30dB had a 1.06 PR (prevalence ratio) of a traffic accident, which increased to 1.31 when exceeding 50dB of hearing loss. (Picard et al., 2008) also found that hearing impaired drivers more often failed to follow the highway code. Although not directly relating to ESVs, the results of the study highlight the importance of auditory cues whilst driving.

A 1977 US Department of Transportation (DOT) report showed that a siren's average signal attenuation through a closed-window automobile resulted in a "maximal siren effective distance of siren penetration of just 8 to 12 metres at urban intersections" (Potter et al., 1977). "Considering that there has been significant improvements to sound suppression in modern vehicles, it is no wonder that autonomous cars [and their passengers] are largely deaf [to the outside world]" (Marchegiani and Newman, 2022). If we also factor in a hearing impaired driver's inherent reduced sensitivity to sound, it is obvious there is a need for a more effective method of warning these drivers of an incoming emergency vehicle.

2.1 Hearing aids

The most immediately apparent and plausible solution to this issue is a hearing aid. However, a survey by (Ikeda and Minami, 2020) on the recognition of ESVs by hearing-impaired drivers found that drivers using hearing aids, a conventional solution to hearing loss, often had headaches due to the loud noises of buses and trucks, and that excessive noise is generated further from adverse weather. Nevertheless, many audiologists are adamant that individuals with prescription hearing aids should wear them whilst driving as an essential tenant of safety; despite that in the UK, a HI is not considered a medical disability, such as poor eyesight, and does not need reporting to the Driver and Vehicle Licensing Agency (Government Digital Service, 2012).

Blogs and websites for medical centres such as (Harrison, 2023; Lis, 2024; Muñiz, 2020) overwhelmingly agree that drivers with prescription hearing aids should always wear them whilst driving even if it is not legally required. In spite of this, not many drivers, especially older adults follow this advice; (Thorslund et al., 2012) found in a Swedish study that hearing aids were only used 57% of the time while driving, as compared to 95% for spectacles.

The reasoning behind this can be explained by how hearing is incorrectly perceived as an irrelevant factor for awareness and safety. (Thorslund et al., 2012) also found through a questionnaire, the effect of hearing on a driver's abilities was not considered an important factor for safety by those with profound hearing loss (and in some cases) with severe hearing loss. Whereas, hearing was rated highly as an important factor by drivers with normal hearing and mild hearing loss. An explanation to this could stem from high degrees of hearing loss forcing a form of driving-adaptation, a strategy to reduce the time in a vehicle, the distance driven and using quieter roads at slower speeds. (Julien et al., 2010) define this as "tactical compensation," a coping strategy among older drivers proven to support mobility and reduce accident risk.

There are around 4.7 million individuals in the UK alone who could benefit from use of a hearing aid (British Academy of Audiology, 2023) yet for varying reasons do not wear one, let alone when driving. It is therefore apparent that a solution harnessing hearing aid technology is simply not viable.

2.2 Machine learning approach

Due to the innate complexity of sirens, many scholarly papers look to harness algorithms capable of comprehending constant frequency shifts whilst also being flexible in time (Schröder et al., 2013). Consequently, convolutional neural networks (CNN) are often favoured. CNNs are a class of artificial neural network (ANN) composed of layers which act as building blocks for the neural network. "CNNs are designed to automatically and adaptively learn spatial hierarchies of features" (Yamashita et al., 2018), and do so with a structure inspired by neurons in human and animal brains (Hubel and Wiesel, 1962). These solutions, such as the CNN for recognising ambulance siren sounds by (Ramirez et al., 2022) prove that a machine learning approach is not only viable, but in this case accurate 98% of the time with synthetic testing.

In these scholarly papers, real-time testing often leads to significantly reduced accuracy when compared with synthetic testing, due to external factors like adverse weather and ambient noise from roads and vehicles. To avoid this pitfall and others like it, significant effort will go into reviewing previous works, analysing their shortcomings, and identifying opportunities to enhance this project's final artefact.

2.3 Previous works

Many scholarly papers built around creating new methods of detecting siren sounds do not go beyond creating a machine learning (ML) model, and testing with synthetic data. The few that tried to create a useable product from the theory, often looked to wearable assistant systems, such as this wrist-watch style device from (Chin et al., 2023):

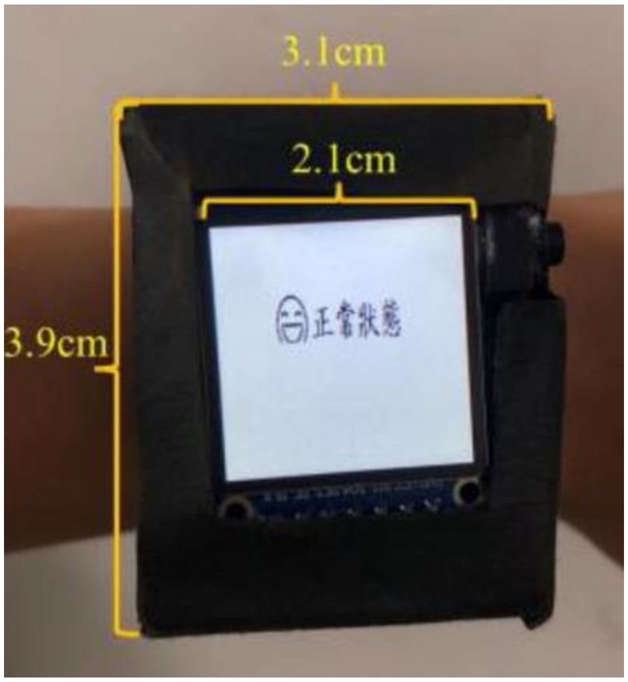


Figure 2: Wearable assistant device (Chin et al., 2023).

This wearable system vibrates on recognition of a car horn, siren, or ambulance siren, which is an intelligent way of alerting the wearer of the device without the need for sound. A problem with a hardware solution such as this is that it requires a user to carry a bulky system for a single use case. This project intends to improve upon this flaw by implementing a ML model directly into a multi-use device which millions of drivers already carry and use every day, a smartphone. Phones often are used in cradles within vehicles as a satnav system or entertainments system alternative. Because smartphones have a screen, microphone and the ability to vibrate, they provide the same benefits to the end user as this wearable assistant device but with a better user experience.

There are also several independently produced, commercial solutions, addressing the gap in accessibility for hearing impaired drivers. The most relevant being the PionEar by (Říha, 2023). Originally a Hackaday project, the PionEar provides early warning to deaf drivers of an approaching emergency vehicle with a simple light-up approach. It is small, lightweight and energy efficient in design, making it an ideal solution to such a problem.

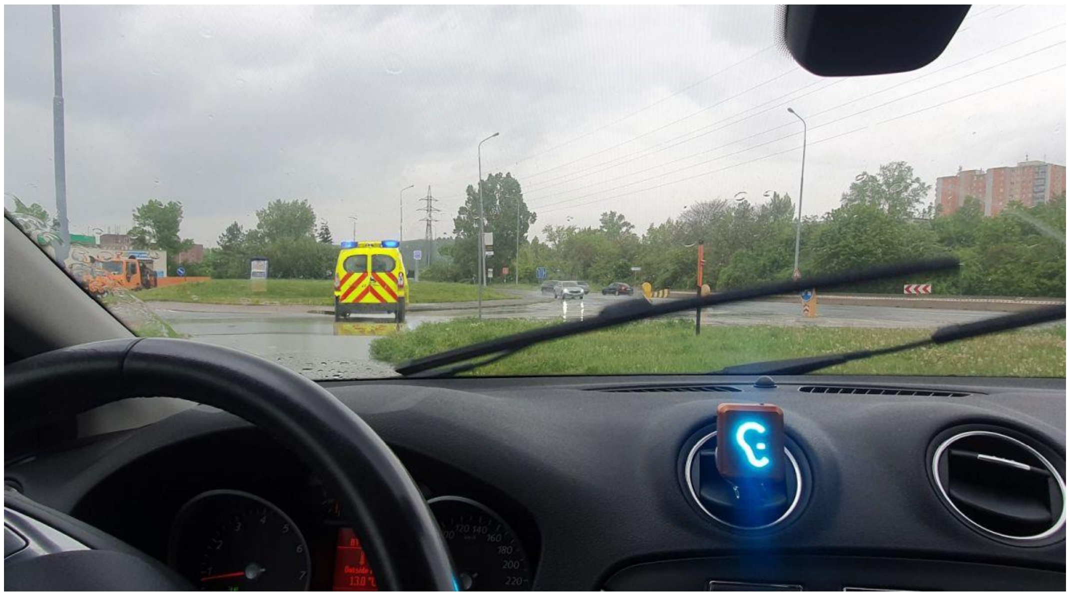


Figure 3: PionEar device on dashboard of vehicle.

The PionEar is a 3D printed box housing a machine learning circuit board and internal microphone. Having the microphone inside the vehicle does limit the effective range of the product in part due to the innate sound suppression of the vehicle. Furthermore, as a hardware solution it is expensive to mass produce. A software solution has the flexibility of being freely available, accessible on a myriad of devices, and has the functionality to either use the device's microphone or a Bluetooth microphone attached outside the vehicle (increasing the effective range). Interestingly, the PionEar is built upon the same dataset of siren and traffic noise as this project's proof-of-concept model, created for the interim report. This does provide confidence that results will be similarly beneficial and meaningful to the end user.

From previous works, we can pinpoint aspects to incorporate and sidestep features which do not fit into this project's overarching scope. Firstly, from the PionEar, we know that the publicly available dataset the 'sireNNet-Emergency Vehicle Siren Classification Dataset for Urban Applications' (Shah and Singh, 2023) is large and detailed enough to produce an accurate model. We also know to avoid creating a device where the microphone is inside the vehicle, as mentioned earlier, this drastically reduces the model's siren detection range. Although being a hardware approach, the wearable assistant device by (Chin et al., 2023) includes apt alternatives to gaining attention such as vibrating and updating the display when a siren is believed to have been heard.

2.4 Defining goals from literature

Based on the findings from previous works in the literature, several project goals can be defined. These goals are presented in the table below, and each include a description of the particular literature which influenced their creation.

Project Goal	**Description**
Improve safety of hearing impaired individuals	(Picard et al., 2008) explores the extensive impact of a HI on driving and (British Academy of Audiology, 2023) magnify the extent of people struggling with a HI in the UK alone. Combined, these sources show a void in accessibility and a clear, broad demographic of possible users.

Create a model with real-time performance	(Ramirez et al., 2022) proved that using a CNN allows for real-time testing of models, and combined with other forms of testing (performance metrics and synthetic data), can provide evidence a ML approach is viable.
Making an accessible and affordable artefact	The PionEar by (Říha, 2023) is an excellent hardware approach to the same problem, but does incur costs for production of components. With a software approach, the artefact can cost nothing to the end user, and be widely accessible on a myriad of platforms and devices.
Maximise detection range	(Říha, 2023; Potter et al., 1977; Marchegiani and Newman, 2022) provide evidence that a device listening from the inside of a vehicle will not be nearly as effective in a modern vehicle with significant sound suppression. To maximise the detection range, the microphone should be installed on the outside of the vehicle, which will mean a dataset including general environmental noise will need to be either found or augmented.
Tuning a model for a high degree of accuracy	(Ramirez et al., 2022) has proven that a CNN approach is not only viable but capable of a high level of accuracy. The ML model created should be able to recognise siren sounds with a similarly high degree of accuracy, ~98% on test datasets.

Table 2: Literature's influence on project goals.

Chapter 3

Requirements Analysis

Since the original project proposal, the scope of this project has both expanded and shrunk where expectations and reality have intertwined. Despite this, the main aim of this project remains unchanged: enhancing the awareness and safety of hearing-impaired drivers with a machine learning approach to detecting ESV siren sounds. To achieve this aim, several objectives need to be fulfilled. Below are the SMART objectives from the original project proposal document, followed by amendments and updated objectives to fit the new scope.

3.1 Original SMART objectives

The italicized SMART objectives are taken from (Harwood, 2023), the original project proposal.

Find and collect a dataset of emergency vehicle sirens.
The dataset should be more than 200 true positive and true negative datapoints each with a sample rate of 22,050 Hz at minimum and split into ~3 second clips.

It was originally assumed that a publicly available dataset would be pre-processed so that each datapoint was of the same length and sample rate, however this was not the case. The dataset chosen, the 'sireNNet-Emergency Vehicle Siren Classification Dataset for Urban Applications' (Shah and Singh, 2023) consists of four classes of data distributed as:

- Ambulance Siren – 400 audio files
- Police Siren – 454 audio files
- Fire Engine Siren – 400 audio files
- Traffic – 421 audio files

This dataset consists of augmented datapoints, which provide ample data for the complexity of the models being trained. Nevertheless, a pre-processing step has been implemented to modify data to the model's needs. Firstly, audio files are made uniform in length using zero padding (adding zeros to the end of the sequence until it reaches the desired length. (Kamali, 2023)). Secondly, down-sampling the audio files sample rate will facilitate faster data processing in real-time. The original recommended sample rate of 22,050 Hz and ~3 second clip duration have been maintained as the default values for the real-time data processing pipeline, however different values have been experimented with to test their efficacy.

Produce a multi-class classification model for recognising siren sounds.

The model should be capable of recognising different emergency vehicle types. Audio snippets (~3 seconds long) will be taken from a microphone in real time and passed through the model.

A multi-class classification model capable of recognising different emergency vehicle types has been trained. However, a binary classification model has also been trained in order to find which provides the greatest benefit to the end user. The reason for this is that multiclass and binary classifiers provide different value to the end user; and without testing individually neither can be favoured. A multi-class classifier would be able to determine which type of ESV is approaching, however it would return results slightly slower, and has the increased likelihood of misclassifying ESV sirens. It is also important to consider if there is any value in knowing which type of emergency vehicle is approaching as each needs to be treated the same. Also, if a ESV siren is misclassified for another, this could cause a driver confusion and potentially lead to a dangerous situation where a driver is looking for a ESV which simply does not exist.

"Audio snippets (~3 seconds long) will be taken from a microphone in real time" – from the outside of the vehicle. This is to hopefully improve the model's siren detection range. Also, considering that modern vehicles have significant sound insulation, and there is a variability in noise levels among them, a model trained on recordings taken inside one vehicle may prove unsuitable even unusable for a multitude of others. This data will be parsed through the machine learning model which will output the predicted class to a visual display. The details regarding displaying data will be discussed in Chapter 4, Design and Methodology.

Tune the model for high recall.

High recall is a desirable quality for the model as missing a siren can have significant consequences. The model will be tuned for a high recall, with the trade-off being reduced precision.

Recall is the number of correct results relative to the number of expected results:

$$Recall = \frac{|G \cap P|}{|G|} \tag{1}$$

Ground truth (G), Predicted result (P).

Recall is a common measure alongside precision to evaluate performance, and often increasing one comes at a trade-off of reducing the other. Recall can be increased by lowering the selection threshold to provide more predictions, at the cost of decreased precision (Fränti and Mariescu-Istodor, 2023). The performance of both precision and recall can be combined to a single metric, F1-Score.

$$F1 = 2\frac{|G \cap P|}{|G| + |P|} \tag{2}$$

Ground truth (G), Predicted result (P).

F1-Score is the harmonic mean of recall and precision, and for a real-time model where false positives can lead to significant distractions for the end user, maximising recall whilst ignoring precision could be counterintuitive to the model’s main function. Therefore, the models have been tuned for a high F1-Score as opposed to a high recall value.

Implement a feature to save and store new true positive data points to the dataset for future training.

This should improve the accuracy of the model for a user, as it reduces the model's reliance on the original dataset.

The challenge in training neural network (NN) models on synthetic data is "the generalisation of the trained models to real data, as that process requires careful identification of a training set and the inclusion of realistic noise and other variables between synthetic and real data" (Alkhalifah et al., 2021). Implementing a feature to append new datapoints to the dataset is an attempt to solve for the inevitable generalisation which comes with a synthetic dataset. Although this will have to be done separately from the main artefact due to constraints relating to suitable and available technology to record data, it does mean that overall the model will be better fitted to the end user's vehicle and environment.

Importantly, false positive and false negative data should be recorded for future training as it is beneficial to include data the model has not been able to correctly classify. Traffic data should also be replaced with environmental noise in areas where ESVs are common, this will help the model to distinguish between sounds associated with ESVs and general background noise.

3.2 Updated SMART objectives

Based on the original SMART objectives and the goals aligning with the findings of the literature review, a set of new, updated SMART objectives have been created. Note, that objectives are written in the past tense for consistency with the previous objectives' sentence structure.

This project's primary aim is to "enhance the awareness and safety of hearing-impaired drivers with a machine learning approach to detecting ESV siren sounds." To fulfil this aim, several objectives needs to be achieved:

Find, collect and preprocess a dataset of emergency vehicle sirens.

An online synthetic dataset will be used to train the machine learning models, and should consist of a minimum 200 datapoints, ~3 seconds each, for each class. Data will be sent through a pre-processing pipeline in a Jupyter Notebook integrated development environment (IDE) where sample rates will be set to a standard level, and audio clips will be given a uniform duration. Audio clips should have a sample rate of around 22,050Hz so new data points can be processed in real-time while maintaining sufficient detail for the model to classify effectively. A ~3 second audio clip duration has been chosen to strike a balance between swift and accurate classifications, with the innate computational overhead of real-time data processing.

Produce a binary and multi-class classification model for recognising siren sounds.

Both a binary and multi-class classifier model are going to be trained on a synthetic dataset. Both will use the same pipeline for development and testing. Models will be interchangeable and should provide class predictions at a standard ~3 second interval. Of course, due to the simplicity of the binary classifier, it is plausible for this model to provide a more constant output. However, to guarantee fair benchmarking of model performance, each model will be tested on real world data (recorded in the City of Lincoln) with a standard ~3 second interval output for classification predictions. All class predictions will then be graphed into confusion matrices, and discussed at length for viability in Chapter 6, Results and Discussion.

Tune the models for F1-Score.

A high F1-Score is desirable for the model as it combines recall, the ability to find all relevant cases, and precision, the ability to find only the relevant data points. With the multiclass classification model, each class will have a unique F1-Score, treating each class as if its own classifier (in a one-vs-rest manner). A classifiers F1-Score will be a key factor when comparing the ML models.

Implement a feature to save and store new data points to the dataset for future training.

Using audio data taken from real-world testing will improve the efficacy of the model in its environment. As mentioned before, testing will take place throughout the urban streets of Lincoln. An urban city setting will provide ample vehicle and pedestrian noise, which will improve the robustness, and help regularise the model (in other words, regularisation will discourage over complicating a model and prevent overfitting). In return we should see a reduction in false positive and false negatives, consequently improving the F1-Score.

The intended functionality of this feature is not to create tailor made machine learning models for a driver's environment, but rather to improve model performance in environments where it is most useful, such as busy urban roads and junctions, as suggested by existing literature.

Chapter 4

Design & Methodology

4.1 Software development methodology

The software development process was very much focused on creating a minimum viable product (MVP), a proof-of-concept, as quickly as possible in order to define the scope of what was possible and to prevent over-promising on aims and objectives.

To follow this routine, a rapid application development (RAD) methodology has been followed loosely. Most methodologies such as RAD are dependent on a responsive customer, however as the aims and objectives of this project have already been defined, to fill in this blank, weekly supervisory meetings took up this role, and provided a structure for what needed to be completed to fulfil an MVP. This methodology is a change to the original project plan as outlined in the project proposal. The project plan used a Gantt Chart to outline the time span in which to complete tasks.

