ECE 529: PROJECT REPORT

Emergency Vehicle Siren Detection & Classification

ABSTRACT

Increasing Vehicular Traffic all around the world causes terrible traffic congestions at intersections. Most traffic lights, mainly in underdeveloped countries, feature a fixed green light sequence, which is usually implemented without taking into consideration the presence of the Emergency Vehicles that pass through them. Therefore, Emergency Vehicles such as Ambulances, Police cars, Fire trucks get stuck in traffic and are delayed in reaching their destination. This can lead to the loss of property and valuable lives. In general, emergency service vehicles use sirens to warn other road users for their quick passage through traffic. However, due to improved soundproofing techniques in modern cars, drivers may not be aware of the approaching Emergency vehicles, especially when the in-vehicle audio system is in use. Consequently, Emergency vehicles may be blocked and may even collide with other vehicles. This project aims to detect and classify the sirens of Emergency Vehicles, distinguishing them from road noise, and to help prioritize their movement by pre-empting traffic lights and alerting drivers.

CONTENTS

INTRODUCTION

Problem Definition:

In order to prioritize the movement of Emergency Vehicles through crowded traffic intersections, we need to detect their presence, early on, before their arrival. Emergency Vehicles use sirens that have unique features that distinguish them from other sounds. These features can be used to establish their presence at traffic intersections. Audio captured from these intersections can be used to extract features. An Artificial Neural Network can be trained to identify the features and determine the presence of an Emergency Vehicle. Digital Signal Processing of the siren is required for its classification as a unique Emergency Sound. The identification and classification of the siren is a subject of technological development in progress and implementation. An autonomous classification system requires obtaining an input signal with the evaluated characteristics (such as frequency components, spectral composition, power spectral density, etc.). It also needs a device that transforms that input signal into an electrical signal and a digital processing system where all the operations and techniques necessary for the processes involved in the analysis of these types of signals are performed. Therefore, to find an identifier and discriminator of emergency signals with good levels of efficiency and work, it is necessary to explore and analyze different techniques and alternatives to extract the characteristics of the sound signals.

Objective:

This project aims to implement an efficient and cost-effective method to detect and classify an Emergency Vehicle siren using MATLAB by extracting its audio features.

Literature Survey and Comparison of Techniques:

Various studies and analyses have been made to efficiently detect the presence of an Emergency Vehicle. We first compare the various detection techniques that are available to us. Anju Jose *et al.* [1] has presented an analysis paper that compares all the technologies. This is summarized in the table below: [1]

System in consideration	Optical System	Line of Sight System	Radio System	GPS System	Sensor System	Acoustic systems
Is a Dedicated Emitter Required?	Yes	Yes	Yes	No	Yes	No
Is it Susceptible to Electronic Noise Interference?	No	No	Yes	No	Yes	No
Is it Affected by weather?	Yes	Yes	No	No	Yes	No

Is a Clear Line of Sight Required?	Yes	No	No	No	Yes	Yes
Is Centralized Traffic Signal Monitoring Required?	No	No	No	Yes	No	No
Possibility of Illegal Triggering of Preemption	Low	High	High	Low	High	High
Possible Preemption of Other Approaches	No	No	Yes	No	Yes	Yes

Due to the cost-effective nature and significant advantages offered, we use an Acoustic system for this project. Following are some of the methods used to identify an Emergency Vehicle using audio techniques:

- **Ambulance Siren Detector using FFT:**

Takuya Miyazakia *et al.* [2] proposed a system of Siren Detection based on Fast Fourier Transform. In this system, FFT is employed twice for feature extraction. This permits us to convert the data into numerical values, which are first used to analyze the frequency tones, and then used to analyze the amplitude changes of each pitch frequency. It works first with an FFT of 64 points and then one of 16 points. The authors used a dsPIC microcontroller and other Frequency analysis tools for siren detection. This method seems to be very efficient even under the Doppler effect and for as low as 0dB SNR. The algorithm works in more rustic conditions, such as greater ambient noise or low volume. One of the main drawbacks of this system is the significant time delay in detecting siren sound signal. The time lag, in this case, was found to be 8 seconds. [2]

- **Ambulance Siren Detection by Longest Common Subsequence:**

Jiun-Jian Liaw *et al.* [3] presented a system of Siren Detection which uses the sequence of frequency changes in the siren sound signal. In this method, the input sound signal is separated into certain segments called frames. The proposed approach compares the pitch features of the siren with the Longest Common Subsequence to predict whether the input signal is from an ambulance or not. The Longest Common Subsequence (LCS) is used to compare the arrangement of frequencies in the frame. The score of LCS is computed using Fuzzy logic and is used to detect the emergency vehicle's siren. The obtained accuracy was 85%. [3]