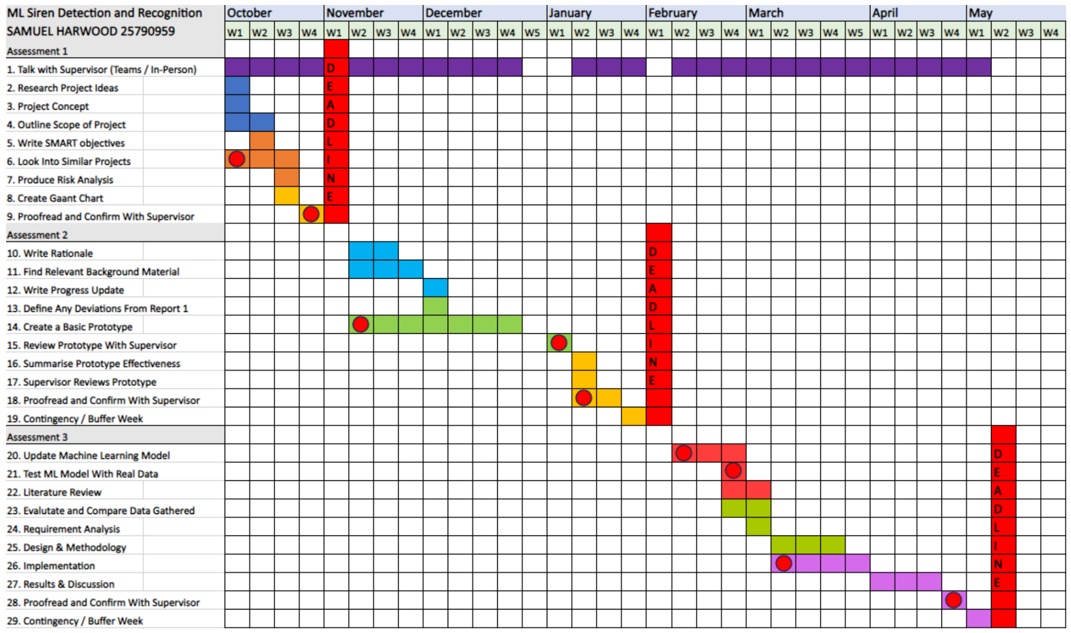
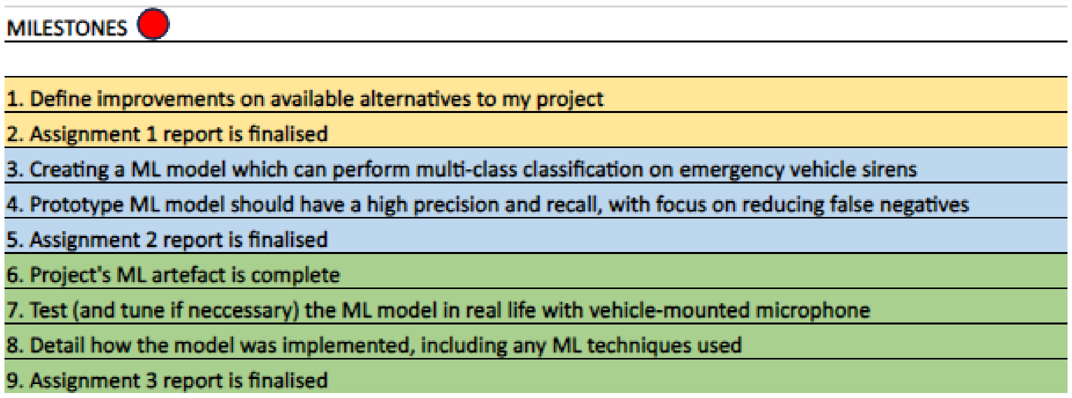


Figure 4.1: Gantt Chart (weekly granularity) (Harwood, 2023).



MILESTONES ●

1. Define improvements on available alternatives to my project
2. Assignment 1 report is finalised
3. Creating a ML model which can perform multi-class classification on emergency vehicle sirens
4. Prototype ML model should have a high precision and recall, with focus on reducing false negatives
5. Assignment 2 report is finalised
6. Project's ML artefact is complete
7. Test (and tune if neccessary) the ML model in real life with vehicle-mounted microphone
8. Detail how the model was implemented, including any ML techniques used
9. Assignment 3 report is finalised

Figure 4.2: Gannt Chart milestones (Harwood, 2023).

With this new methodology, tasks were consistently being completed before the deadline. With this additional time, it was agreed during supervisory meetings that more focus could therefore go into expanding the scope of the project, and this is evident from the updated Aims and Objectives.

4.2 Choice of model

For both the binary and multi-class classifiers, the Keras Sequential model API is utilised. The sequential class groups a linear stack of layers into a model (Keras, 2023a) allowing customisation for each layer and their unique parameters. A layer is the highest-level building block in deep learning, the first and last layers in a network are input and output layers respectively, and all layers in-between are called hidden layers (Dettmers, 2015); it is with these hidden layers that data can be transformed. The original proof-of-concept siren recognition model had a simple structure with very few layers, this model's complexity is depicted in the below graphic with a comparison to the finalised model structure:

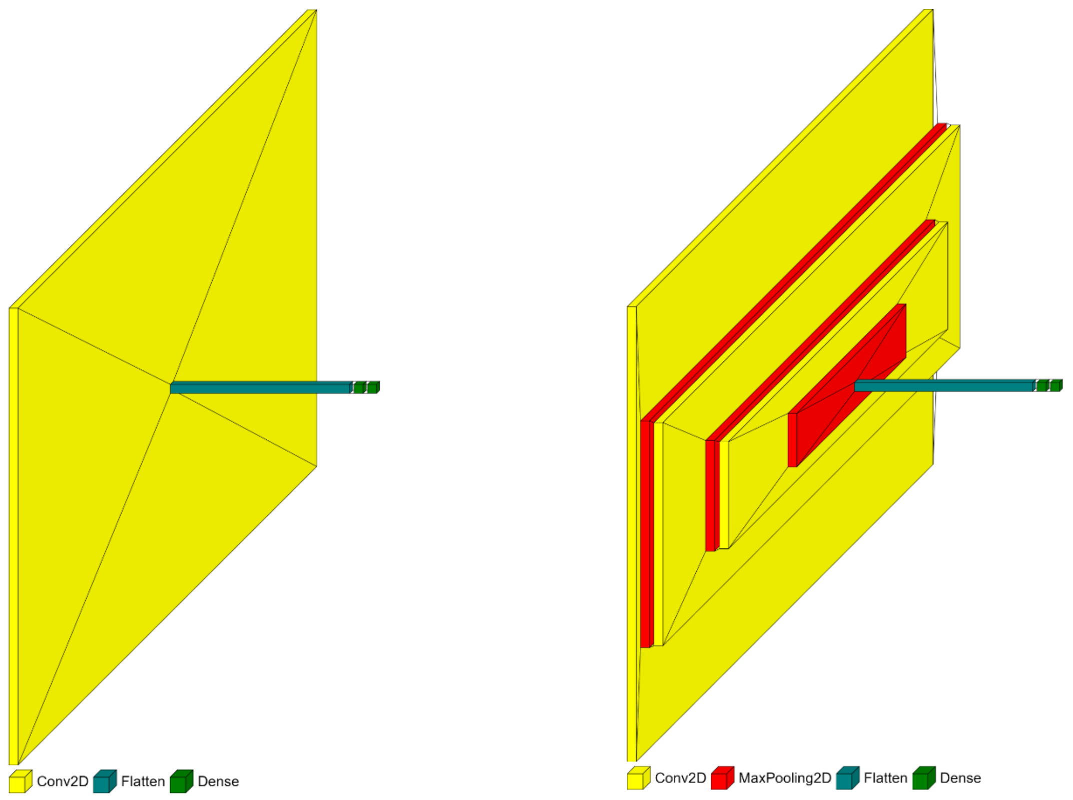


Figure 5: Original sequential model layers (left). Updated sequential model layers (right).

The original sequential model had a total of four layer layers, with three unique layer types:

- Conv2D: The two-dimensional convolutional layer creates a convolution kernel convolved with the layer input over a single spatial or temporal dimension to produce a tensor of outputs (Keras, 2023b). As audio data unfolds over time, the convolution occurs along the temporal dimension, however in this case, audio data is represented in spectrogram form, and so it applies convolution along the spatial dimensions. Conv2D essentially uses a filter or a kernel to ‘slide’ over 2D input data, performing elementwise multiplication (Databricks, 2019), the result of this process will sum the results into a single pixel output, and the kernel will continue this operation for the entire 2D matrix of features. The

Conv2D layer is excellent at feature detection as it is translation-invariant, meaning the same pattern or feature will be found irrespective of the location it appears in the data. This property is crucial for a model handling time-series data where in real-time, consistent and accurate recognition is required no matter the temporal position of a siren.

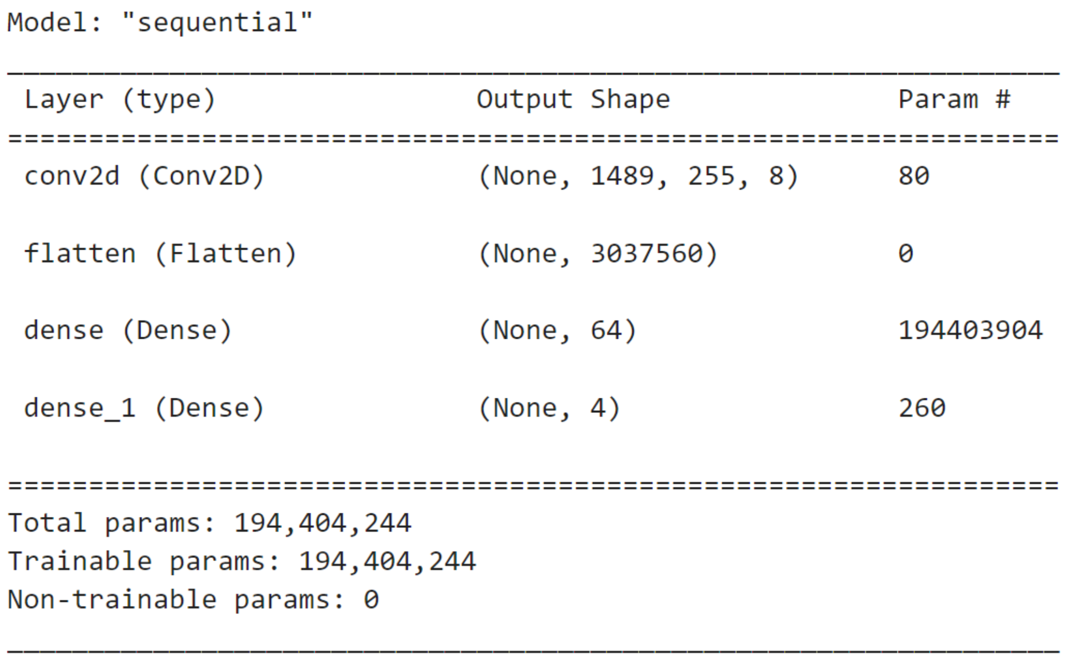
- Flatten: The flatten layer works as an intermediary component between a CNN and ANN comprising of dense layers (Shah, 2023). The flatten layer's function is to convert the output format of a CNN into a format the ANN can understand, which is done by converting a multi-dimensional array into a one-dimensional array. In the case of the original model, the layer shape of (None, 1489, 255, 8) is the output shape from the Conv2D layer, which is the input of the flatten layer. The flatten layer converts this three-dimesional array into a one-dimensional array with a length equal to the values multiplied together: ($1489 \times 255 \times 8 = 3{,}037{,}560$)

- Dense Layer: A dense layer is deeply connected with the preceding layer, it connects every neuron in one layer to every neuron in the next, which provides the necessary network to learn complex relationships between features in the data.

The updated model has a total of nine layers, with three of these being a type of layer not found in the original, MaxPooling2D:

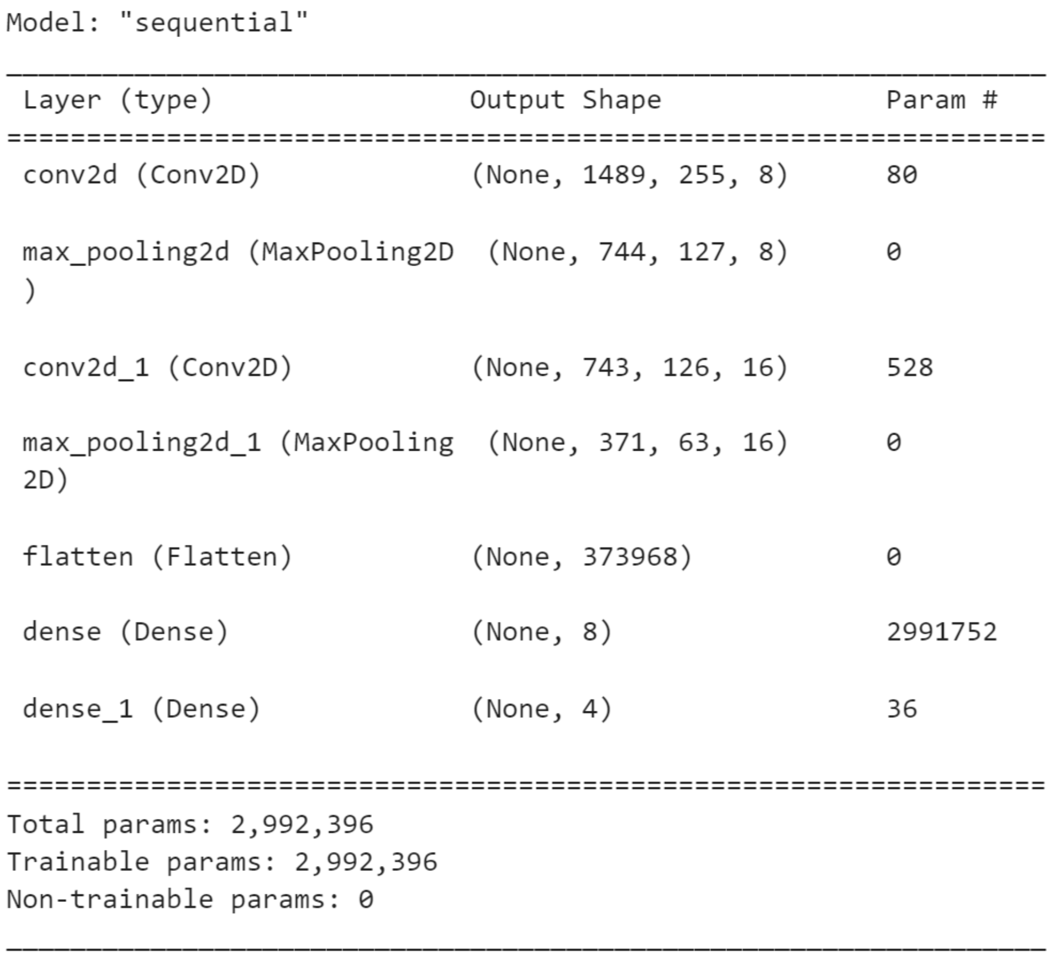
- MaxPooling2D: An operation for two-dimensional spatial data which down-samples the input along the spatial dimensions. It performs this operation by taking the maximum value over an input window for each channel of the input, this window is shifted by strides along each dimension (Keras, 2023c). MaxPooling reduces the overall computational complexity of the model with very little impact on the values of most pooled outputs.

To ensure a fair comparison between models, the model fitting process has been kept identical. Firstly, both are using the Adam optimiser, a method for “stochastic optimisation which is computationally efficient with little memory requirement, well suited for problems that are large in terms of data/parameters” (Kingma and Ba, 2014). The optimiser has a learning rate set to 0.001, the default value. The categorical cross entropy loss function has been utilised as standard for use with the softmax activation in the output layer for multi-class classification tasks (Wood, 2019).



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 1489, 255, 8)      80

 flatten (Flatten)           (None, 3037560)           0

 dense (Dense)               (None, 64)                194403904

 dense_1 (Dense)             (None, 4)                 260

=================================================================
Total params: 194,404,244
Trainable params: 194,404,244
Non-trainable params: 0
_________________________________________________________________
```

Figure 6.1: Original sequential model layers, output shape and parameters.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 1489, 255, 8)      80

 max_pooling2d (MaxPooling2D  (None, 744, 127, 8)      0
 )

 conv2d_1 (Conv2D)           (None, 743, 126, 16)      528

 max_pooling2d_1 (MaxPooling  (None, 371, 63, 16)      0
 2D)

 flatten (Flatten)           (None, 373968)            0

 dense (Dense)               (None, 8)                 2991752

 dense_1 (Dense)             (None, 4)                 36

=================================================================
Total params: 2,992,396
Trainable params: 2,992,396
Non-trainable params: 0
_________________________________________________________________
```

Figure 6.2: Updated sequential model layers, output shape and parameters.

When summarising both these models, we can see that despite the increase in layer complexity, there are significantly less trainable parameters in the updated model. This is mostly due to the configuration of the dense layers units, the positive integer responsible for the dimensionality of the output space (Keras , 2023d), which in the original model is set to 64, meaning 64 nodes receive input from the preceding flatten layer and outputs based on learned parameters (weights). More nodes typically means a greater ability to capture intricate patterns in data, however in the case of the original model it has had the negative effect of increasing the number of parameters, has increased the potential to overfit with the knock on effect of significantly longer training times. It is necessary to note that a greater number of parameters does not yield a more competent model, which becomes apparent when analysing and comparing the model’s evaluation metrics.

F1-Score is the chosen metric to analyse model performance. Each class will have an F1-Score measured independently to avoid intrinsic weighting biases from the slightly skewed class imbalance. An early stopping procedure has also been included, after five epochs without improvement to the validation loss score, the model stops and returns to the state with the best weights.

4.3 Performance evaluation

Model evaluation took place within the IDE. During model training, the model preserves all metric values within a history object. These values then are graphically represented using matplotlib, a comprehensive library for data visualisation (Matplotlib, 2012), making analysis of performance very simple.

The two most important metrics are F1-Score, our metric of performance, and Loss, our metric to determine generalisation, overfitting or underfitting.

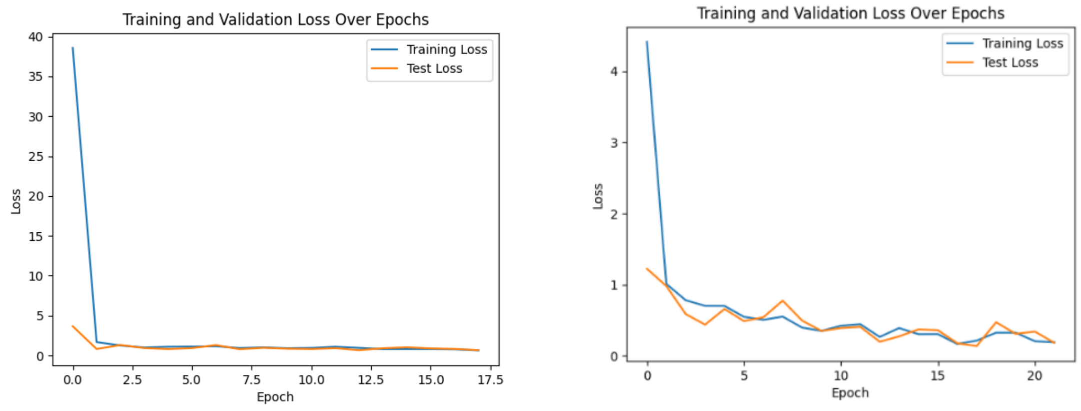


Figure 7.1: Graphical representation of loss and validation loss (train, test/validation). Original model (left) spans 17 epochs, updated model (right) spans 22 epochs.

The loss metrics for both models are promising, trending downwards and achieving a local minima. The original model has a steep drop which quickly plateaus, whereas the updated model takes longer, and differences in training and validation metrics are more sporadic. Furthermore, the values for validation loss (also known as test loss) is generally higher than the training counterpart. This behaviour is expected and shows the updated model is continuing to learn, although this behaviour does not occur in the original model. In fact, the plateau implies the model has generalised well to unseen data, but in spite of that, considering this is after just two epochs, this is more of a cause for concern.

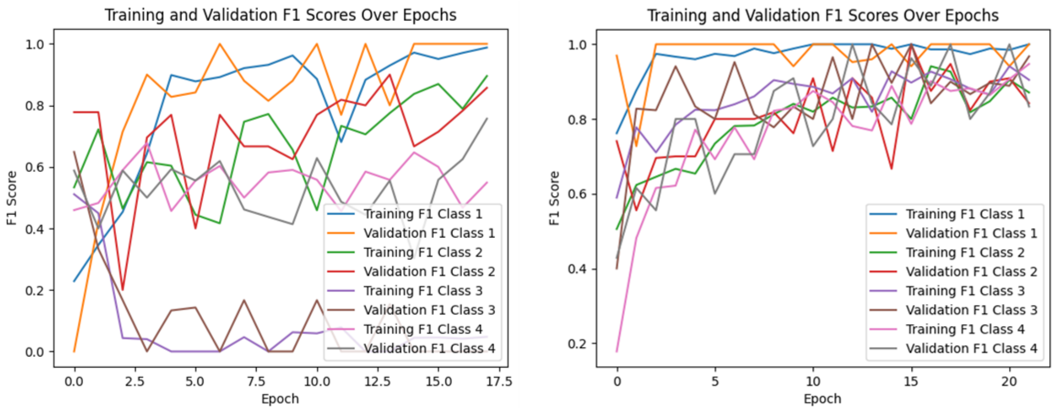


Figure 7.2: F1-Score metrics. Original model (left) updated model (right).

The original model's F1-Score for class 3 (police car siren) poses an interesting anomaly in what could otherwise be considered a suitable model. Interestingly, this is something only affecting the original model. In both models, class 1 (traffic noise) has best performed in terms of predicting instances belonging to that class, this is expected given its distinction from other classes. The 20th epoch of the updated model can be considered the most performant according to the F1-Score, whereas the loss metrics would suggest the 16th epoch to be the best as this is where the model generalises well to the validation data while maintaining a low training loss value.

According to the metrics, the updated model is able to accurately classify each class with a high degree of recall and precision, and far outperforms the original. Even so, both these models have a cause for concern, that is the time to classify. As explained in Chapter 3 Aims and Objectives, class predictions are expected to run at around ~3 second intervals. This value is due in part to the fact that differences in sirens are minimal and therefore more data is needed to guarantee an accurate prediction. What is more, a single ESV may use a different version of siren depending on the situation. In the UK, police vehicles use a mixture of short, long and contrasting tones depending on their location, time of day and congestion of traffic (Brook, 2024). Considering that each of these sirens will have a unique fluctuation in pitch, intensity, patten and modulation, makes classification based on type of ESV challenging (Harwood, 2024), which is why a ~3 second interval for the multi-class classifier is necessary to ensure enough data is ingested to prompt an accurate prediction. Of course, this constraint does not exist for the binary classifier, which is why an attempt has been made to modify this into a lightweight, faster model.

4.4 Modifications

Firstly, for context the multi-class classifier processes a spectrogram which is effectively treated as an image, therefore two spatial dimensions are used. The kernel moves on the x-axis and y-axis to calculate the convolutional output, which is a two-dimensional matrix. Moving across the spectrogram in two directions is advantageous for collecting as much information as possible and the ~3 second interval gives ample time for this process to take place. Instead, if we remove a dimension and work solely with time-series data we reduce the dimensionality, and enhance computational efficiency whilst still retaining important temporal features. This means we can still decipher if a siren is heard, just at a much faster rate, reducing the time it takes to alert a user. To practically implement this, the model layers need to be setup to handle just one direction rather than two. So, Conv2D and MaxPooling2D layers found in the multi-class classifier are replaced with Conv1D and MaxPooling1D, and dimensionality can be reduced by simply reshaping the input.

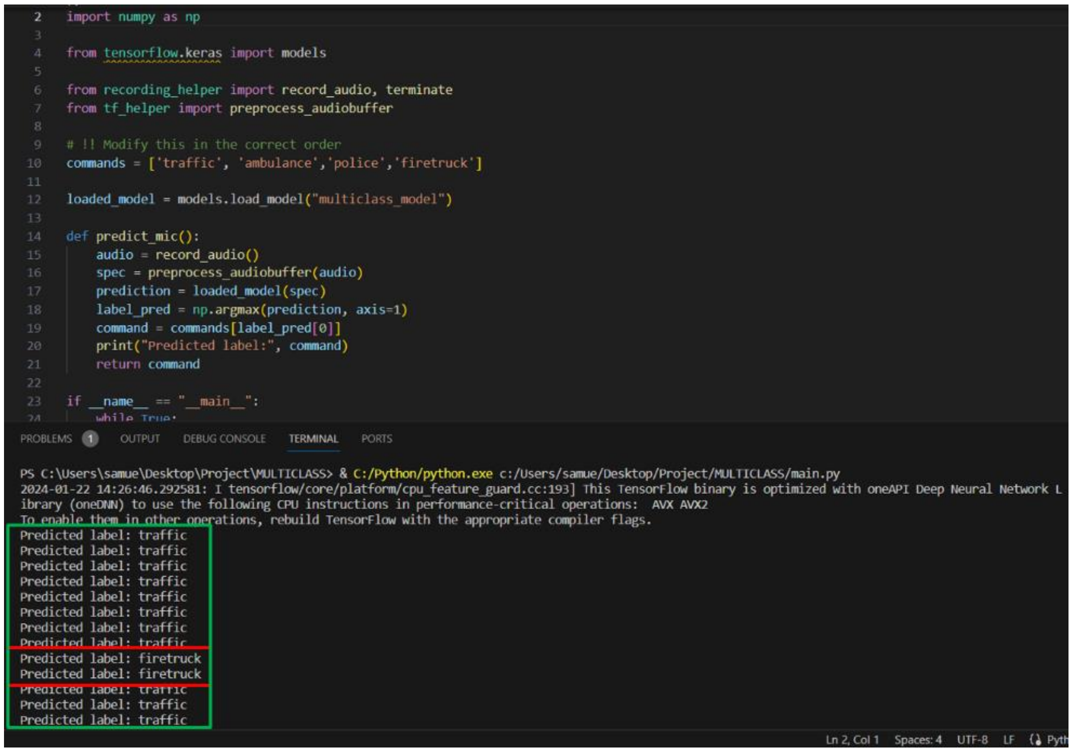
	Conv Direction	Input	Filter	Output
tf.nn.conv1d	1-direction →	3-dimensions	3- dimensions	2- dimensions
tf.nn.conv2d	2-direction → ↓	4- dimensions	4- dimensions	3- dimensions
tf.nn.conv3d	3-direction ↗ → ↓	5- dimensions	5- dimensions	4- dimensions

Table 3: Direction of operation for CNNs in TensorFlow (Thakur, 2020).

4.5 Creating a usable final artefact

As mentioned in Chapter 3 Aims and Objectives, the artefact needs to be accessible by an end user, meaning the model should be in a format which is safe to use in a motor vehicle. Ideally, the final artefact would be incorporated into the entertainment system of modern vehicles, linking a microphone on the outside of the vehicle which passes audio data to the model for classification. When a siren is recognised, the screen would present a warning message, alerting the driver (Harwood, 2024). The best alternative to this is a smartphone, a device the majority of the population own, carry with them at all times and often use as a replacement for an entertainment system or for help with navigation, where the device is directly in the driver’s field-of-view.

In this project’s interim report it was also suggested that the final artefact should be able to record new siren recognitions, gradually transitioning from an online crowdsourced dataset to field data collected by in-vehicle testing around the City of Lincoln (Harwood, 2024). Unfortunately, due to limitations with connecting external microphones to a smartphone, two artefacts need to be created. One, which is run on a laptop, will be used solely for recording high quality audio data for future inclusion in the model’s dataset. The other will be designed with the end user in mind, operated on a smartphone. Despite this inconvenience, a prototype of the model running in real-time on a laptop has already been created in a previous report, and it was this which could be relied upon to fulfil several of the project’s objectives.

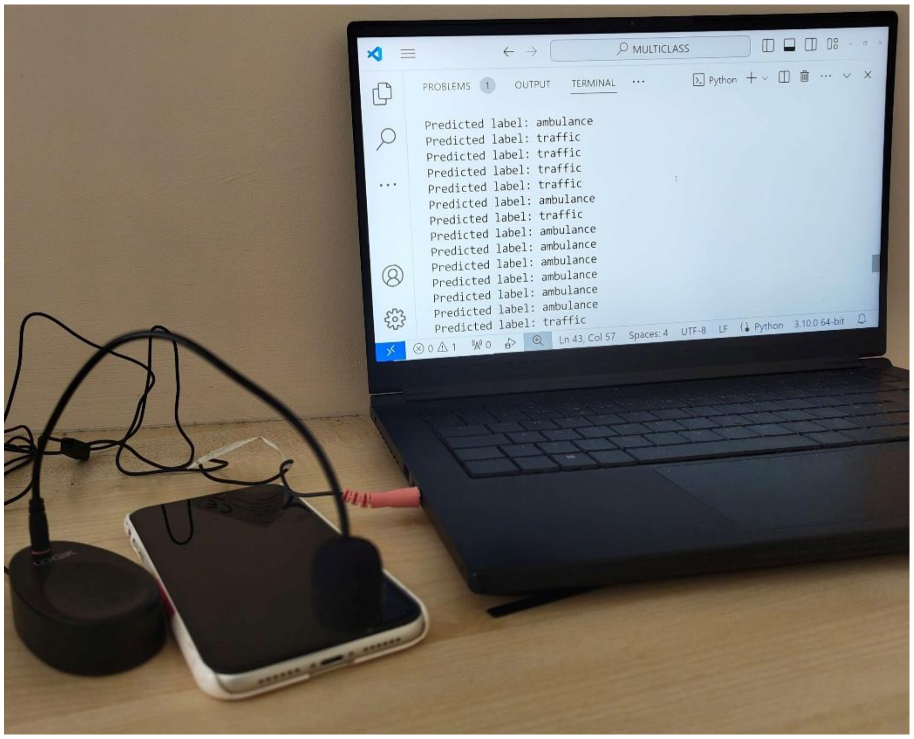


```python
import numpy as np

from tensorflow.keras import models

from recording_helper import record_audio, terminate
from tf_helper import preprocess_audiobuffer

# !! Modify this in the correct order
commands = ['traffic', 'ambulance','police','firetruck']

loaded_model = models.load_model("multiclass_model")

def predict_mic():
    audio = record_audio()
    spec = preprocess_audiobuffer(audio)
    prediction = loaded_model(spec)
    label_pred = np.argmax(prediction, axis=1)
    command = commands[label_pred[0]]
    print("Predicted label:", command)
    return command

if __name__ == "__main__":
    while True:
```

```
PS C:\Users\samue\Desktop\Project\MULTICLASS> & C:/Python/python.exe c:/Users/samue/Desktop/Project/MULTICLASS/main.py
2024-01-22 14:26:46.292581: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network L
ibrary (oneDNN) to use the following CPU instructions in performance-critical operations:  AVX AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
Predicted label: traffic
Predicted label: traffic
Predicted label: traffic
Predicted label: traffic
Predicted label: traffic
Predicted label: traffic
Predicted label: traffic
Predicted label: traffic
Predicted label: firetruck
Predicted label: firetruck
Predicted label: traffic
Predicted label: traffic
Predicted label: traffic
```

Figure 8.1: Realtime audio recognition code (Harwood, 2024).

This code with a few modifications to use a newer version of the model and implementing a record feature, is now capable of recording the values it processes. If the model predicts the audio clip is a fire engine siren, then the audio file will be sent to the fire engine folder of the original dataset. Each clip has the hour minute and second of the recording as a timestamp attatched to the filename to differentiate them from the original dataset.

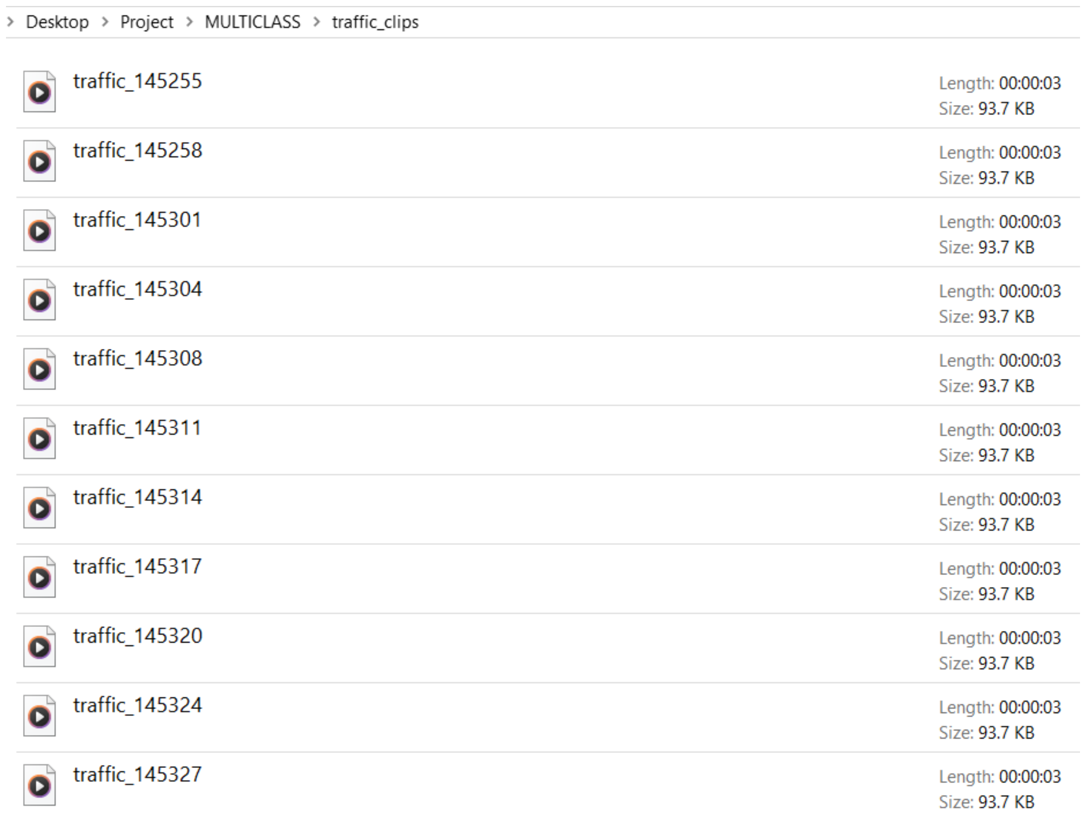


```
Predicted label: ambulance
Predicted label: traffic
Predicted label: traffic
Predicted label: traffic
Predicted label: traffic
Predicted label: ambulance
Predicted label: traffic
Predicted label: ambulance
Predicted label: ambulance
Predicted label: ambulance
Predicted label: ambulance
Predicted label: ambulance
Predicted label: ambulance
Predicted label: traffic
```

Figure 8.2: Setup for testing ML models.

```python
def save_audio(filename, audio_clip):
    with wave.open(filename, 'wb') as wf:
        wf.setnchannels(1)
        wf.setsampwidth(2)  # 16-bit audio
        wf.setframerate(16000)  # Sample rate 16 kHz
        wf.writeframes(audio_clip.tobytes())


if __name__ == "__main__":
    while True:
        command, audio_clip = predict_mic()
        save_dir = (f"{command}_clips") #Save to folder specific for that class
        if not os.path.exists(save_dir):
            os.makedirs(save_dir)

        #Adds Hour Minutes Second to file name
        timestamp = datetime.datetime.now().strftime("%H%M%S")
        filename = os.path.join(save_dir, f"{command}_{timestamp}.wav")
        save_audio(filename, audio_clip)
```

Figure 8.3: New save audio function.



```
› Desktop › Project › MULTICLASS › traffic_clips

traffic_145255    Length: 00:00:03   Size: 93.7 KB
traffic_145258    Length: 00:00:03   Size: 93.7 KB
traffic_145301    Length: 00:00:03   Size: 93.7 KB
traffic_145304    Length: 00:00:03   Size: 93.7 KB
traffic_145308    Length: 00:00:03   Size: 93.7 KB
traffic_145311    Length: 00:00:03   Size: 93.7 KB
traffic_145314    Length: 00:00:03   Size: 93.7 KB
traffic_145317    Length: 00:00:03   Size: 93.7 KB
traffic_145320    Length: 00:00:03   Size: 93.7 KB
traffic_145324    Length: 00:00:03   Size: 93.7 KB
traffic_145327    Length: 00:00:03   Size: 93.7 KB
```

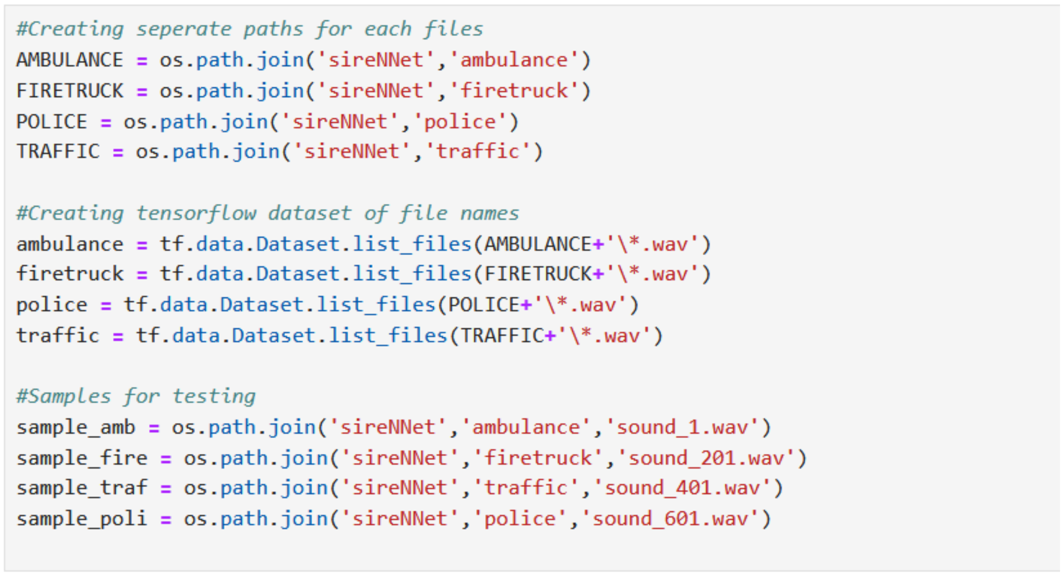
Figure 8.4: Timestamped audio recordings.