- **Ambulance Siren Detection by MDF and Pitch:**

F. Meucci *et al*. [4] proposed a pitch-based algorithm deployed on an Atmel processor for siren detection based on its periodic repetition characteristics. Module Difference Function (MDF) and peak detection algorithms were used to represent the received signal as a pitch function varying over time. First, MDF, a time-domain technique, aimed to classify each portion of the audio signal as pitched or unpitched by giving us an estimate of the pitch. Secondly, the Pitch over time is

analyzed in order to recognize a time pattern of the desired siren type and declare its presence or absence. It also emphasized that the MDF implementation requires less computational load on low-power devices like Atmel than conventional correlation methods. The maximum detection accuracy reported is 98% when the SNR (signal to noise ratio) is 10dB. It was shown that the algorithm could work well with positive signal-to-noise ratio (SNR) signals. However, detection accuracy decreases dramatically to under 50% when dealing with lower SNR signals, e.g., -10dB and -15dB. [4]

- **Detection of Ambulance Siren Sounds Using Neural networks:**

Van-Thuan-Tran *et al.* [6] presented a system of ambulance siren detection based on machine learning principles. This system mainly includes the detection of siren signals using a multi-layer perceptron neural network system. Their experiments demonstrated that the proposed method could achieve the detection accuracy of 90% in simulated -15dB noisy sound data and 93.8% in real-life recorded data in traffic conditions on a downtown street. The sound signal is transformed from the time domain to the frequency domain. The Mel Frequency Cepstral Coefficients (MFCCs) and the Linear Prediction Cepstral Coefficients (LPCCs) are used to extract the sound features. The authors also proposed [5], a deep learning model to detect emergency vehicles based on the siren sound of variable input lengths. The system incorporated two CNN-based networks, one stream processed raw acoustic information, and the other was trained with MFCC and log-mel spectrogram combined features. The study reported a maximum accuracy of 98.24%. [6]

- **Other methods:**

In [1] a Radio Frequency Identification (RFID) based system is proposed in which the density of vehicles is determined to decide whether to change the traffic signal to green. This method also suggests the use of RFIDs of varying ranges to detect emergency vehicles and clear the traffic on their detection. This method has significant limitations. Firstly, if the density of vehicles is not adequate then the signal will not turn green, and the emergency vehicles will remain stuck at the signal. Secondly, if each emergency vehicle is fitted with a unique RFID tag, it will need to be very close to the RFID scanner to get detected, which is not always possible if there is not enough space for the vehicle to maneuver.

In an image processing method has been proposed that uses convolutional neural networks to detect emergency vehicles in images of traffic sourced from CCTV cameras. The main limitation of this method is that it will fail to detect emergency vehicles that are behind larger vehicles such as a bus or a truck. Another major limitation is the relatively high cost of the equipment required for this system.

Other proposed sound-based emergency detection methods also have certain limitations. In [3], detection of siren sound is done using a modified minimum mean square error method which uses the peaks of the sound wave frequency. This method uses the distinct frequencies of siren sounds as its base for detection. The limitation of this method is that there may exist sounds having frequencies like those of the siren sounds of emergency vehicles. In such cases, the system will falsely identify a non-siren sound as a siren. Our method overcomes this limitation by using many real audio samples of siren sounds with which an input sample is matched to detect sound patterns similar to those present in the real audio samples.

Here is a summary of the compared methods from [1]:

Proposed Approach	***Outcome***	***Comments***
Design and implementation of an acoustic sensor-based automatic traffic control system.	Detects the emergency vehicle by their siren sounds.	The traffic light sequence is interrupted with the approaching emergency vehicle, thereby reducing its waiting time at the intersection.
Detection of siren sounds using Fast Fourier Transform (FFT)	Detects the siren sound in 0 dB (S/N ratio). Detects the siren sound using the Doppler effect.	This work only detects the ambulance siren sound, and neither alerts the traffic nor changes the traffic signals.
Low computational microcontroller-based siren sound detection system.	Designed a siren sound detection system with low processing power.	The proposed method outperforms the existing siren detection methods in terms of processing power and cost.
Detection of siren sounds based on a pitch detection algorithm.	Capable of detecting the emergency vehicle in the presence of pitched and non-pitched noise.	The proposed algorithm outperforms the complex pattern recognition algorithms. The siren signal miss rate of the algorithm is meager.
Emergency vehicle's siren and flashing light detection based on acoustic and optical sensors.	Cost-effective solution. Different emergency vehicles are detected.	The proposed system alert's the drivers of standard vehicles and pedestrians about the approaching emergency vehicle.
Cross microphone array-based emergency vehicle detection.	Determines the incoming direction of the siren sound.	The proposed system for source detection outperforms the existing sound intensity techniques. It delivers precise warning data to the driver.

THEORY

Overview:

From the analysis of papers [6], [7], [8], [9] we implement this project with a combination of pre-processing and Artificial Neural Network (ANN) based regression to extract and detect siren sounds of emergency vehicles from real-time traffic noise. An array of sensitive microphones placed at intersections can record the traffic noise and the audio will be divided into clips of fixed short durations for processing. The input signal is windowed into blocks to make the algorithm expeditious and efficient. After dividing the signals into several windows, the proposed method extracts Mel-frequency cepstral coefficients (MFCCs), spectral centroid, spectral flux, spectral roll-off and zero-crossing rate as features from input audio signal. These features are adjusted to train an ANN. This reduces possible overfitting of the Neural Network

The training and testing of the ANN is done in MATLAB by using standard data set available in the repository. For the purposes of this project, over 100 audio files of emergency vehicle noise were recorded conventionally by utilizing files on the internet. These were carefully combined with traffic noises to simulate real-world conditions. The audio files for Non-emergency Sounds were obtained from an online repository [10]. The data is separated for training and evaluation. The trained network can be stored and used for the project.