Chapter 5

Implementation

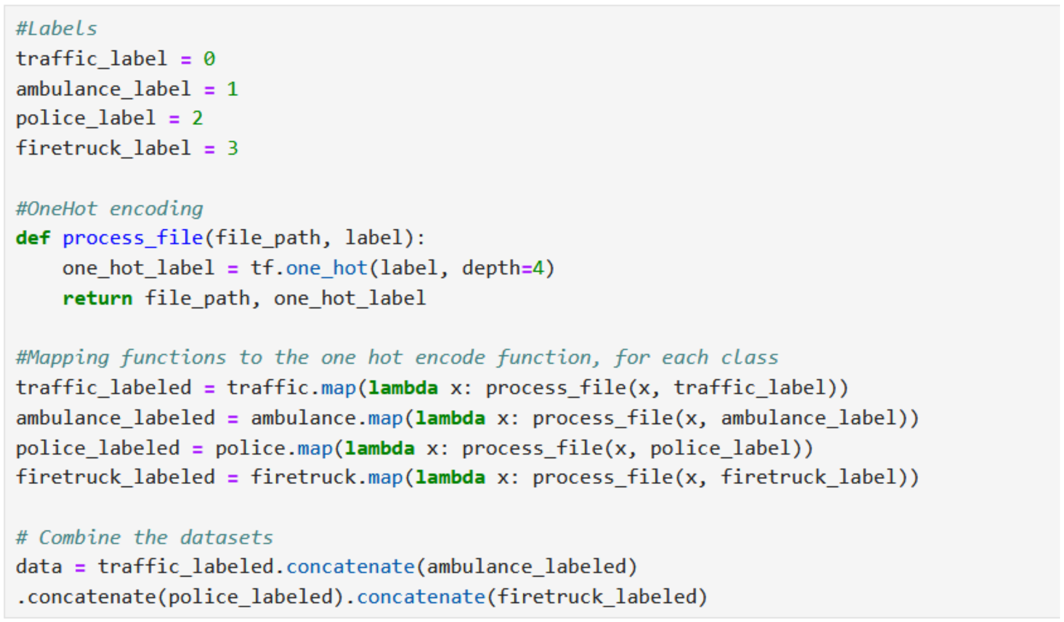
5.1 Original data pipeline

First, audio data needs to be ingested and placed into separate classes for label encoding. Both models use the same dataset the "sireNNet-Emergency Vehicle Siren Classification Dataset for Urban Applications" (Shah and Singh, 2023) and so ingestion is similar for both models.



```
#Creating seperate paths for each files
AMBULANCE = os.path.join('sireNNet','ambulance')
FIRETRUCK = os.path.join('sireNNet','firetruck')
POLICE = os.path.join('sireNNet','police')
TRAFFIC = os.path.join('sireNNet','traffic')

#Creating tensorflow dataset of file names
ambulance = tf.data.Dataset.list_files(AMBULANCE+'\*.wav')
firetruck = tf.data.Dataset.list_files(FIRETRUCK+'\*.wav')
police = tf.data.Dataset.list_files(POLICE+'\*.wav')
traffic = tf.data.Dataset.list_files(TRAFFIC+'\*.wav')

#Samples for testing
sample_amb = os.path.join('sireNNet','ambulance','sound_1.wav')
sample_fire = os.path.join('sireNNet','firetruck','sound_201.wav')
sample_traf = os.path.join('sireNNet','traffic','sound_401.wav')
sample_poli = os.path.join('sireNNet','police','sound_601.wav')
```

Figure 9.1: File paths for ESV dataset.



```python
#Labels
traffic_label = 0
ambulance_label = 1
police_label = 2
firetruck_label = 3

#OneHot encoding
def process_file(file_path, label):
    one_hot_label = tf.one_hot(label, depth=4)
    return file_path, one_hot_label

#Mapping functions to the one hot encode function, for each class
traffic_labeled = traffic.map(lambda x: process_file(x, traffic_label))
ambulance_labeled = ambulance.map(lambda x: process_file(x, ambulance_label))
police_labeled = police.map(lambda x: process_file(x, police_label))
firetruck_labeled = firetruck.map(lambda x: process_file(x, firetruck_label))

# Combine the datasets
data = traffic_labeled.concatenate(ambulance_labeled)
.concatenate(police_labeled).concatenate(firetruck_labeled)
```

Figure 9.2: One hot encoding data.

The files are then one hot encoded, meaning to encode categorical features as a one-hot numeric array (ScikitLearn, 2019a), as seen in the tables below.

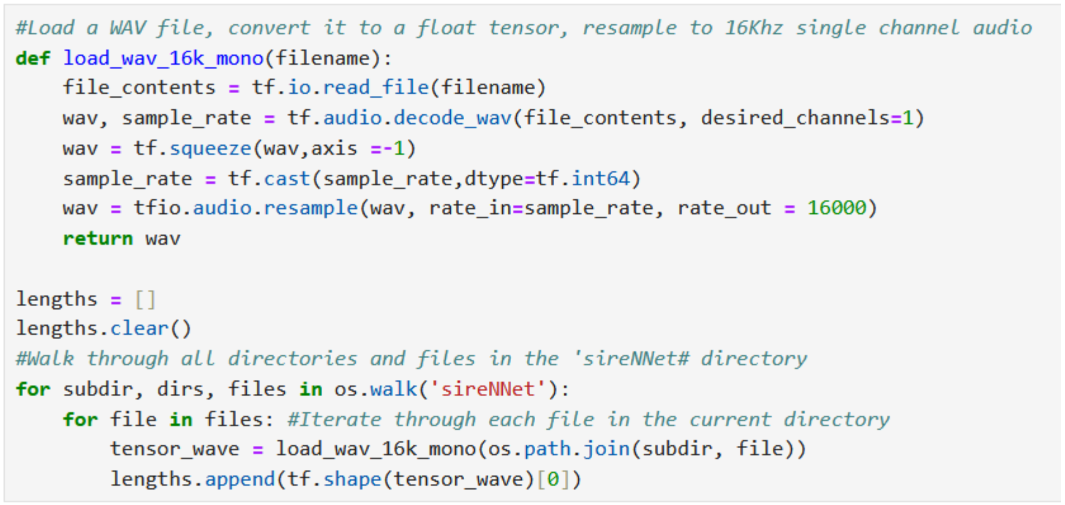
ID	SOUND
0	Traffic
1	Ambulance Siren
2	Police Siren
3	Fire engine Siren

Table 4.1: Label encoder, linking an ID value to each sound.

ID	TRAFFIC	AMBULANCE SIREN	POLICE SIREN	FIRE ENGINE SIREN
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1

Table 4.2: One-hot encoded data, linking an ID to a one-hot encoded vector.

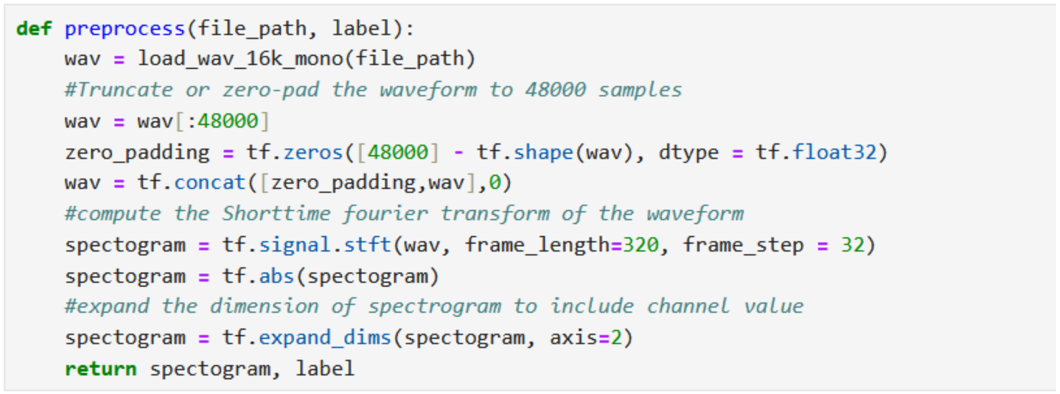
One-hot encoding is the most widely used coding scheme for ML. It compares each level of the categorical variable to a fixed reference level by transforming a single variable with N observations and D distinct values, to D binary variables with N observations each (Potdar et al., 2017). One-hot encoding does not suffer from ordinality issues which stem from label encoding. For example, the label encoded ID of 3 which represents a fire engine siren could be misconstrued as a more important class by the model as it has the highest ID value.



```
#Load a WAV file, convert it to a float tensor, resample to 16Khz single channel audio
def load_wav_16k_mono(filename):
    file_contents = tf.io.read_file(filename)
    wav, sample_rate = tf.audio.decode_wav(file_contents, desired_channels=1)
    wav = tf.squeeze(wav,axis =-1)
    sample_rate = tf.cast(sample_rate,dtype=tf.int64)
    wav = tfio.audio.resample(wav, rate_in=sample_rate, rate_out = 16000)
    return wav

lengths = []
lengths.clear()
#Walk through all directories and files in the 'sireNNet# directory
for subdir, dirs, files in os.walk('sireNNet'):
    for file in files: #Iterate through each file in the current directory
        tensor_wave = load_wav_16k_mono(os.path.join(subdir, file))
        lengths.append(tf.shape(tensor_wave)[0])
```

Figure 9.3: Function for loading .WAV files.

After one-hot encoding, this nested-for loop walks through all the files in the datasets directory and parses them into a resampling function titled load_wav_16k_mono. This function is taken from the (Tensorflow, 2024a) audio data tutorial code. The function’s role is to load in the .WAV files to be formatted with a single channel output (mono), and resample audio to a standard 16kHz value.



```
def preprocess(file_path, label):
    wav = load_wav_16k_mono(file_path)
    #Truncate or zero-pad the waveform to 48000 samples
    wav = wav[:48000]
    zero_padding = tf.zeros([48000] - tf.shape(wav), dtype = tf.float32)
    wav = tf.concat([zero_padding,wav],0)
    #compute the Shorttime fourier transform of the waveform
    spectogram = tf.signal.stft(wav, frame_length=320, frame_step = 32)
    spectogram = tf.abs(spectogram)
    #expand the dimension of spectrogram to include channel value
    spectogram = tf.expand_dims(spectogram, axis=2)
    return spectogram, label
```

Figure 9.4: preprocess function.

The preprocess function performs three key operations. Firstly, not all audio clips are the exact same length, and so files need to be truncated, in this case the first 48,000 samples have been kept. As we know that audio clips have been resampled to a rate of 16kHz (16,000 samples/second), this corresponds to capturing the amplitude of the audio at 48,000 points over 3 seconds.

$$\frac{48000}{16000kHz} = 3\ seconds\ of\ audio \qquad (3)$$

Secondly, it is possible that some audio files will contain less than three seconds of data, in this case zero-padding: a procedure by which the length of a time series is extended by the addition of zeros (Castiglioni, 2005), is necessary to implement. The primary benefit of zero padding is keeping all inputs with a consistent length. For the convolutional layers within a NN, kernels with a fixed size are far more effective at finding all relevant patterns in data which has identical spatial dimensions.

Thirdly and finally, the audio data currently in waveform format is transformed into a spectrogram by expanding the dimensions of the data. To perform this, the Short-time Fourier transform (STFT) of the input waveform is computed. STFT is a sequence of Fourier transforms, a formula to transform a signal sample in time/space to temporal/spatial frequency, with a specific use case for windowed signals. STFT provides time-localised frequency information (Nasser Kehtarnavaz, 2008, 178- 181), ideal for situations where frequency components of a signal vary over time. STFT can be mathematically defined as:

$$\mathrm{X}m(\omega) = \sum_{n=-\infty}^{\infty} x(n)\, w(n - mR) e^{-jwn} \tag{4}$$

$$= \mathrm{DTFT}\omega\big(x \cdot SHIFTmR(w)\big)$$

Where:

DTFT = Discrete Time Fourier Transform

$x(n)$ = input signal at time n

$w(n)$ = length M window function

$\mathrm{X}m(\omega)$ = DTFT of windowed data centred about time mR

R = hop size, in samples, between successive DTFTs

(Smith, 2011)

The STFT computation process can also be described graphically:

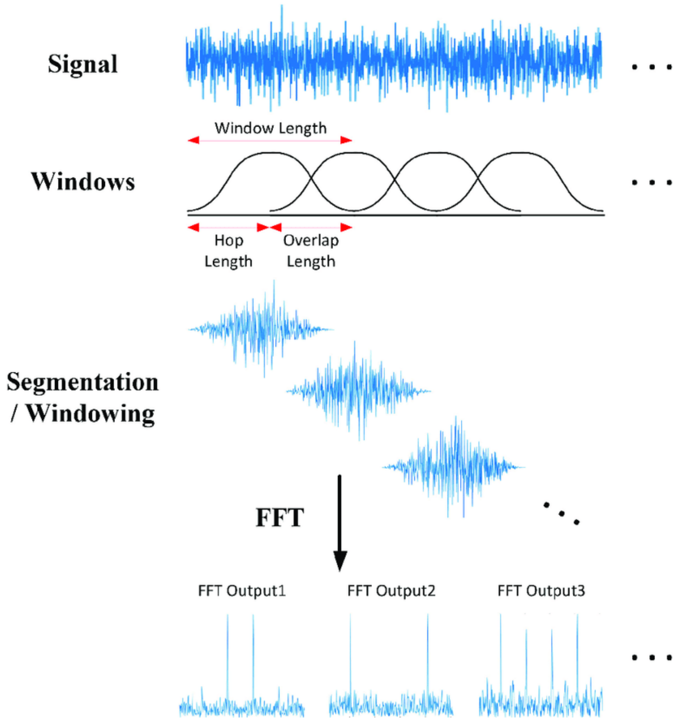


Figure 10.1: STFT overview (Jeon et al., 2020).

Once the STFT is calculated, the remaining data is a complex-valued spectrogram, representing frequency of an audio signal over time, with both magnitude (strength of signal) and phase (sine/cosine phase of each frequency) being represented (Harley, 2022). This data is far too complex to process, and also is not compatible with NNs, therefore the absolute values for magnitude are kept and phase information is disregarded.

Finally, the spectrogram is expanded to include a new dimension for compatibility with a CNN. As a spectrogram is treated as if it is an image, the dimensionality needs to reflect this. The current tensor with three dimensions: batch size, timesteps and features is converted to: batch size, timesteps, channels and features (Renotte, 2022). Here, the batch size and timesteps are analogous with width and height respectively, and the channel is set to 1, as if it is a greyscale image.

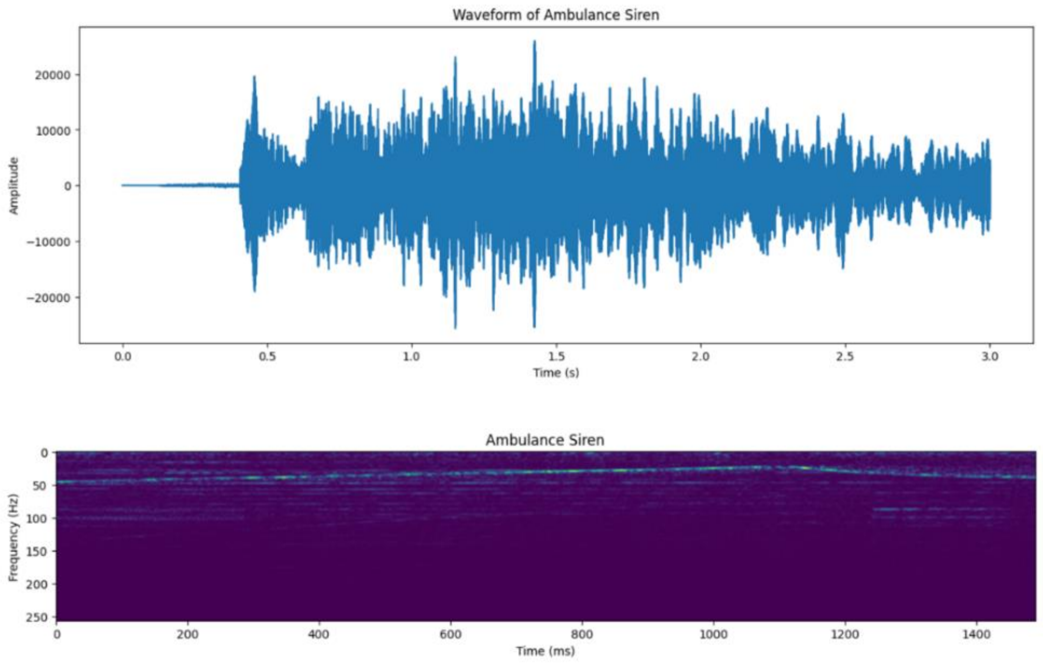
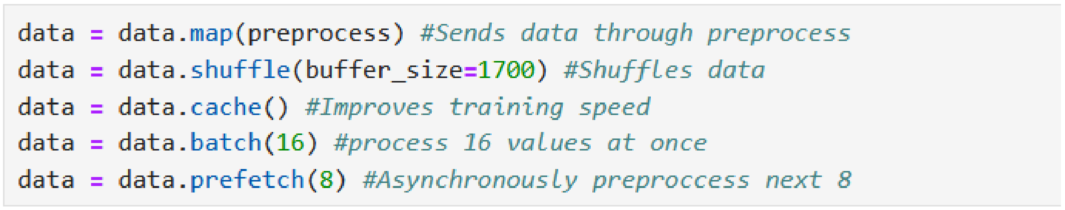


Figure 10.2: Waveform (upper), Spectrogram (lower).

The figure above shows the before and after results of this function on an ambulance siren. The original waveform displays the amplitude over time, whereas the spectrogram shows both frequency and magnitude over time.

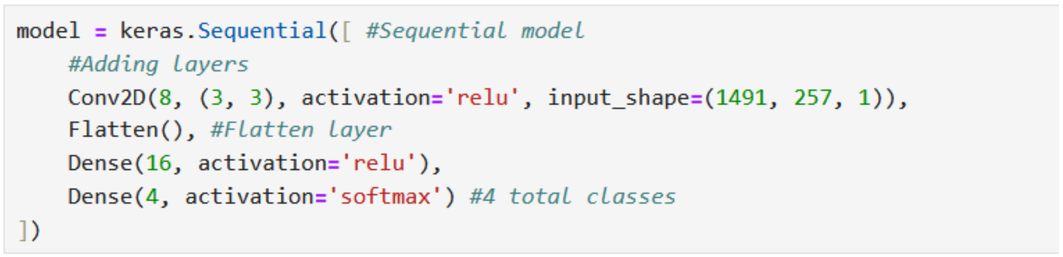
This data is now ready to be ingested into a ML model, the specifics of which have been discussed in-depth already, so only a brief explanation of the code will be shown.



```
data = data.map(preprocess) #Sends data through preprocess
data = data.shuffle(buffer_size=1700) #Shuffles data
data = data.cache() #Improves training speed
data = data.batch(16) #process 16 values at once
data = data.prefetch(8) #Asynchronously preproccess next 8
```

Figure 11.1 (Fig. 9.4 cont'd): Data preparation.

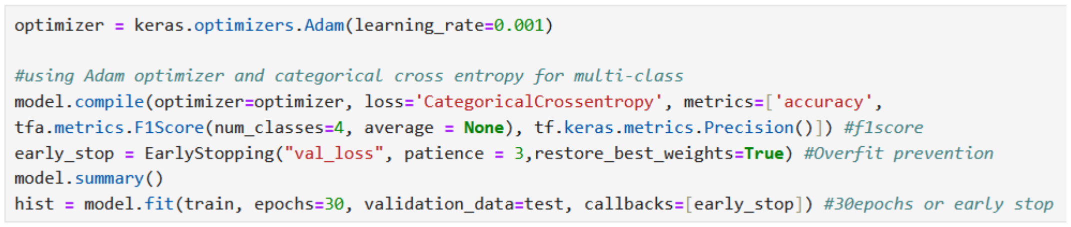
The map function sends data through the preprocess function seen in Figure 9.4, this data is then shuffled to be completely random. The buffer size is set just above the total number of elements to guarantee a thorough randomisation and it also specifies the maximum number of elements to keep in memory while shuffling. Data shuffling is known to enhance the learning model quality and lead to significant statistical gains (Elmahdy and Mohajer, 2020). The shuffled data is then cached, a copy is made in local memory for quick access during training and to prevent the need to reshuffle the data, a high-memory cost procedure. Batching takes advantage of hardware parallelism, based on the model complexity and total number of data points, a batch sizes of 16 is ideal. The prefetch line works behind the scenes to asynchronously fetch and preprocess the next 8 batches whilst the model is concurrently training with the current batch.



```
model = keras.Sequential([ #Sequential model
    #Adding layers
    Conv2D(8, (3, 3), activation='relu', input_shape=(1491, 257, 1)),
    Flatten(), #Flatten layer
    Dense(16, activation='relu'),
    Dense(4, activation='softmax') #4 total classes
])
```

Figure 11.2: Sequential CNN (example structure).

The model is then created using the Keras function 'Sequential,' which groups a linear stack of layers into a model (Tensorflow, 2024b). The specifics of these layers have been thoroughly discussed in Chapter 4 Design & Methodology. This figure shows a basic implementation for a multi-class model, which is obvious from the final dense layer, which has four neurons, meaning four possible classifications. The choice for the ‘softmax’ activation function in the final dense layer guarantees the output of the total class predictions percentages is equal to 1, so the probabilities for each class can be compared against one another.



```
optimizer = keras.optimizers.Adam(learning_rate=0.001)

#using Adam optimizer and categorical cross entropy for multi-class
model.compile(optimizer=optimizer, loss='CategoricalCrossentropy', metrics=['accuracy',
tfa.metrics.F1Score(num_classes=4, average = None), tf.keras.metrics.Precision()]) #f1score
early_stop = EarlyStopping("val_loss", patience = 3,restore_best_weights=True) #Overfit prevention
model.summary()
hist = model.fit(train, epochs=30, validation_data=test, callbacks=[early_stop]) #30epochs or early stop
```

Figure 11.3: Compiling and fitting model.

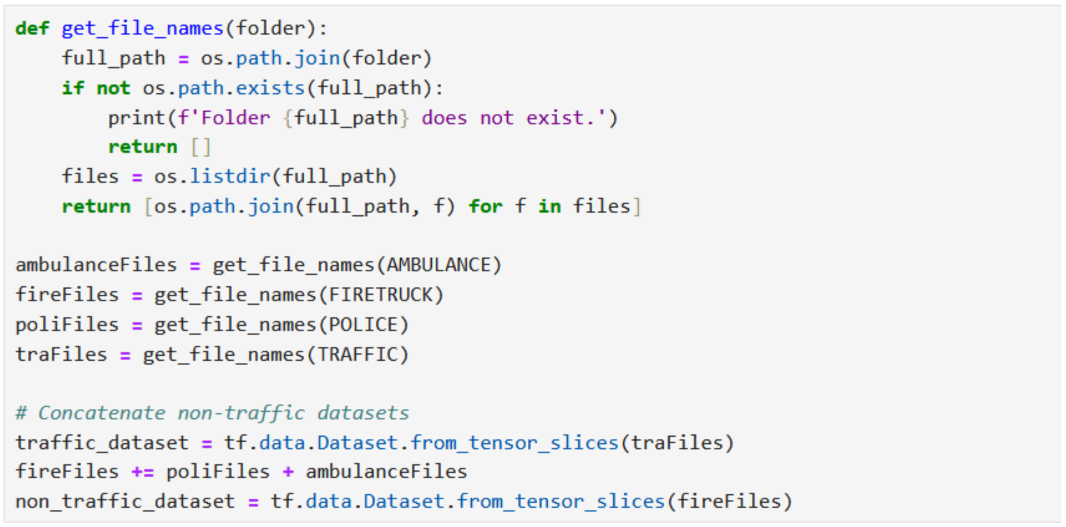
Before fitting the model, an optimiser first needs to be defined. An optimiser algorithm intends to improve a NN’s performance by modifying weights and learning rate to reduce loss (Doshi, 2020). This optimiser uses the Adam algorithm, a stochastic gradient descent method described by (Kingma and Ba, 2014) as computationally efficient and well suited for problems that are large in terms of data or parameters. The learning rate was intentionally set to the default value of 0.001; as a parameter which scales how much model weights should be updated (Keras, 2023e), there are several implications which come from experimenting with this value and considering the complexity of the data, at first it seemed unnecessary to tweak.

The model is then compiled with the Adam optimiser, using both accuracy and F1-Score as the metrics to decipher model performance. Early stopping is a regularisation technique included for the sole purpose of preventing overfitting. When training a large network, there will be a point during training when the model will stop generalising and start learning the statistical noise in the training dataset (Brownlee, 2018). All standard NN architectures are prone to overfitting. While the network seems to get better and better, ie, the error on the training set decreases, at some point during training it actually begins to get worse again (Orr and Klaus-

Robert Müller, 2003, 55, 69). Early stopping is a solution to this: when a specific performance metric begins to degrade and does not improve after a preset number of epochs, training is stopped early. In this case, if the validation sets loss value does not improve after three epochs, the training is cut short, and the model reverts back to the epoch with the best weights. In the final line of code, the model is fitted with the data, completing the pipeline from raw data to CNN.

5.2 One-dimensional binary classifier

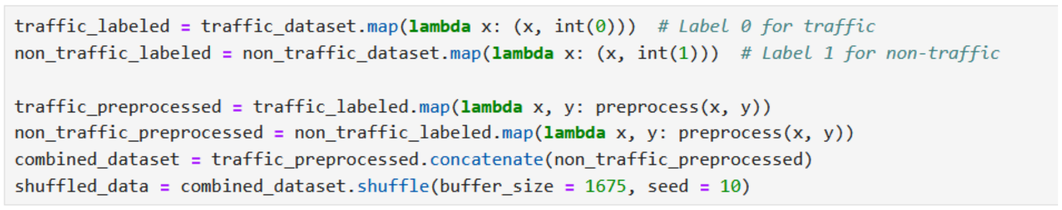
As mentioned in Chapter 4 Design & Methodology, the binary classifier should be capable of outputting class predictions much quicker that the multi-class variant, and to practically implement this, the model layers need to be setup to handle just one convolutional direction rather than two. Updating the binary classifier model to work with a single dimension posed a significant challenge as the data pipeline had been setup for spectrogram ingestion, meaning simply changing model layers was not a viable solution. Therefore, the entire pipeline needed to be updated. This update allowed for other changes, such as making data ingestion a more efficient process.



```
def get_file_names(folder):
    full_path = os.path.join(folder)
    if not os.path.exists(full_path):
        print(f'Folder {full_path} does not exist.')
        return []
    files = os.listdir(full_path)
    return [os.path.join(full_path, f) for f in files]

ambulanceFiles = get_file_names(AMBULANCE)
fireFiles = get_file_names(FIRETRUCK)
poliFiles = get_file_names(POLICE)
traFiles = get_file_names(TRAFFIC)

# Concatenate non-traffic datasets
traffic_dataset = tf.data.Dataset.from_tensor_slices(traFiles)
fireFiles += poliFiles + ambulanceFiles
non_traffic_dataset = tf.data.Dataset.from_tensor_slices(fireFiles)
```

Figure 12.1: File names function.

This setup over the previous delivers greater efficiency when converting the files to a WAV format. Originally, each folder, subfolder, and file were searched through twice, once for defining each class and again for sending data to the function for decoding the files into WAV format. Instead, now the custom function above handles storing each WAV file as a string of its path, which is later used to decode the WAV file each string represents.

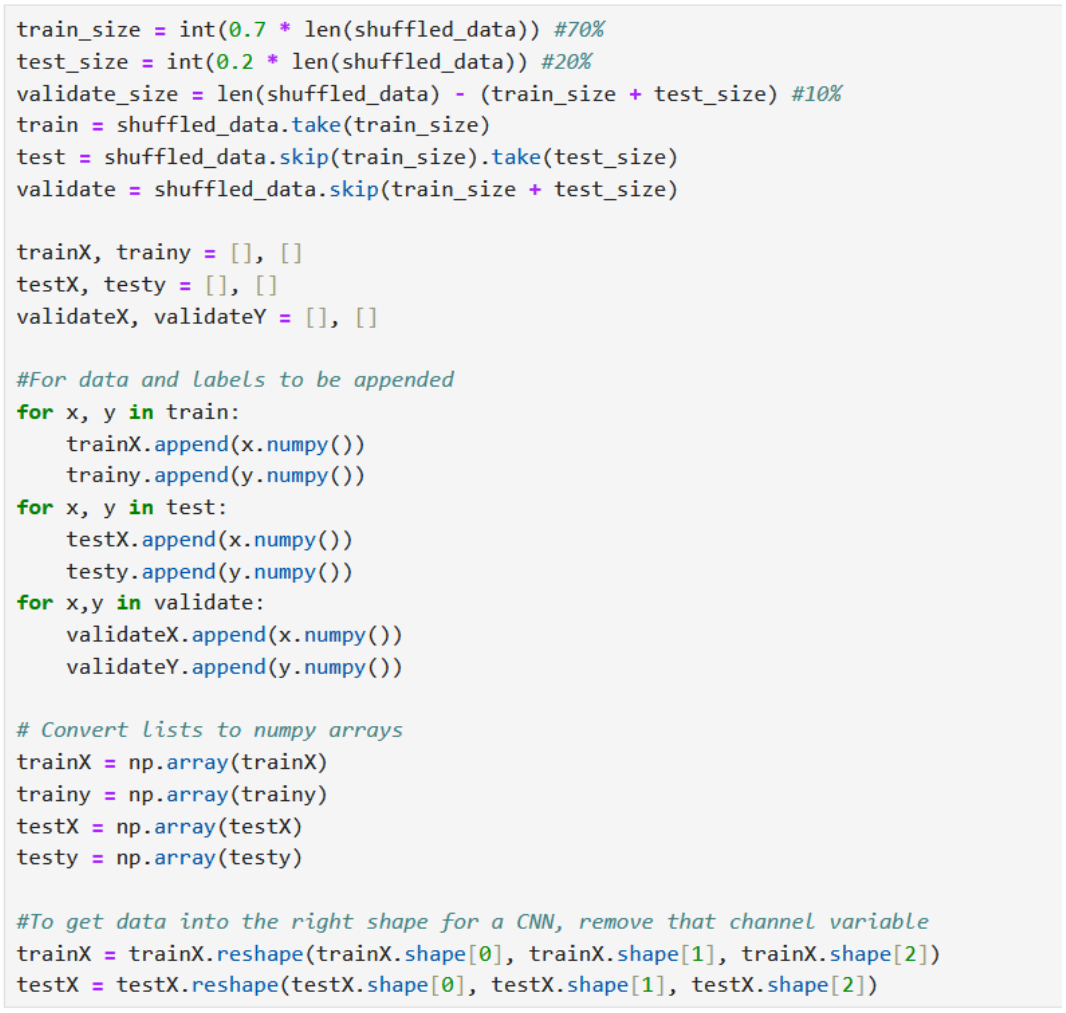


```
traffic_labeled = traffic_dataset.map(lambda x: (x, int(0)))  # Label 0 for traffic
non_traffic_labeled = non_traffic_dataset.map(lambda x: (x, int(1)))  # Label 1 for non-traffic

traffic_preprocessed = traffic_labeled.map(lambda x, y: preprocess(x, y))
non_traffic_preprocessed = non_traffic_labeled.map(lambda x, y: preprocess(x, y))
combined_dataset = traffic_preprocessed.concatenate(non_traffic_preprocessed)
shuffled_data = combined_dataset.shuffle(buffer_size = 1675, seed = 10)
```

Figure 12.2: Class labelling and concatenating.

These string files are labelled either 0 or 1, where 1 indicates a siren sound, and this is sent to the preprocess function which was discussed earlier. The file names and labels are then concatenated into a single dataset and shuffled.

The shuffled data is then used to create a test and train set each with an X and y representing value and label correspondingly. This is a more traditional way of separating data and made debugging far easier. Interestingly, this completely avoids the use of batch and prefetch functions, which are deprecated, and therefore best practice to avoid for future use cases. The values of trainX and testX are reshaped to remove the channel parameter, which is present in the spectrogram variant, but unnecessary for the one-dimensional binary model.



```python
train_size = int(0.7 * len(shuffled_data)) #70%
test_size = int(0.2 * len(shuffled_data)) #20%
validate_size = len(shuffled_data) - (train_size + test_size) #10%
train = shuffled_data.take(train_size)
test = shuffled_data.skip(train_size).take(test_size)
validate = shuffled_data.skip(train_size + test_size)

trainX, trainy = [], []
testX, testy = [], []
validateX, validateY = [], []

#For data and labels to be appended
for x, y in train:
    trainX.append(x.numpy())
    trainy.append(y.numpy())
for x, y in test:
    testX.append(x.numpy())
    testy.append(y.numpy())
for x,y in validate:
    validateX.append(x.numpy())
    validateY.append(y.numpy())

# Convert lists to numpy arrays
trainX = np.array(trainX)
trainy = np.array(trainy)
testX = np.array(testX)
testy = np.array(testy)

#To get data into the right shape for a CNN, remove that channel variable
trainX = trainX.reshape(trainX.shape[0], trainX.shape[1], trainX.shape[2])
testX = testX.reshape(testX.shape[0], testX.shape[1], testX.shape[2])
```

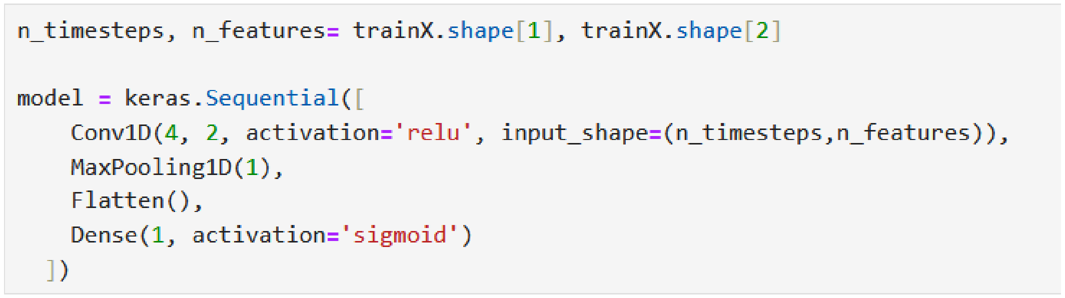
Figure 12.3: Creating test, train, validate sets.

	Before Reshape	**After Reshape**
trainX shape	(1172,1491,257,1)	(1172,1491,257)
testX shape	(335,1491,257,1)	(355,1491,257)

Table 5: Reshape operation.

Where:
1172 = train dataset size
335 = test dataset size
1491 = number of timesteps
257 = number of features
1 = number of channels

After reshaping data into a suitable format, a machine learning model similar to the original binary classifier is created, but with far fewer layers.



```
n_timesteps, n_features= trainX.shape[1], trainX.shape[2]

model = keras.Sequential([
    Conv1D(4, 2, activation='relu', input_shape=(n_timesteps,n_features)),
    MaxPooling1D(1),
    Flatten(),
    Dense(1, activation='sigmoid')
  ])
```

Figure 12.4: Small sequential model for binary classification

Because only one-dimension of audio data is being processed, with too many layers or filters, the model can overfit by memorising patterns unique to the training data. This smaller model has only 8,000 total trainable parameters, which in comparison to the original model's 2.9 million total trainable parameters, has significantly less data to learn from. The initial results of this model reflected that simply reducing the complexity was not enough to create a useable model.

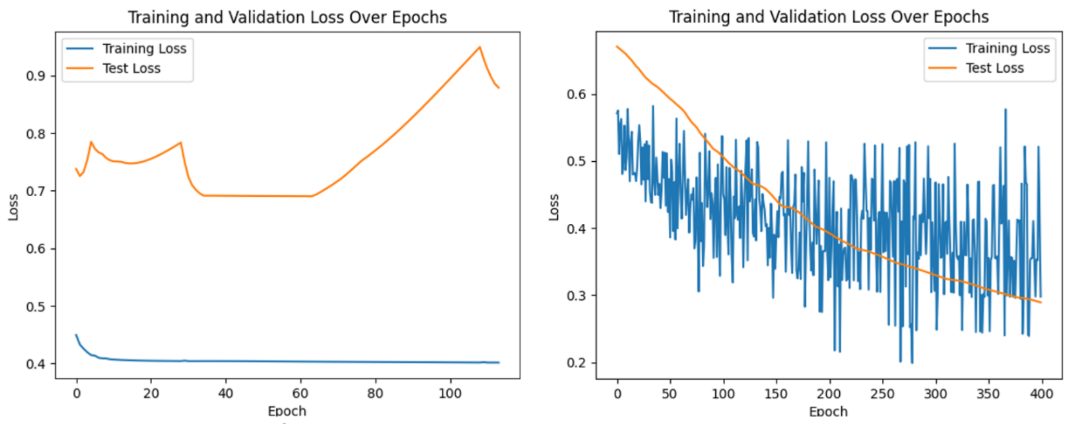


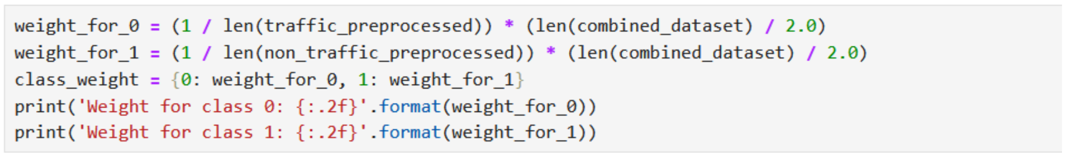
Figure 13: (Left) data generalisation. (Right), erroneous training loss.

The left graph shows a generalisation of the training dataset, with a quick plateau for the loss metric. The loss metric for the test dataset interestingly increases after around 60 epochs. The jumps between increasing and decreasing loss metrics from epoch 0-40 can be attributed to experimentation with including class weights with the model fitting process, where the higher weights to the minority class lead to a higher loss as the model prioritises learning from scarcer examples.

Class weights have been calculated using code from (Tensorflow, 2024c), which should lead the model to pay more attention to examples from an under-represented class, in this case the traffic class.

Class	**Weight**
Siren	1.99
Traffic	0.67

Table 6: Weights for binary classifier.



```
weight_for_0 = (1 / len(traffic_preprocessed)) * (len(combined_dataset) / 2.0)
weight_for_1 = (1 / len(non_traffic_preprocessed)) * (len(combined_dataset) / 2.0)
class_weight = {0: weight_for_0, 1: weight_for_1}
print('Weight for class 0: {:.2f}'.format(weight_for_0))
print('Weight for class 1: {:.2f}'.format(weight_for_1))
```

Figure 14 (Fig. 12.4 cont'd): Calculating class weights.

Several issues believed to have stemmed from the significant class imbalance actually were found to be impacted more by the learning rate of the Adam optimiser. Altering the default value of 1e-3 (0.001) to 1e-4 (0.0001), showed far more promising results.

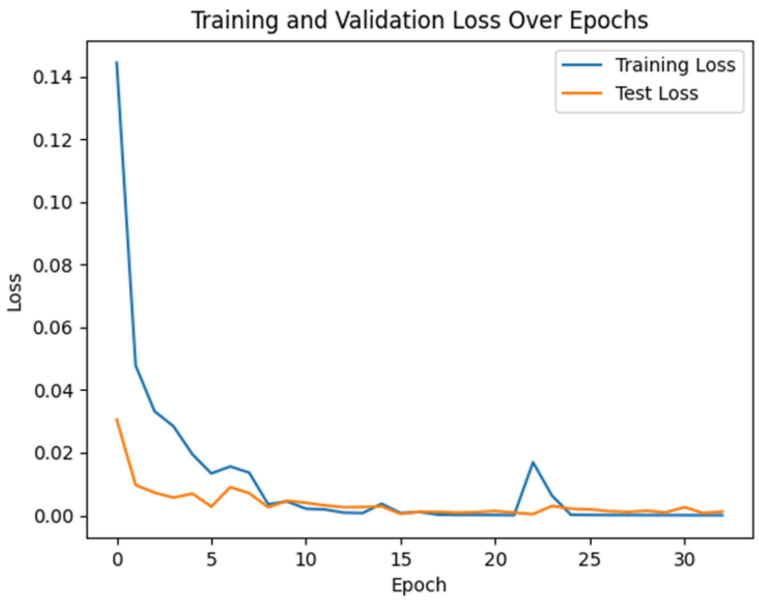


Figure 15: 1D binary classifier loss metrics.