First, we go through some theoretical concepts required for the project.

Sirens and their characteristics:

Siren is a unique signal sounded by alarm systems or emergency service vehicles such as fire trucks, police cars and ambulances. Emergency vehicles' issue a siren sound to alert other drivers for their need for the right of way on the road. In this section, we focus on the detection of siren sounds, in which the operation modes are basically different from those of other vehicles. The spectrum of an ambulance siren is as shown below:

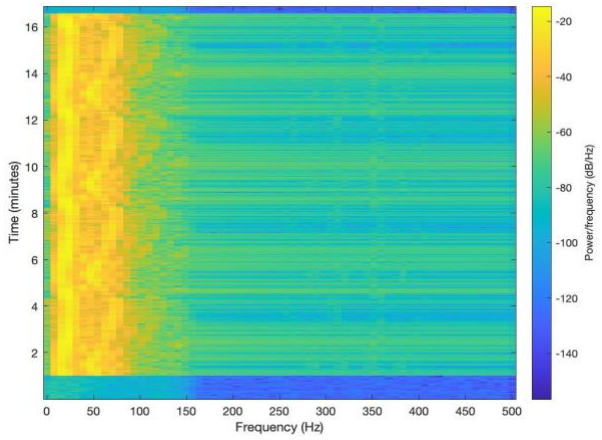

Figure 1: Frequency spectrum of an Ambulance Siren generated in MATLAB

We can identify two distinctive qualities of a typical pure siren signal: the frequency content and the periodic repetition.

Even though the siren signal consists of several harmonic spectral components, the one corresponding to the lowest frequency is dominant. The periodic variation of this dominant component can be visualized as a curve that relates the present value of frequency with time. This curve, which is called the frequency characteristic curve of the siren, may be regarded as a pattern and thus, the problem of siren identification reduces to a pattern recognition task. [4]

Therefore, the techniques based on spectral analysis (such as Fast Fourier Transform or Short Time Fast Discrete Fourier Transform) often do not identify the correct dominant frequency of the siren as many harmonics persist as significant components in the audio signal. Methods based on using a filter bank centered around the High and Low frequencies also do not detect the siren power. The detection method must take advantage of the relatively long duration of the siren signal, lasting at least some seconds with respect to other sound sources present in the traffic background that are usually of shorter duration. [4]

Feature Extraction from Sirens:

The selection of features is a key issue when designing an effective signal detector. [8]Each time frame requires components that comprise and capture the essence of information relevant for classification. They must be robust to variations in the signal and noise and should be efficient in terms of computational and space complexity. A large set of previously used features for different audio processing tasks are used to find such features. These are divided into three main categories:

- Time-domain features
- Frequency-domain features
- Wavelet-based features

- **Time Domain Features**

Time-domain features include: [8]

– Pitch: Pitch can be defined as the sensation of frequency. A high pitch sound corresponds to a high-frequency sound wave and a low pitch sound corresponds to a low-frequency sound wave.
– Short-Time Energy: Siren sounds tend to have high energy in the given spectral band compared to the energy attributed to noise.
– Zero crossing rate (ZCR): The number of times the sign of a time series changes within a frame. It roughly indicates the frequency that dominates during that frame.

- **Frequency Domain Features**

– MFCC: Mel-Frequency Cepstral Coefficients are used to describe the spectral shape of the signal. These coefficients are extracted by applying the discrete cosine transform (DCT) to the log-energy outputs of the nonlinear Mel-scale filter bank.

- Spectral flux: Spectral Flux is a measure of how quickly the power spectrum of a signal is changing, defined as the Euclidean distance between the spectra of two adjacent frames. The spectral flux can be used to determine the timbre of an audio signal, or for onset detection. [8]

- **Wavelet-based features**

The wavelet coefficients capture time and frequency localized information about the audio waveform. The wavelet transform can be viewed as a multi-level process of filtering the signal using a low-pass (scaling) filter and a high-pass (wavelet) filter. [8]

- Discrete Wavelet Transform (DWT): Each level is calculated by passing only the previous wavelet approximation coefficients through discrete-time low and high pass filters.
- Wavelet packet transform (WPT): In the WPT, both the detail and approximation coefficients are decomposed to form a complete binary tree.

In this project, frequency domain features such as MFCC's and Spectral Centroid and flux are used for siren detection.

Siren Sound Detection:

The Siren Sound identification system was created using neural networks trained using the Back Propagation (BP) algorithm. In each of the networks, the FCC's vector is used as input of the first layer, which connects to a hidden layer in front of an output layer with three nodes. MLP is a simple form of feed-forward neural network and is easy to implement. [6]
The implementation of a typical neural network includes three steps:

- Training
- Validation
- Testing.

- **Training**

In the training stage, the extracted MFCC features of the training dataset are provided as input to estimate the weight parameters in a neural network.

- **Validation**

In the validation phase, we use the validation dataset to evaluate the performance of the trained model, and through this basis to tune the parameters to achieve the best one.

- **Testing**

In the testing phase, we pass the unknown audio signal to the fully trained network to obtain the corresponding output, the output with maximum response value is selected as identified result.

The structure of the Siren Signal Identification Network is as shown below: [5]

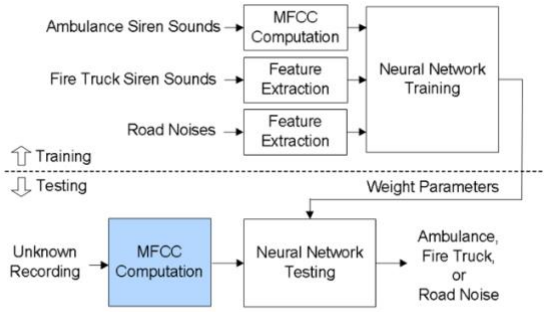

Figure 2: Structure of a Siren Signal Identification Network

MFCC:

The Mel-frequency cepstrum (MFC) depicts the short-term power spectrum of a sound signal based on the linear cosine transform of a log power spectrum on a nonlinear Mel scale of frequency. Mel-frequency cepstral coefficients (MFCCs) are the coefficients that jointly make up an MFC. They are formed from a cepstral representation of a sound signal (i.e. a nonlinear spectrum of another spectrum). The difference between the cepstrum and the Mel-frequency cepstrum is that in MFC, the frequency bands are equally spaced on the Mel scale, which approaches the human auditory system's response more accurately than the linearly spaced frequency bands used in a typical spectrum. This kind of frequency warping can allow for a better representation of sound. *(Source: Wikipedia/MFCC)*

MFCCs are commonly derived as follows:

- Take the Fourier transform of a windowed part of the signal.
- Map the powers of the spectrum found above onto the Mel scale, using overlapping triangular windows or cosine overlapping windows.
- Take the logs of the powers at each of the Mel frequencies.
- Take the discrete cosine transform of the list of Mel log powers.
- The MFCCs will be the amplitudes of the resulting spectrum.

The Mel-spectrogram for an ambulance siren is shown below:

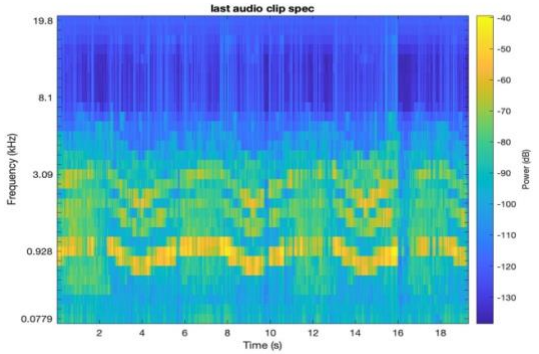

Figure 3: Mel-Spectrogram of an ambulance siren generated from MATLAB

Spectral Centroid *(Source: MATLAB Documentation/Spectral)*

Spectral centroid is the frequency-weighted sum normalized by the unweighted sum:

$$\mu_1 = \frac{\sum_{k=b1}^{b2} f_k s_k}{\sum_{k=b1}^{b2} s_k}$$

where,

- f_k is the frequency in Hz corresponding to *k*.
- s_k is the spectral value at *k*. Both magnitude and power spectrum are commonly used.
- b_1 and b_2 are the band edges, over which the spectral centroid is calculated.

Spectral Flux

Spectral flux is the measure of the variability of the spectrum over time:

$$\text{Flux}(t) = \left(\sum_{k=b1}^{b2} |s_k(t) - s_k(t-1)|\right)^{\frac{1}{p}}$$

where,

- s_k is the spectral value at *k*. Both magnitude and power spectrum are commonly used.
- b_1 and b_2 are the band edges, over which the spectral flux is calculated.
- *p* is the norm type.

Spectral flux is popularly used in onset detection and audio segmentation.

Spectral Slope

Spectral slope measures the amount of decrease of the spectrum:

$$\text{Slope} = \frac{\sum_{k=b1}^{b2}(f_{k-\mu_f})(s_{k-s})}{\sum_{k=b1}^{b2}(f_{k-\mu_f})^2}$$

where

- f_k is the frequency in Hz corresponding to *k*.
- μ_f is the mean frequency.
- s_k is the spectral value at *k*. The magnitude spectrum is commonly used.
- μ_s is the mean spectral value.
- b_1 and b_2 are the band edges, in bins, over which to calculate the spectral slope.

Spectral slope is used widely in speech analysis, particularly in modeling speaker stress. The slope is directly related to the resonant features of the vocal folds and is also applied to speaker identification. Spectral slope is a socially important aspect of timbre. Spectral slope is most pronounced when the energy in the lower formants. *(Source: MATLAB Documentation/Spectral)*

IMPLEMENTATION & ANALYSIS

Implementation in MATLAB:

The project is implemented in Matlab version 9.10 R2021 using essential toolboxes such as Audio Toolbox and Deep Learning Toolbox.