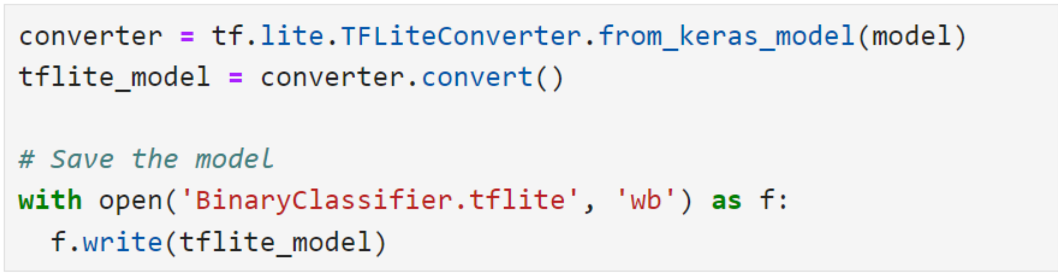
A new layer was also experimented with for the one-dimensional classifier, the dropout layer.

- Dropout: The dropout layer randomly sets input units to 0 with a preset frequency at each step during training to help prevent overfitting (Keras, 2023f). Essentially, the dropout layer will randomly drop neurons to prevent the model adapting, and forcing it to learn more robust features in the data. The user can choose how common this occurs, through a preset value from 0-1.

Ultimately, a dropout layer was not included in the final binary model as it led to a slight reduction in performance. Despite this, the overall changes did prompt an update to the original data pipeline. These changes led to significant improvements in kernel processing speeds, from what originally took ~130 seconds, down to under a minute ~58 seconds.

5.3 TensorFlow lite

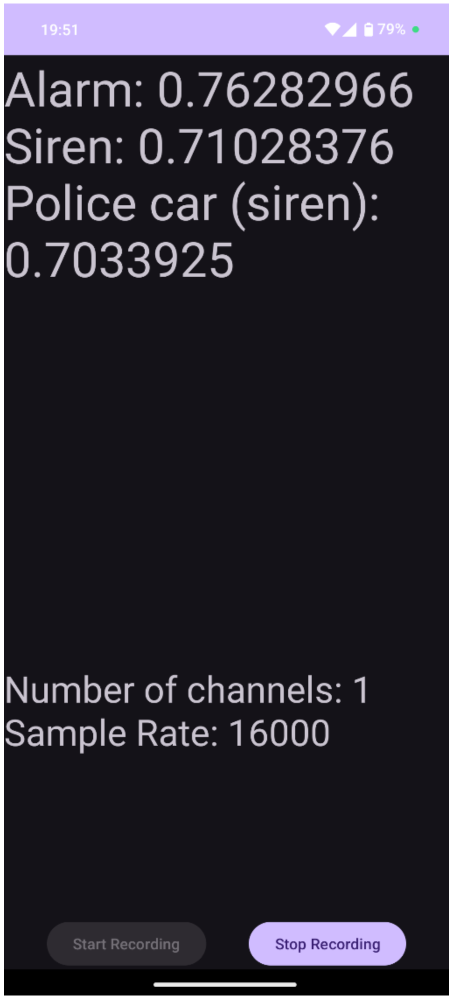
To install a TensorFlow model on a smartphone, it is necessary to convert it to a less sizable format which can more easily be downloaded and run. TensorFlow recommends TensorFlow Lite (TFLite), a mobile library for deploying models on mobile and other edge devices (TensorFlow, 2019). Converting a Keras Sequential model to a TFLite format proved an easy step, requiring only a few lines of code provided in the documentation.



```python
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the model
with open('BinaryClassifier.tflite', 'wb') as f:
  f.write(tflite_model)
```

Figure 16: Keras model to TFLite conversion (Tensorflow, 2024d).

This conversion converted what would be a 200KB folder of 5 files to a single file of just 35KB. From here, Android Studio, an IDE for Android app development was used. Because of a lack of familiarity with smartphone app development, particularly with Android Studio, a tutorial was followed to create a basic template app with a model running. The app was built using Java, and followed a tutorial by (The Mobile Dev, 2021), showing how to import a TFLite model and create a recording feature using a smartphones built-in microphone. Below is an image of an app created following this tutorial, running on Android 13, to show this app can work with an ML model, the YAMNet audio classification model, a deep neural network with 521 classes (Tensorflow, 2024e) has been used.

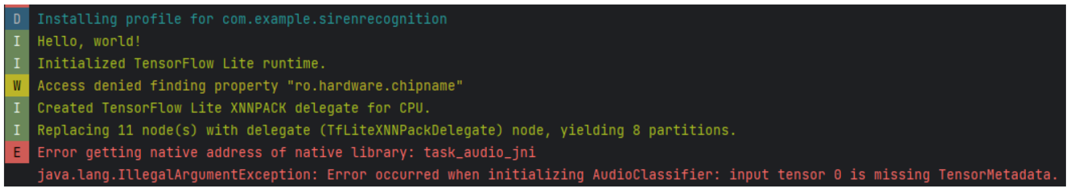

Alarm: 0.76282966
Siren: 0.71028376
Police car (siren): 0.7033925

Number of channels: 1
Sample Rate: 16000

Figure 17.1: Android application running YaMNet model.

From the above figure it can be noted that the application created has functionality for stopping and starting recording. On starting the recording, classifications are outputted in text format, followed by a percentage reflecting the confidence of the classification. In this example, a police siren was played close to the speaker, and the YaMNet model detected either an alarm, siren or police siren.

Sadly, adding the siren detection TFLite models to the android application was not so simple. Complications occurred during runtime, which when analysing the Android device's logcat panel, was found to be an issue regarding the TFLite model's metadata.



```
D  Installing profile for com.example.sirenrecognition
I  Hello, world!
I  Initialized TensorFlow Lite runtime.
W  Access denied finding property "ro.hardware.chipname"
I  Created TensorFlow Lite XNNPACK delegate for CPU.
I  Replacing 11 node(s) with delegate (TfLiteXNNPackDelegate) node, yielding 8 partitions.
E  Error getting native address of native library: task_audio_jni
   java.lang.IllegalArgumentException: Error occurred when initializing AudioClassifier: input tensor 0 is missing TensorMetadata.
```

Figure 17.2: Android device logcat.

After testing with different smartphones and following documentation for adding metadata to TFLite models (Tensorflow, 2024f), it was still not possible to get past the error "input tensor 0 is missing metadata." Eventually, when testing needed to take place and trying to solve this error became a hinderance to the rest of this project, it was agreed with the project supervisor to nevertheless move forward. Fortunately, testing could still continue despite not getting the model on a smartphone device, but this does mean some modifications to the testing procedure needed to be made. These modifications and the overall testing procedure will be mentioned in-depth in Chapter 6 Results and Discussion.

Chapter 6

Results & Discussion

6.1 Recording data

Data was recorded in the City of Lincoln the week commencing 26th February 2024 using the recording feature of the laptop artefact referenced in Figure 8.2. This laptop was connected to an external microphone, of which was attatched to the boot of a car.

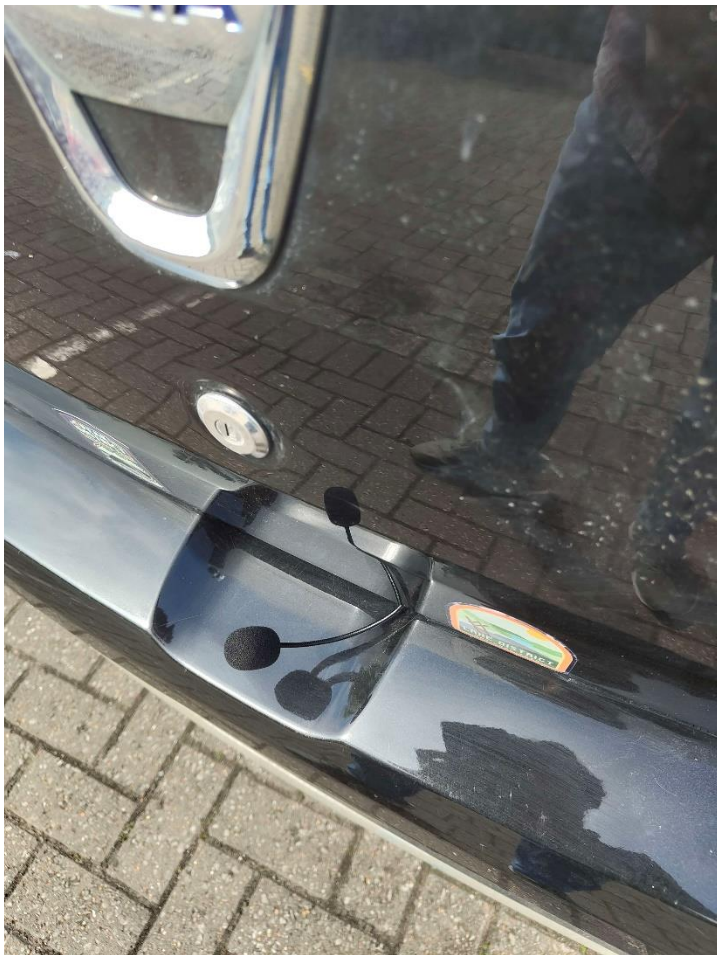


Figure 18: Microphone outside the vehicle.

To get as many sample ESV siren sounds as possible, recordings were taken on roads near Lincoln hospital and Lincoln South Park's ambulance, fire & police station.

The functionality of the multi-class model to classify each 3 second clip with a label was used in partnership with the recording feature seen in Figure 8.3, storing them in one of four folders, each corresponding to a class. This functionality aligns with one of the project's objectives: to save and store new data points to the dataset for future training. Additionally, maintaining separate folders for each class mimics the original dataset's structure, streamlining the process of adding new data without manual handling.

Although plenty of true negative (traffic) data was collected from in-vehicle testing, it proved challenging to capture an adequate amount of siren sounds. To fulfil this shortfall, some ESV siren sounds were recorded on a smartphone around Lincoln, on a myriad of days between January and March 2024. As these recordings do not include the traffic and environmental noise associated with recording on the outside of a moving vehicle, they have been modified to include additional noise.

In total, the final test set consists of:

- 8 Ambulance sirens
- 10 Police sirens
- 8 Fire engine sirens
- 40 Traffic clips

Interestingly, the recorded traffic noise is far quieter than that of the original dataset traffic. The audio sample in the figure below was taken on a dual carriageway, at around 50mph, which is why the considerable reduction in environmental noise is surprising.

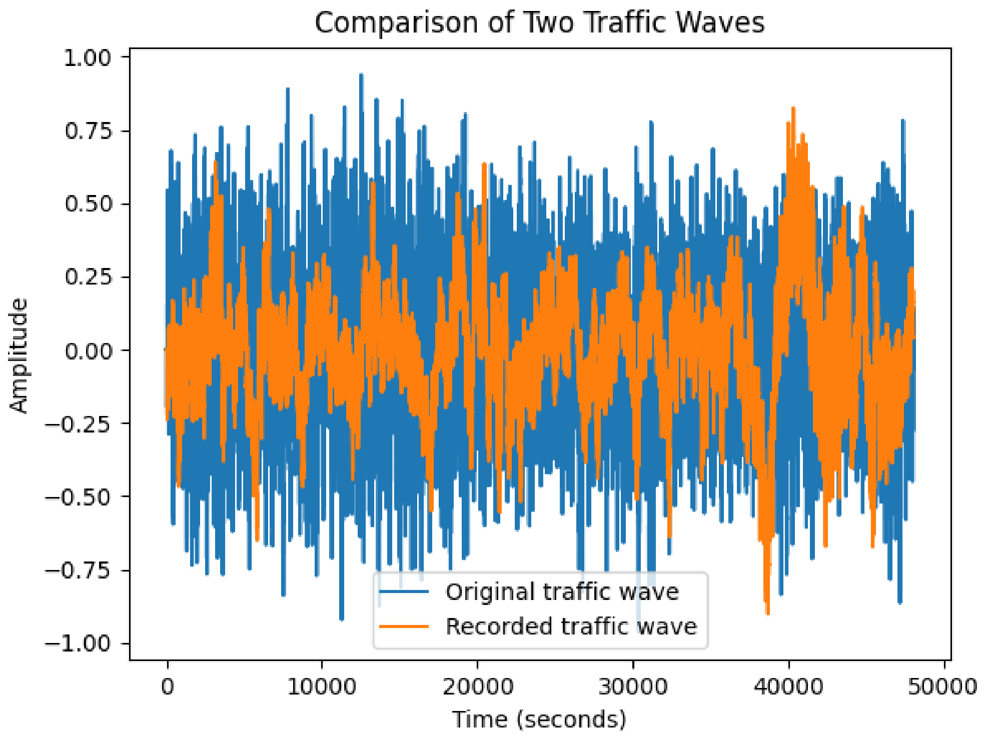


Figure 19: Comparison of traffic noise amplitude.

This may be a result of the microphone's strategic placement on the rear of the vehicle minimising the impact of wind noise, in conjunction with the pop-filter reducing the amount of noise reaching the microphone. The reasoning behind placing the microphone at the back of the vehicle was that a driver is far less likely to see an ESV approaching from the rear; the added benefit of decreased environmental noise was not expected, but a fortunate bonus.

6.2 Model testing

The siren recordings were interspersed between traffic recordings and concatenated into a single 3 minute audio file. To ensure fairness, both models were played the same dataset a total of three times, at a noise level consistent with that of the original recordings. A minimum confidence level of 70% was required for a class prediction to be considered. If multiple classes exceed this threshold, the one with the highest confidence level will be selected. The average values of the three tests were then placed into confusion matrices to graph their performance.

Binary Classifier	True Siren	True Traffic
Predicted Siren	19	2
Predicted Traffic	7	38

Table 7.1: Binary-classifier confusion matrix.

Multi-Class Classifier	True Ambulance	True Police	True Fire-Engine	True Traffic
Predicted Ambulance	6	0	1	5
Predicted Police	0	8	0	0
Predicted Fire-Engine	2	0	6	6
Predicted Traffic	0	2	1	29

Table 7.2: Multi-Class classifier confusion matrix.

6.3 Reviewing model results

Overall, these real-world results are promising and show that in the majority of cases, both models are able to classify the correct class. Some misclassifications were expected, and both models seemed to struggle with the same data points. For example, both models misclassified some traffic data as sirens; this incident occurred with audio clips where the microphone tapped the boot of the car, creating a vibration sound. Instances where a siren was misclassified as traffic data could be solved by simply increasing the volume of these clips, providing evidence that volume plays a key factor in detection.

6.4 Edge cases

In situations where there are more than one siren, the multi-class classifier always favoured the ambulance class, never predicting two different siren types with confidence levels above a 70% threshold.

Each model was also tested with an assortment of siren-esque sounds recorded around Lincoln, such as: car horns, car alarms, level crossing gate alarms and pedestrian crossing alarms. Additionally, building alarm sounds were supplemented from online video sources. None of these recordings prompted a classification other than traffic noise, which provides a level of confidence in the robustness of the models.

6.5 Performance metrics

The confusion matrices values were then converted to performance metrics. The below tables display the Accuracy, Precision, Recall and F1-Score of each model's individual classes. Of course, accuracy will be the same value for each class since it is calculated by dividing correct predictions from the total predictions.

Binary Classifier	Accuracy	Precision	Recall	F1-Score
Siren	0.8636	0.9048	0.7308	0.8085
Traffic	0.8636	0.8444	0.9500	0.8941

Table 8.1: Binary classifier performance metrics.

Multi-Class Classifier	Accuracy	Precision	Recall	F1-Score
Ambulance	0.7424	0.5	0.7500	0.600
Police	0.7424	1	0.800	0.8889
Fire Engine	0.7424	0.4286	0.7500	0.5455
Traffic	0.7424	0.9063	0.7250	0.8056

Table 8.2: Multi-class classifier performance metrics.

The results of real-time testing show a decrease in performance across all metrics, aligning with findings in the literature (Ramirez et al., 2022) which also found that real-world testing decreased model performance. From these metrics, it can be said that in most cases, the binary classifier was more effective at recognising siren sounds. This is likely as the difference between traffic noise and a siren is significant enough to make classification simpler.

By a significant margin, a fire engine siren was the hardest class to classify, with more misclassifications than correct detections. A substantial factor was that traffic noise was sometimes mistaken for fire engine sirens. It is worth noting that this occurred mostly in audio clips with slight vibrations from the microphone tapping the boot of the vehicle. Nevertheless, to maintain impartiality this audio was not removed from the dataset. Overall these results are promising, ideally more data points would be gathered, but considering that real-world data was required, data quality was favoured over quantity.

Chapter 7

Conclusion

7.1 Development

The ML models and pipeline created provide a strong foundation for further research and experimentation. With more time, more hyperparameters could be tested, providing marginally better results. These could be tested with only a need for a fundamental understanding of their function, using search methods such as the grid search function from (ScikitLearn, 2019b) which finds the optimal hyperparameters for a model through cross-validation.

If the models were to be developed anew, more effort would go into fortifying their data pipeline to save the wasted time spent renewing it for the binary classification model. One of the key issues with the binary classification model was the learning rate, which ultimately ended up being a significant determinant for the associated performance metrics (F1-Score and Loss). Learning rate controls the speed that a model can learn by a fixed amount per epoch. As a constant value, a model can easily be under or overfed with data resulting in underfitting or overfitting respectively. An adaptive learning rate is the idea of using a step size varying with the model (Defazio et al., 2023), which means the models performance influences a change in learning rate. For example, if the performance plateaus, the learning rate may dynamically decrease by an order of magnitude.

Despite the literature suggesting that a CNN approach is favourable, an effort to use more traditional ML algorithms such as k-nearest neighbours or support vector machines would prove useful to compare and contrast effectiveness in real-time classification. Some other ML related changes which could have been further researched regard experimenting with the parameter values of each Sequential model layer. Particularly, the kernel size parameter for the two-dimensional and one-dimensional convolutional layers. As the primary feature extractors responsible for finding the patterns which help to classify data, it seems vital to spend time ensuring a balance is found between computational cost and performance.

7.2 Testing

Testing the models proved to be far more challenging than anticipated due to the time consuming process of collecting real-world examples and the scarcity of ESVs. Ideally, samples gathered could be augmented to create new data points, providing a more comprehensive set of data for testing. More thought could have also been put into the location of the microphone. Despite the microphones placement on the back of the car proving to be effective, alternative placements could still be more beneficial. Recording the volume for environmental noise for each possible placement and selecting the one with the lowest value would identify the ideal microphone location. Furthermore, a larger microphone may prove more effective at detecting sirens from further away, as it was found that volume plays a key factor in siren detection and recognition. In spite of that, it should be first tested whether including quieter clips of siren sounds in the dataset would improve detection. This would be a cost-free alternative to buying a new microphone and could provide a more generalised model which better understands the patterns of sirens, irrespective of volume.

This segways nicely into the project's objective to implement a feature to save and store new data points to the dataset for future training. Although this has been implemented and data is ready to be included in the data pipeline, it would have been interesting to see if the data captured could be used to improve the models.

7.3 Design

One of the challenges of this project was balancing time for creating and tuning the ML models with designing a system for the target demographic. Beyond improvements to the ML aspect of the project, there is a desire to research the intricacies of designing a user interface for a HI driver. Currently, the mobile version uses a text output of a predicted label, followed by a percentage reflecting the confidence of the classification. The ideal approach would be to notify the user of an approaching ESV using prominent visual cues such as bright bold text, animations, or a light-up screen, whilst also minimising distractions from the road. Developing a solution which focuses on safety and unobtrusive alerts seems a fascinating challenge and ties directly into possible future works.