Two separate programs were implemented:

- To train and evaluate the Neural Network
- To execute Real-Time Detection

Two separate sets of data were recorded and used for training and evaluation categorized as Emergency and Non-Emergency sound signals. 70% of the pre-recorded dataset is used for training the Neural Network, while 30% is used for the evaluation of the network. This is to prevent the overfitting of data. The Non-Emergency sounds were obtained for an online repository. [8]

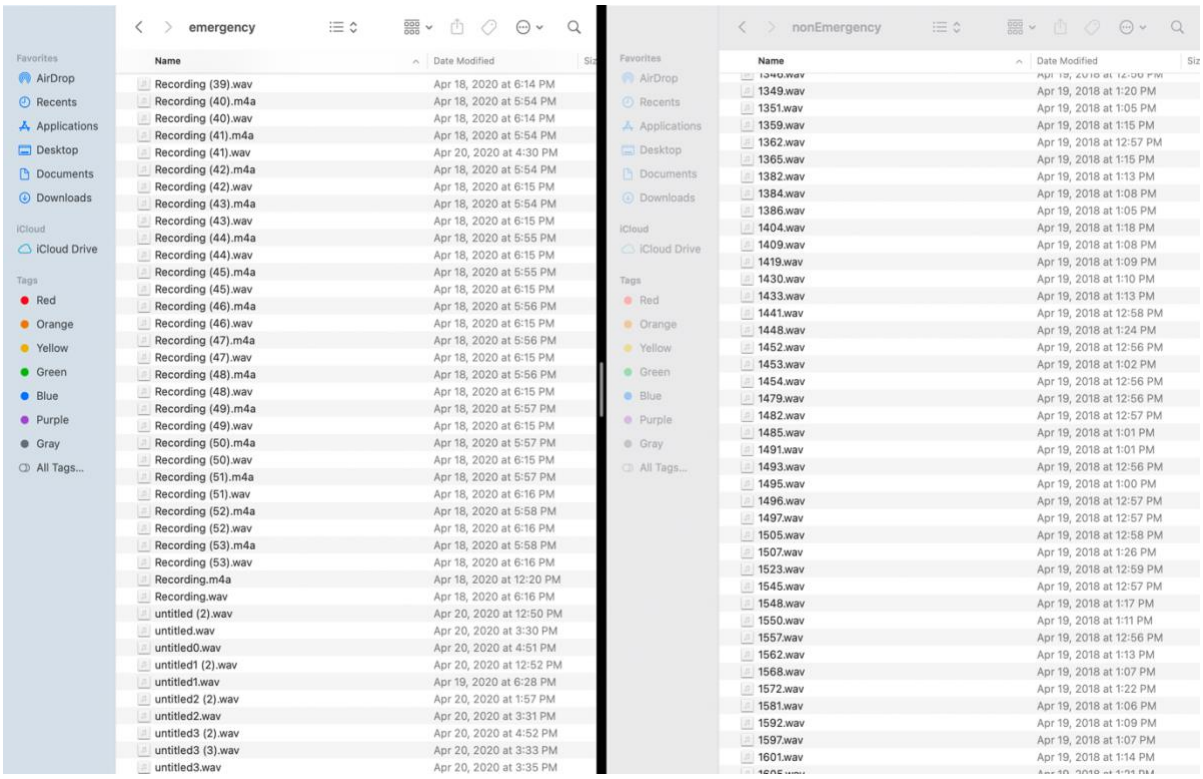

Figure 4: The pre-recorded data for training and evaluation

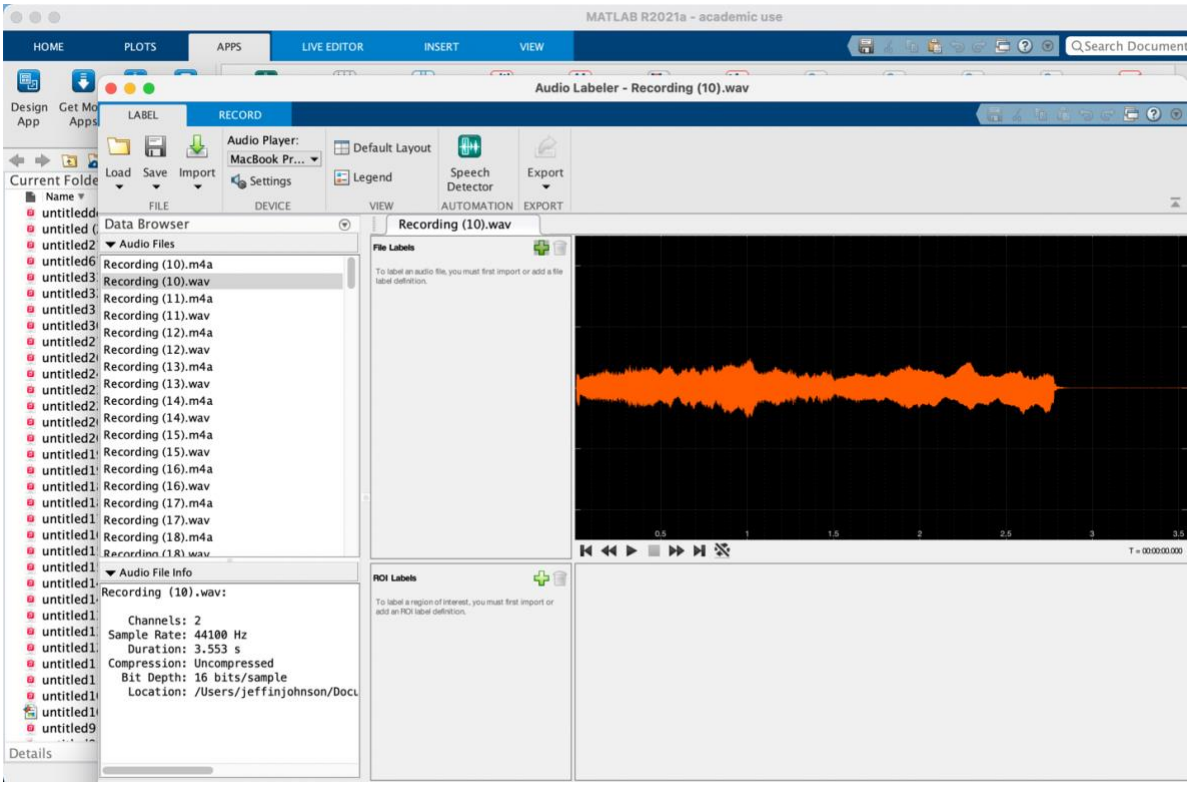

Figure 5: Using the Audio Toolbox in MATLAB

Audio Feature Extraction:

MATLAB has an inbuilt method to extract required features from audio signals: (from MATLAB Documentation) *(Source: MATLAB Documentation/audioFeatureExtractor)*

```
audioFeatureExtractor: encapsulates multiple audio feature extractors into a
streamlined and modular implementation
```

Some of the parameters of this function are:

- Analysis window: A Hamming window is used in this project of length 0.3xSampling Rate.
- Overlap length of adjacent analysis windows: Specified as an integer in the range [0, numel(Window)). 0.2xSampling Rate.
- FFT length: The default, [], means that the FFT length is equal to the window length, (numel(Window)).
- Input Sampling Rate: (in Hz) The default value of 44kHz is used.