7.4 Future works

Although a smartphone version utilising one of the created models has not been implemented, the basic application found in Figure 17.1 provides strong evidence that such an implementation is feasible; and it could then be published to an app store for public use. Despite this caveat, it was mentioned in Chapter 4 Design & Methodology that a more commercial use case would be to use these models as part of a vehicles entertainment system, using the screen to communicate to the driver. This opens up several opportunities to increase the scope, including omni-directional speakers, predicting the direction an ESV approaches from, and user-friendly alerts of a siren being heard. The possible future works can all be boiled down to creating a useable final artefact that a HI driver could interact with, without the need to set up the system themselves.

Overall this project has been successful, the attained results demonstrate that a ML approach to siren detection and recognition is functional in a real-world environment and displays where future work could improve the flaws of the models created. The four primary objectives have all been completed, meeting their outlined scope. A broad range of ML techniques have been utilised, coupled with efficient data handling to create a streamlined data pipeline. The ML models have been custom designed to fit their use case, and have demonstrated in both synthetic and real-world testing their effectiveness at detecting and recognising ESV sirens.

References

Alkhalifah, T., Wang, H. and Ovcharenko, O. (2021) MLReal: Bridging the gap between training on synthetic data and real data applications in machine learning. *King Abdullah University of Science and Technology Repository (King Abdullah University of Science and Technology)*, 2021(1) 1-5. Available from [accessed 12 February 2024].

AssemblyAI (2022) *Build your own real-time voice command recognition model with TensorFlow* [video]. Available from https://www.youtube.com/watch?v=m-JzldXm9bQ [accessed 6 May 2024].

British Academy of Audiology (2023) *Facts about Hearing Loss and Deafness* Available from https://www.baaudiology.org/about/media-centre/facts-about-hearing-loss-and-deafness/ [accessed 24 April 2024].

Brook, T. (2024) *An Ultimate Guide To The Police Siren* Available from https://www.jointhecops.co.uk/guide-to-the-police-siren/ [accessed 14 March 2024].

Brown, L.H., Whitney, C.L., Hunt, R.C., Addario, M. and Hogue, T. (2000) Do warning lights and sirens reduce ambulance response times? *Prehospital Emergency Care*, 4(1) 70-74. Available from https://www.sciencedirect.com/science/article/pii/S1090312700700797.

Brownlee, J. (2018) *A Gentle Introduction to Early Stopping to Avoid Overtraining Neural Networks* Available from https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/ [accessed 21 April 2024].

Castiglioni, P. (2005b) Zero Padding. *Encyclopedia of Biostatistics*, Available from https://10.0.3.234/0470011815.b2a12087 [accessed 28 March 2024].

Chin, C.-L., Lin, C.-C., Wang, J.-W., Chin, W.-C., Chen, Y.-H., Chang, S.-W., Huang, P.-C., Zhu, X., Hsu, Y.-L. and Liu, S.-H. (2023) A Wearable Assistant Device for the Hearing Impaired to Recognize Emergency Vehicle Sirens with Edge Computing. *Sensors*, 23(17) 7454. Available from https://www.mdpi.com/1424-8220/23/17/7454 [accessed 17 February 2024].

Databricks (2019) *What is a Convolutional Layer?* Available from https://www.databricks.com/glossary/convolutional-layer [accessed 7 March 2024].

De Lorenzo, R.A. and Eilers, M.A. (1991) Lights and siren: A review of emergency vehicle warning systems. *Annals of Emergency Medicine*, 20(12) 1331–1335. Available from [accessed 17 February 2024].

Defazio, A., Cutkosky, A., Mehta, H. and Mishchenko, K. (2023) When, Why and How Much? Adaptive Learning Rate Scheduling by Refinement. *arXiv (Cornell University)*, Available from https://doi.org/10.48550/arXiv.2310.07831 [accessed 30 April 2024].

Dettmers, T. (2015) *Deep Learning in a Nutshell: Core Concepts* Available from https://developer.nvidia.com/blog/deep-learning-nutshell-core-concepts/ [accessed 4 March 2024].

Doshi, S. (2020) *Various Optimization Algorithms For Training Neural Network* Available from https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6 [accessed 21 April 2024].

Elmahdy, A.M. and Mohajer, S. (2020) On the Fundamental Limits of Coded Data Shuffling for Distributed Machine Learning. *IEEE Transactions on Information Theory*, 66(5) 3098–3131. Available from https://doi.org/10.1109/TIT.2020.2964547 [accessed 19 April 2024].

Fränti, P. and Mariescu-Istodor, R. (2023) Soft Precision and Recall. *Pattern Recognition Letters*, 167 115-121. Available from [accessed 8 February 2024].

Government Digital Service (2012) *Driving eyesight rules* Available from https://www.gov.uk/driving-eyesight-rules [accessed 24 April 2024].

Harley, J. (2022) *EEE 5502: Foundations of Digital Signal Processing* Available from https://smartdata.ece.ufl.edu/eee5502/lecture.html?lecture=14 [accessed 16 April 2024].

Harrison, P. (2023) *Driving with Hearing Loss* Available from https://www.hearingaid.org.uk/hearing-loss-awareness/driving-with-hearing-loss [accessed 24 April 2024].

Harwood, S. (2023) *Siren Detection and Recognition for Deaf and Hearing-Impaired Individuals: A Machine Learning Approach* CMP3753: Project. The University of Lincoln. Unpublished essay

Harwood, S. (2024) *Interim Report - Siren Detection and Recognition for Hearing-Impaired Individuals: A Machine Learning Approach* CMP3753: Project. The University of Lincoln. Unpublished essay

Hersh, M., Ohene-Djan, J. and Naqvi, S. (2010) Investigating Road Safety Issues and Deaf People in the United Kingdom: An Empirical Study and

Recommendations for Good Practice. *Journal of Prevention & Intervention in the Community*, 38(4) 290–305. Available from https://doi.org/10.1080/10852352.2010.509021 [accessed 24 April 2024].

Ho, J. and Casey, B. (1998) Time Saved With Use of Emergency Warning Lights and Sirens During Response to Requests for Emergency Medical Aid in an Urban Environment. *Annals of Emergency Medicine*, 32(5) 585-588. Available from https://doi.org/10.1016/S0196-0644(98)70037-X [accessed 3 May 2024].

Ho, J. and Lindquist, M. (2001) TIME SAVED WITH THE USE OF EMERGENCY WARNING LIGHTS AND S IREN WHILE RESPONDING TO REQUESTS FOR EMERGENCY MEDICAL AID IN A RURAL ENVIRONMENT. *Prehospital Emergency Care*, 5(2) 159–162. Available from https://doi.org/10.1080/10903120190940056.

Hubel, D.H. and Wiesel, T.N. (1962) Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1) 106–154. Available from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1359523/ [accessed 24 April 2024].

Hunt, R.C., Brown, L.H., Cabinum, E.S., Whitley, T.W., Prasad, N.H., Owens, C.F. and Mayo, C.E. (1995) Is Ambulance Transport Time With Lights and Siren Faster Than That Without? *Annals of Emergency Medicine*, 25(4) 507–511. Available from https://doi.org/10.1016/S0196-0644(95)70267-9 [accessed 3 May 2024].

Ikeda, H. and Minami, S. (2020) Survey on the recognition of emergency vehicles by hearing-impaired drivers. *Journal of Global Tourism Research*, 5(2) 173-176.

Jeon, H., Jung, Y., Lee, S. and Jung, Y. (2020) Area-Efficient Short-Time Fourier Transform Processor for Time–Frequency Analysis of Non-Stationary Signals. *Applied Sciences*, 10(20) 7208. Available from https://www.mdpi.com/2076-3417/10/20/7208 [accessed 28 March 2024].

Julien, A., Virginie, P., Mihcele, M. and Andre, C. (2010) Implication of the cognitive functions and personality traits on tactical compensation among older drivers: A gender comparison. In: *TRANSED 2010: 12th International Conference on Mobility and Transport for Elderly and Disabled Persons*. 2010 Hong Kong Society for Rehabilitation,.

Kamali, A. (2023) *An Introduction to Time Series Padding Techniques in Python* Available from https://medium.com/full-metal-data-scientist/an-introduction-to-time-series-padding-techniques-in-python-b7307a2eba87 [accessed 8 February 2024].

Keras (2023a) *Keras documentation: The Sequential class* Available from https://keras.io/api/models/sequential/ [accessed 22 February 2024].

Keras (2023b) *Keras documentation: Conv2D layer* Available from https://keras.io/api/layers/convolution_layers/convolution2d/ [accessed 7 March 2024].

Keras (2023c) *Keras documentation: MaxPooling2D layer* Available from https://keras.io/api/layers/pooling_layers/max_pooling2d/ [accessed 7 March 2024].

Keras (2023d) *Keras documentation: Dense layer* Available from https://keras.io/api/layers/core_layers/dense/ [accessed 23 March 2024].

Keras (2023e) *Keras documentation: Optimizers* Available from https://keras.io/api/optimizers/ [accessed 21 April 2024].

Keras (2023f) *Keras documentation: Dropout layer* Available from https://keras.io/api/layers/regularization_layers/dropout/ [accessed 24 April 2024].

Kingma, D.P. and Ba, J. (2014) Adam: A Method for Stochastic Optimization. In: *3rd International Conference for Learning Representations*. 2014.

Lis, O. (2024) *Driving with Hearing Aids: A Journey of Sound and Safety* Available from https://audiologycentral.com/driving-with-hearing-aids-a-journey-of-sound-and-safety/ [accessed 24 April 2024].

Marques-Baptista, A., Ohman-Strickland, P., Baldino, K.T., Prasto, M. and Merlin, M.A. (2010) Utilization of Warning Lights and Siren Based on Hospital Time-Critical Interventions. *Prehospital and Disaster Medicine*, 25(4) 335, 339. Available from https://10.0.3.249/S1049023X0000830X [accessed 3 May 2024].

Marchegiani, L. and Newman, P. (2022) Listening for Sirens: Locating and Classifying Acoustic Alarms in City Scenes. *IEEE Transactions on Intelligent Transportation Systems*, 23(10) 1–10. Available from https://ieeexplore.ieee.org/document/9737390 [accessed 13 February 2024].

Matplotlib (2012) *Matplotlib: Python plotting — Matplotlib 3.1.1 documentation* Available from https://matplotlib.org/ [accessed 9 March 2024].

Muñiz, A. (2020) *Driving with hearing impairments* Available from https://www.fundacionmapfre.org/en/education-outreach/road-safety/mobility-safe-health/did-you-know/driving-hearing-impairments/ [accessed 24 April 2024].

Murray, B. and Kue, R. (2017) The Use of Emergency Lights and Sirens by Ambulances and Their Effect on Patient Outcomes and Public Safety: A Comprehensive Review of the Literature. *Prehospital and Disaster Medicine*, 32(2) 209–216. Available from https://10.0.3.249/S1049023X16001503 [accessed 3 May 2024].

Nasser Kehtarnavaz (2008) *Digital Signal Processing System Design*. 2nd edition. Elsevier BV.

Orr, G.B. and Klaus-Robert Müller (2003) *Neural Networks: Tricks of the Trade*. Springer.

Picard, M., Girard, S.A., Courteau, M., Leroux, T., Larocque, R., Turcotte, F., Lavoie, M. and Simard, M. (2008) Could Driving Safety be Compromised by Noise Exposure at Work and Noise-Induced Hearing Loss? *Traffic Injury Prevention*, 9(5) 489–499. Available from [accessed 20 February 2024].

Potdar, K., S., T. and D., C. (2017) A Comparative Study of Categorical Variable Encoding Techniques for Neural Network Classifiers. *International Journal of Computer Applications*, 175(4) 7–9.

Potter, R.C., Fidell, S.A., Myles, M.M. and Keast, D.N. (1977) *Effectiveness of Audible Warning Devices on Emergency Vehicles.* Available from https://rosap.ntl.bts.gov/view/dot/9457 [accessed 17 February 2024].

Ramirez, A.E., Donati, E. and Chousidis, C. (2022) A siren identification system using deep learning to aid hearing-impaired people. *Engineering Applications of Artificial Intelligence*, 114. Available from https://doi.org/10.1016/j.engappai.2022.105000 [accessed 24 April 2024].

Renotte, N. (2022) *Build a Deep Audio Classifier with Python and Tensorflow* [video]. Available from https://www.youtube.com/watch?v=ZLIPkmmDJAc [accessed 16 April 2024].

Říha, J. (2023) *PionEar: Making Roads Safer for Deaf Drivers* Available from https://hackaday.io/project/191087-pionear-making-roads-safer-for-deaf-drivers [accessed 17 February 2024].

RNID (2023) *Prevalence of deafness and hearing loss* Available from https://rnid.org.uk/get-involved/research-and-policy/facts-and-figures/prevalence-of-deafness-and-hearing-loss/ [accessed 29 April 2024].

Schröder, J., Goetze, S., Grützmacher, V. and Anemüller, J. (2013) *Automatic acoustic siren detection in traffic noise by part-based models* Available from https://ieeexplore.ieee.org/document/6637696 [accessed 24 April 2024].

ScikitLearn (2019a) *sklearn.preprocessing.OneHotEncoder — scikit-learn 0.22 documentation* Available from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html [accessed 27 March 2024].

ScikitLearn (2019b) *sklearn.model_selection.GridSearchCV — scikit-learn 0.22 Documentation* Available from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html [accessed 30 April 2024].

Shah, A. and Singh, A. (2023) sireNNet-Emergency Vehicle Siren Classification Dataset For Urban Applications. *data.mendeley.com*, 1. Available from https://data.mendeley.com/datasets/j4ydzzv4kb/1 [accessed 20 February 2024].

Shah, S.M.A. (2023) *What is a neural network flatten layer?* Available from https://www.educative.io/answers/what-is-a-neural-network-flatten-layer [accessed 7 March 2024].

Smith, J.O. (2011) *Spectral audio signal processing*. Stanford, California: W3k.

TensorFlow (2019) *TensorFlow Lite | TensorFlow Lite | TensorFlow* Available from https://www.tensorflow.org/lite [accessed 29 April 2024].

Tensorflow (2024a) *Transfer learning with YAMNet for environmental sound classification | TensorFlow Core* Available from https://www.tensorflow.org/tutorials/audio/transfer_learning_audio [accessed 28 March 2024].

Tensorflow (2024b) *tf.keras.Sequential | TensorFlow Core v2.3.0* Available from https://www.tensorflow.org/api_docs/python/tf/keras/Sequential [accessed 19 April 2024].

Tensorflow (2024c) *Classification on imbalanced data | TensorFlow Core* Available from https://www.tensorflow.org/tutorials/structured_data/imbalanced_data [accessed 24 April 2024].

Tensorflow (2024d) *Convert TensorFlow models | TensorFlow Lite* Available from https://www.tensorflow.org/lite/models/convert/convert_models [accessed 29 April 2024].

Tensorflow (2024e) *Sound classification with YAMNet | TensorFlow Hub* Available from https://www.tensorflow.org/hub/tutorials/yamnet [accessed 29 April 2024].

Tensorflow (2024f) *Adding metadata to TensorFlow Lite models* Available from https://www.tensorflow.org/lite/models/convert/metadata [accessed 29 April 2024].

The Stationery Office (1931) *Road users requiring extra care - Other vehicles (219 to 225)* Available from https://www.highwaycodeuk.co.uk/other-vehicles.html [accessed 6 May 2024].

Thakur, A. (2020) *Intuitive understanding of 1D, 2D, and 3D convolutions in convolutional neural networks.* Available from https://wandb.ai/ayush-thakur/dl-question-bank/reports/Intuitive-understanding-of-1D-2D-and-3D-convolutions-in-convolutional-neural-networks---VmlldzoxOTk2MDA [accessed 14 March 2024].

The Mobile Dev (2021) *Bird Sound Identifier | Custom Audio Classification | Custom TF lite model in an Android app* [video]. Available from https://www.youtube.com/watch?v=kBH2O6XRIw8 [Accessed 29 April 2024].

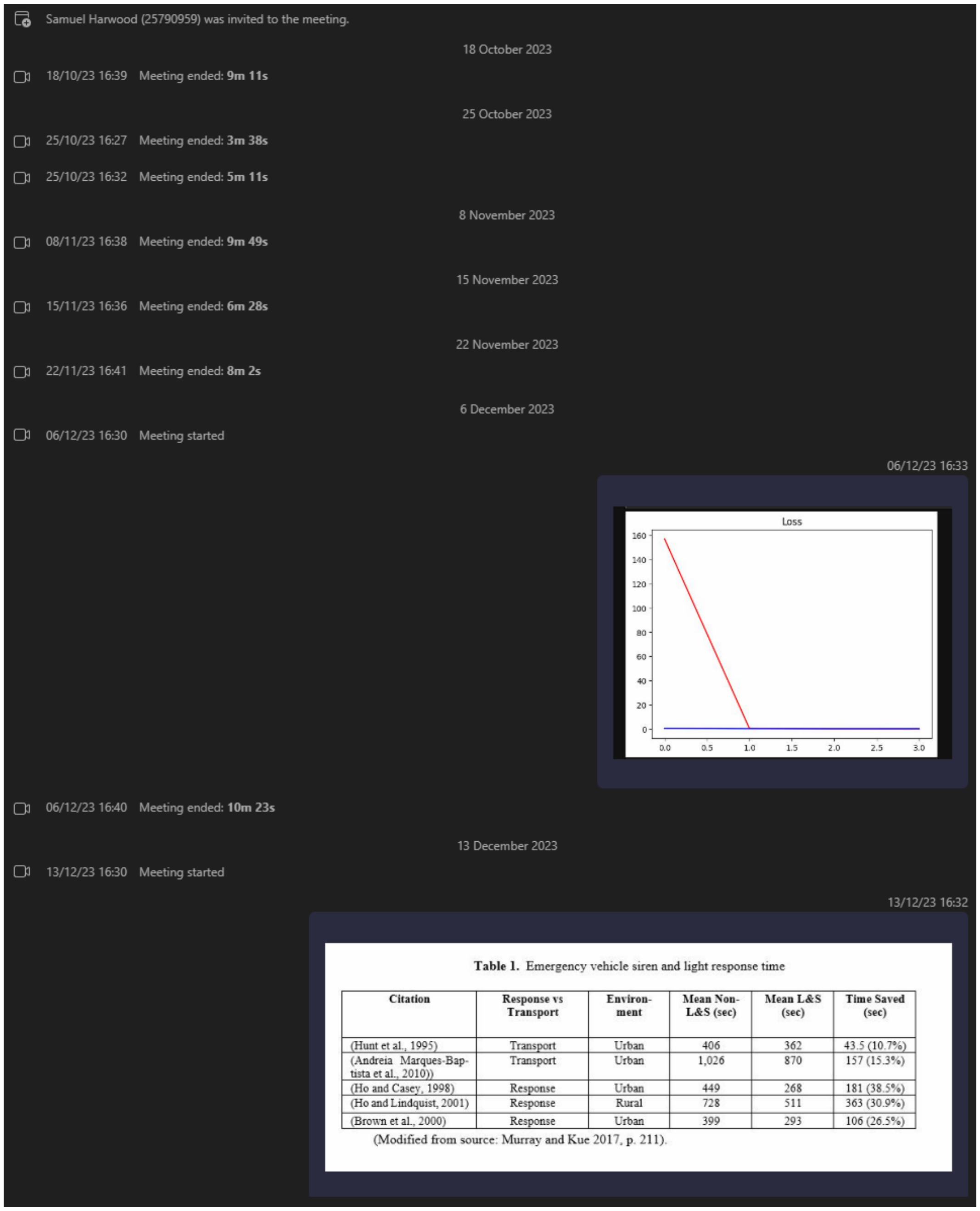
Thorslund, B., Nygårdhs, S., Malicka, A.N., Black, A.A., Hickson, L. and Wood, J.M. (2019) Exploring older adults hearing and vision and driving – The Swedish study. *Transportation Research Part F: Traffic Psychology and Behaviour*, 64 274-284. Available from https://doi.org/10.1016/j.trf.2019.04.011 [accessed 24 April 2024].