Some of the extracted features:

- `linearSpectrum` – Extract linear spectrum
- `melSpectrum` – Extract mel spectrum
- `mfcc` – Extract mel-frequency cepstral coefficients (MFCC)
- `mfccDelta` – Extract delta of MFCC
- `mfccDeltaDelta` – Extract delta-delta of MFCC
- `spectralCentroid` – Extract spectral centroid
- `spectralFlux` – Extract spectral flux
- `spectralSlope` – Extract spectral slope
- `pitch` – Extract pitch

Settable parameters for the Mel spectrum extraction are:

- `FrequencyRange` - Frequency range of the extracted spectrum in Hz; defaults to `[0, SampleRate/2]`.
- `SpectrumType`- Spectrum type, "power" or "magnitude". If unspecified, it defaults to "power".
- `NumBands`- Number of Mel bands, NumBands defaults to 32.
- `FilterBankNormalization`- Normalization applied to bandpass filters, "bandwidth", "area", or "none". It defaults to "bandwidth".
- `WindowNormalization`- Apply window normalization
- `FilterBankDesignDomain`- Domain in which the filter bank is designed, either "linear" or "warped". It defaults to "linear".

Settable parameters for the MFCC extraction are:

- `NumCoeffs`- Number of coefficients returned for each window. It defaults to 13.
- `DeltaWindowLength`- Delta window length. It defaults to 9. This parameter affects the `mfccDelta` and `mfccDeltaDelta` features.

The Mel-frequency cepstral coefficients are calculated using the Mel Spectrum.

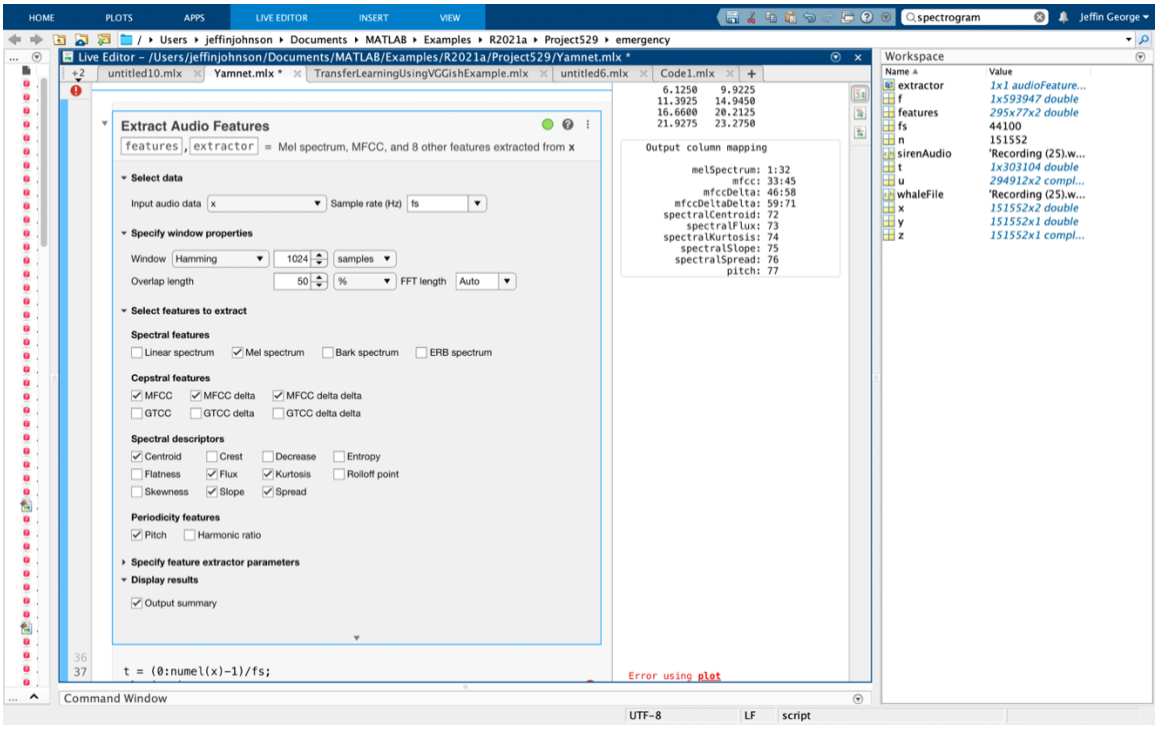

Figure 6: Audio Toolbox in MATLAB

Algorithm:

The `audioFeatureExtractor` creates a feature extraction pipeline based on the selected features. To reduce computations, `audioFeatureExtractor` reuses intermediary representations. Some intermediate representations can be output as features. *(Source: MATLAB Documentation/audioFeatureExtractor)*

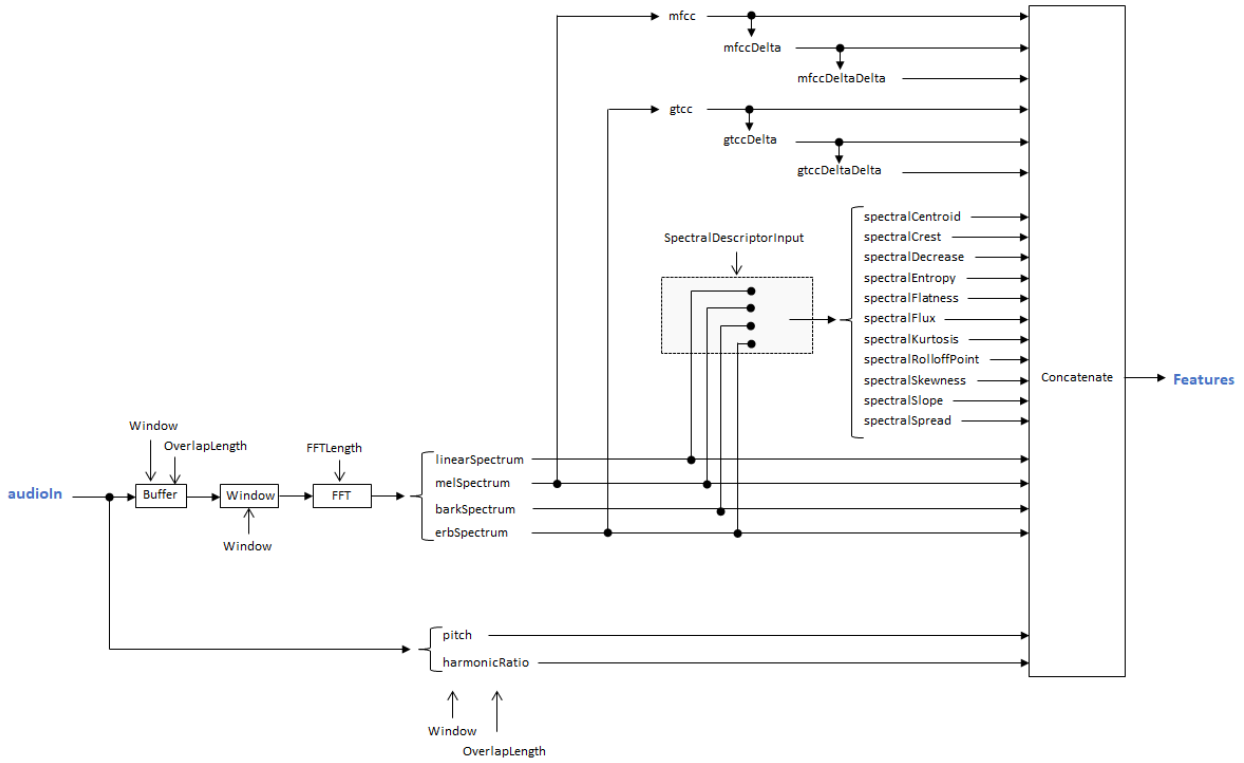

Figure 7: Pipeline model of audioFeatureExtractor

Training of the Neural Network:

Various types of neural networks can be trained using the `trainNetwork` function for classification and regression tasks. In this project, a recurrent neural network (RNN) with a long short-term memory (LSTM) is trained.