Thorslund, B., Peters, B., Lyxell, B. and Lidestam, B. (2012) The influence of hearing loss on transport safety and mobility. *European Transport Research Review*, 5(3) 117-127. Available from https://doi.org/10.1007/s12544-012-0087-4 [accessed 24 April 2024].

Wood, T. (2019) *Softmax Layer* Available from https://deepai.org/machine-learning-glossary-and-terms/softmax-layer [accessed 9 March 2024].

Yamashita, R., Nishio, M., Do, R.K.G. and Togashi, K. (2018) Convolutional Neural networks: an Overview and Application in Radiology. *Insights into Imaging*, 9(4) 611–629. Available from https://doi.org/10.1007/s13244-018-0639-9 [accessed 24 April 2024].

Appendices



Samuel Harwood (25790959) was invited to the meeting.

18 October 2023

18/10/23 16:39 Meeting ended: **9m 11s**

25 October 2023

25/10/23 16:27 Meeting ended: **3m 38s**

25/10/23 16:32 Meeting ended: **5m 11s**

8 November 2023

08/11/23 16:38 Meeting ended: **9m 49s**

15 November 2023

15/11/23 16:36 Meeting ended: **6m 28s**

22 November 2023

22/11/23 16:41 Meeting ended: **8m 2s**

6 December 2023

06/12/23 16:30 Meeting started

06/12/23 16:33

06/12/23 16:40 Meeting ended: **10m 23s**

13 December 2023

13/12/23 16:30 Meeting started

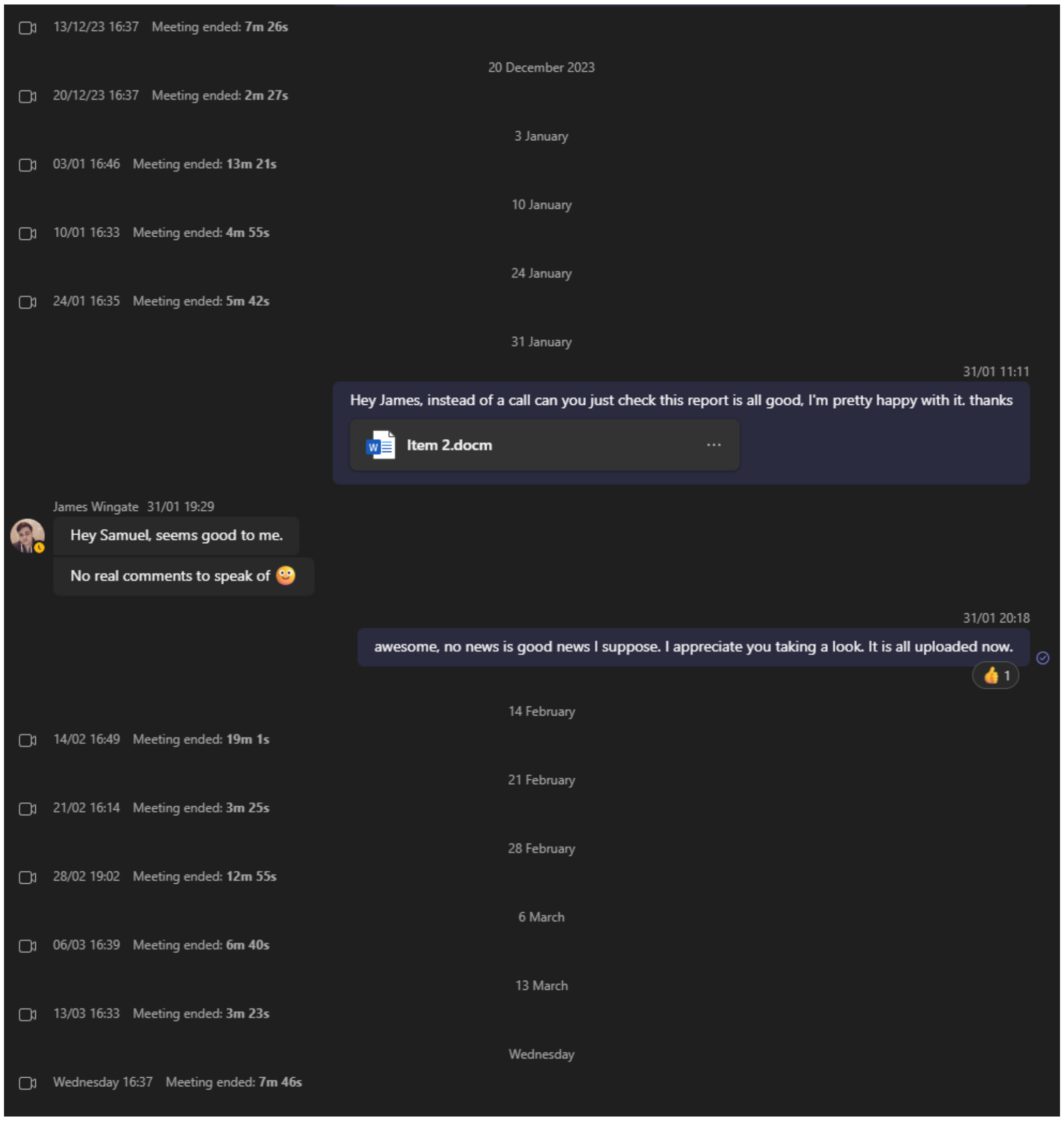
13/12/23 16:32

Table 1. Emergency vehicle siren and light response time

Citation	Response vs Transport	Environ-ment	Mean Non-L&S (sec)	Mean L&S (sec)	Time Saved (sec)
(Hunt et al., 1995)	Transport	Urban	406	362	43.5 (10.7%)
(Andreia Marques-Baptista et al., 2010))	Transport	Urban	1,026	870	157 (15.3%)
(Ho and Casey, 1998)	Response	Urban	449	268	181 (38.5%)
(Ho and Lindquist, 2001)	Response	Rural	728	511	363 (30.9%)
(Brown et al., 2000)	Response	Urban	399	293	106 (26.5%)

(Modified from source: Murray and Kue 2017, p. 211).

Figure 20.1: Evidence of weekly supervisory meetings.



13/12/23 16:37 Meeting ended: **7m 26s**

20 December 2023

20/12/23 16:37 Meeting ended: **2m 27s**

3 January

03/01 16:46 Meeting ended: **13m 21s**

10 January

10/01 16:33 Meeting ended: **4m 55s**

24 January

24/01 16:35 Meeting ended: **5m 42s**

31 January

31/01 11:11

Hey James, instead of a call can you just check this report is all good, I'm pretty happy with it. thanks

Item 2.docm

James Wingate 31/01 19:29

Hey Samuel, seems good to me.

No real comments to speak of 😳

31/01 20:18

awesome, no news is good news I suppose. I appreciate you taking a look. It is all uploaded now.

👍 1

14 February

14/02 16:49 Meeting ended: **19m 1s**

21 February

21/02 16:14 Meeting ended: **3m 25s**

28 February

28/02 19:02 Meeting ended: **12m 55s**

6 March

06/03 16:39 Meeting ended: **6m 40s**

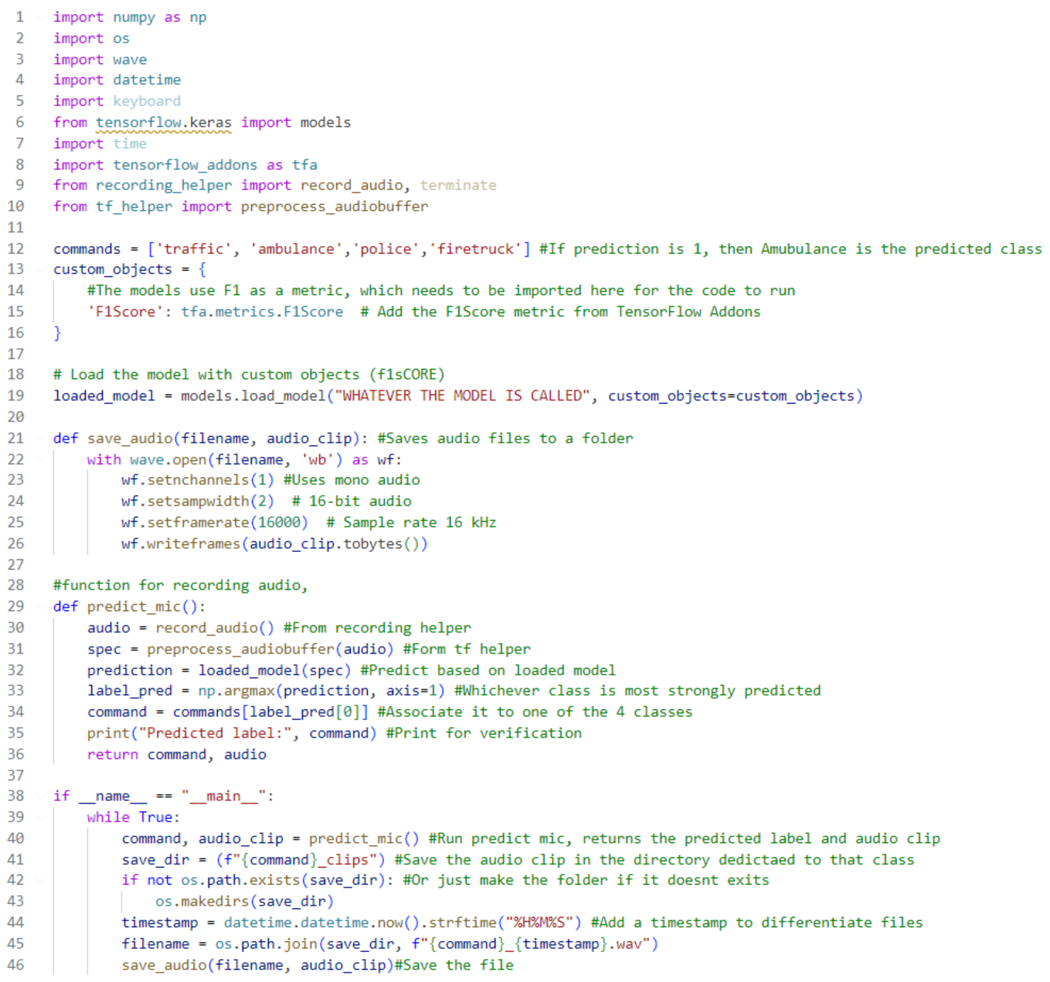
13 March

13/03 16:33 Meeting ended: **3m 23s**

Wednesday

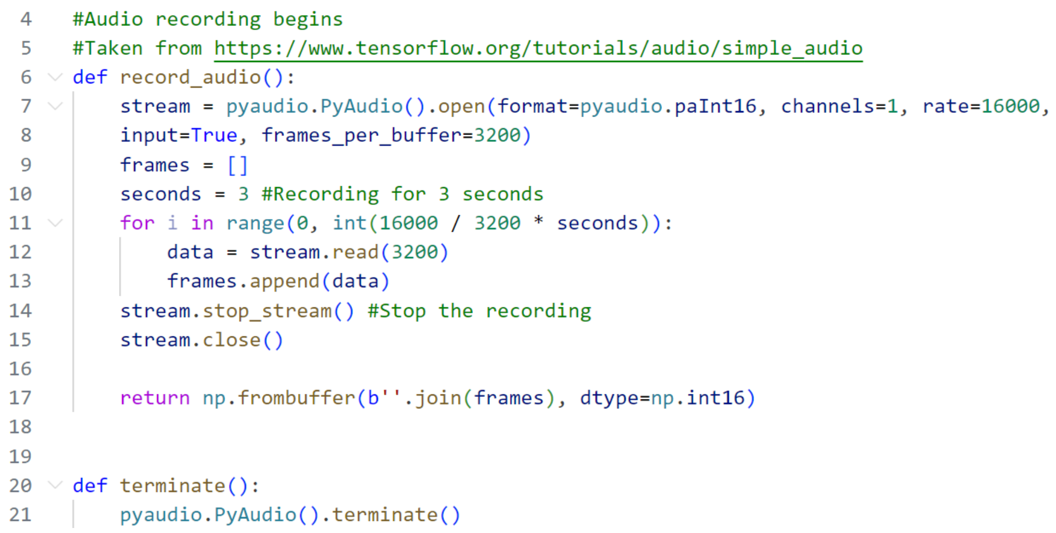
Wednesday 16:37 Meeting ended: **7m 46s**

Figure 20.2: Evidence of weekly supervisory meetings.



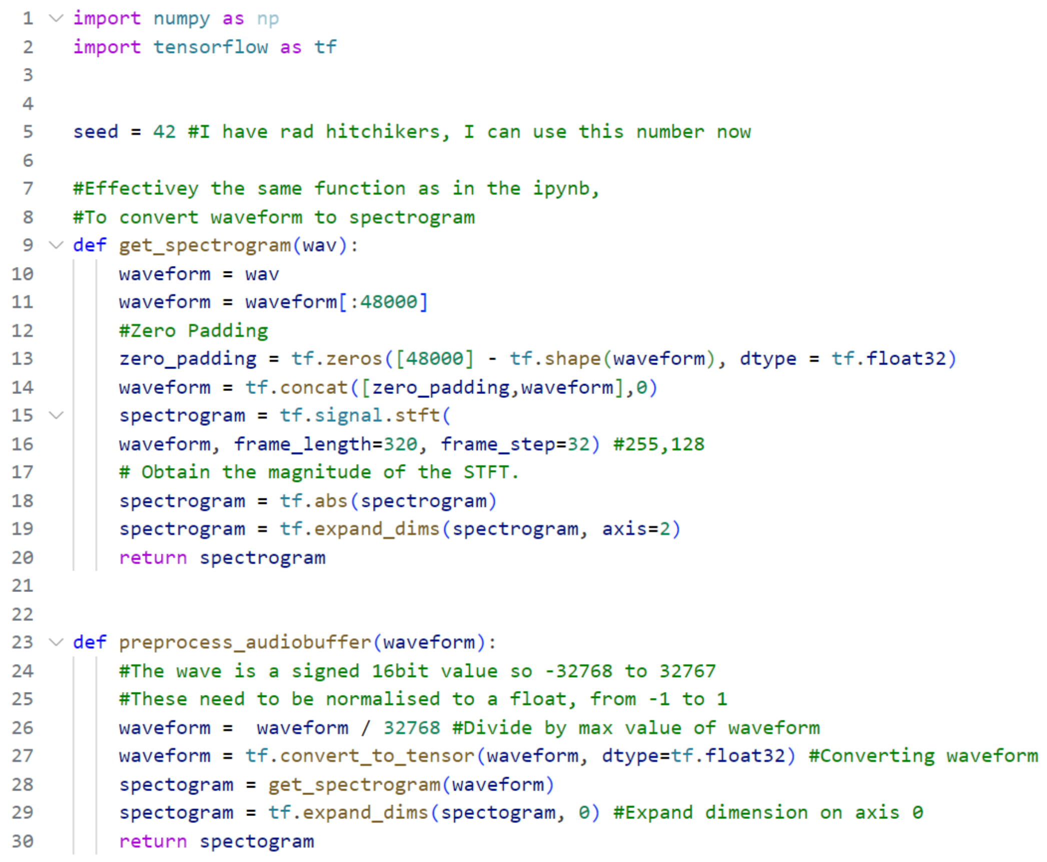
```python
import numpy as np
import os
import wave
import datetime
import keyboard
from tensorflow.keras import models
import time
import tensorflow_addons as tfa
from recording_helper import record_audio, terminate
from tf_helper import preprocess_audiobuffer

commands = ['traffic', 'ambulance','police','firetruck'] #If prediction is 1, then Amubulance is the predicted class
custom_objects = {
    #The models use F1 as a metric, which needs to be imported here for the code to run
    'F1Score': tfa.metrics.F1Score  # Add the F1Score metric from TensorFlow Addons
}

# Load the model with custom objects (f1sCORE)
loaded_model = models.load_model("WHATEVER THE MODEL IS CALLED", custom_objects=custom_objects)

def save_audio(filename, audio_clip): #Saves audio files to a folder
    with wave.open(filename, 'wb') as wf:
        wf.setnchannels(1) #Uses mono audio
        wf.setsampwidth(2)  # 16-bit audio
        wf.setframerate(16000)  # Sample rate 16 kHz
        wf.writeframes(audio_clip.tobytes())

#function for recording audio,
def predict_mic():
    audio = record_audio() #From recording helper
    spec = preprocess_audiobuffer(audio) #Form tf helper
    prediction = loaded_model(spec) #Predict based on loaded model
    label_pred = np.argmax(prediction, axis=1) #Whichever class is most strongly predicted
    command = commands[label_pred[0]] #Associate it to one of the 4 classes
    print("Predicted label:", command) #Print for verification
    return command, audio

if __name__ == "__main__":
    while True:
        command, audio_clip = predict_mic() #Run predict mic, returns the predicted label and audio clip
        save_dir = (f"{command}_clips") #Save the audio clip in the directory dedictaed to that class
        if not os.path.exists(save_dir): #Or just make the folder if it doesnt exits
            os.makedirs(save_dir)
        timestamp = datetime.datetime.now().strftime("%H%M%S") #Add a timestamp to differentiate files
        filename = os.path.join(save_dir, f"{command}_{timestamp}.wav")
        save_audio(filename, audio_clip)#Save the file
```

Figure 21.1 : Realtime audio recognition code, main.py .



```python
#Audio recording begins
#Taken from https://www.tensorflow.org/tutorials/audio/simple_audio
def record_audio():
    stream = pyaudio.PyAudio().open(format=pyaudio.paInt16, channels=1, rate=16000,
    input=True, frames_per_buffer=3200)
    frames = []
    seconds = 3 #Recording for 3 seconds
    for i in range(0, int(16000 / 3200 * seconds)):
        data = stream.read(3200)
        frames.append(data)
    stream.stop_stream() #Stop the recording
    stream.close()

    return np.frombuffer(b''.join(frames), dtype=np.int16)


def terminate():
    pyaudio.PyAudio().terminate()
```

Figure 21.2: recording_helper.py (AssemblyAI, 2022).



```python
import numpy as np
import tensorflow as tf


seed = 42 #I have rad hitchikers, I can use this number now

#Effectivey the same function as in the ipynb,
#To convert waveform to spectrogram
def get_spectrogram(wav):
    waveform = wav
    waveform = waveform[:48000]
    #Zero Padding
    zero_padding = tf.zeros([48000] - tf.shape(waveform), dtype = tf.float32)
    waveform = tf.concat([zero_padding,waveform],0)
    spectrogram = tf.signal.stft(
    waveform, frame_length=320, frame_step=32) #255,128
    # Obtain the magnitude of the STFT.
    spectrogram = tf.abs(spectrogram)
    spectrogram = tf.expand_dims(spectrogram, axis=2)
    return spectrogram


def preprocess_audiobuffer(waveform):
    #The wave is a signed 16bit value so -32768 to 32767
    #These need to be normalised to a float, from -1 to 1
    waveform =  waveform / 32768 #Divide by max value of waveform
    waveform = tf.convert_to_tensor(waveform, dtype=tf.float32) #Converting waveform
    spectogram = get_spectrogram(waveform)
    spectogram = tf.expand_dims(spectogram, 0) #Expand dimension on axis 0
    return spectogram
```

Figure 21.3: tf_helper.py (AssemblyAI, 2022).