LSTM-RNN: [6] Long Short-Term Memory Recurrent Neural Networks are trained using the backpropagation (BP) algorithm. In each network, the MFCC's as a vector input is used in the first layer, which connects to a hidden layer in between the output layer with three nodes. RNN networks are compelling models that are often difficult to train. However, the Long Short-Term Memory (LSTM) is a specific RNN architecture whose design makes it much simpler to train. This architecture achieves excellent results in speech processing or natural language processing. LSTM-RNN can considerably reduce the number of network parameters by weight sharing, and recurrent connections help to increase the network depth. This can efficiently contract the sizes of the input and hidden layers. In LSTM, each cell has four components: the cell weights, the input gate, the forget gate, and the output gate. Each part has weights associated with all its inputs from the preceding layers, plus the input from the previous time step.

The predictors and responses are listed as a single input or two separate inputs when training a neural network. The extracted features are given as sequences and used in a `sequenceInputLayer` as the first layer of the deep learning model. When this is used as the first layer in a network, `trainNetwork` expects the training and validation data to be formatted in cell arrays of sequences, where each sequence consists of feature vectors over time. `sequenceInputLayer` requires the time dimension to be along the second dimension. These are the other layers used in the LSTM-RNN: *(Source: MATLAB Documentation/DeepLearning)*

- `sequenceInputLayer(inputSize)`: A sequence input layer inputs sequence data to a network.
- `lstmLayer(numHiddenUnits)`: An LSTM layer that learns long-term dependencies between time steps in time series and sequence data. The layer implements additive interactions, which will help develop gradient flow over long sequences during training.
- `fullyConnectedLayer(outputSize)`: A fully connected layer that multiplies the input by a weight matrix and then adds a bias vector.
- `classificationLayer`: A classification layer that calculates the cross-entropy loss for classification and weighted classification tasks. The layer then infers the number of classes from the output size of the previous layer.

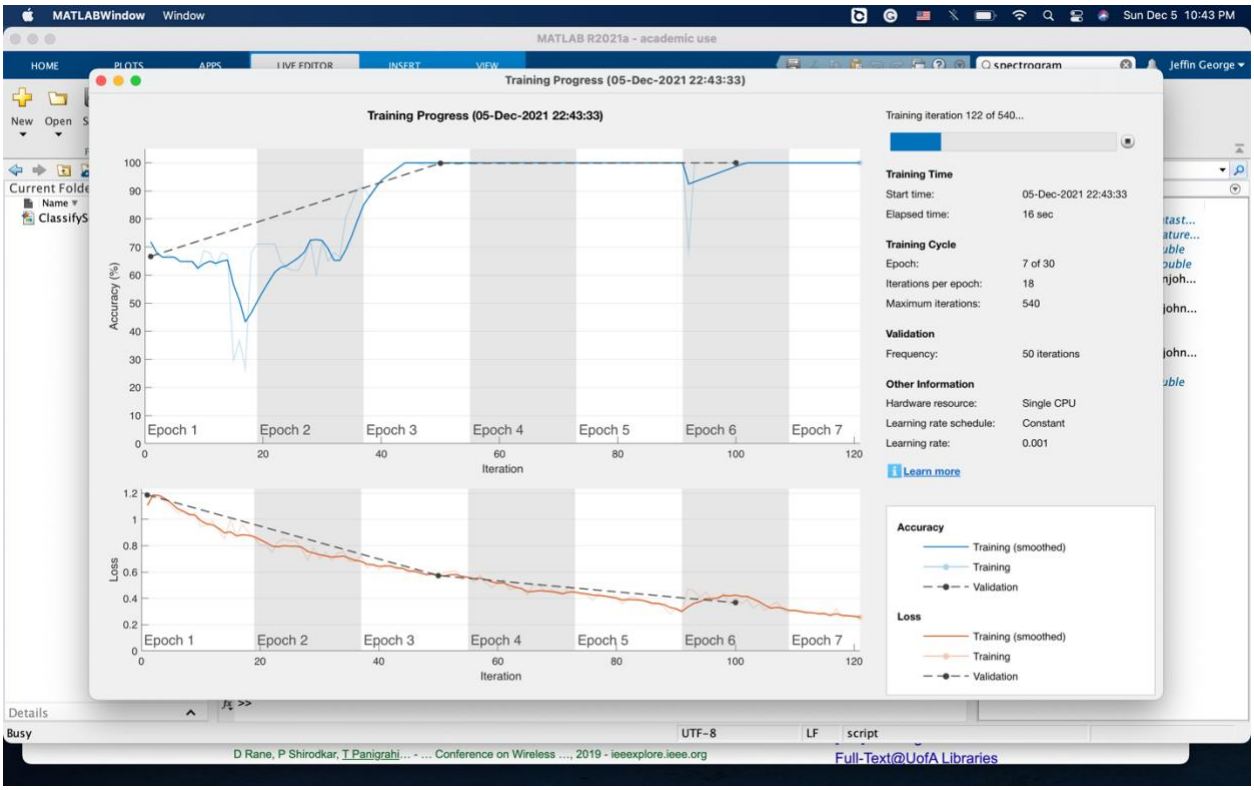

Figure 8: Training of the LSTM-RNN in MATLAB

Analysis using YAMNet:

YAMNet is a pretrained deep learning Neural Network that predicts 521 audio event classes based on the AudioSet ontology and employs a depth-wise-separable convolution architecture. The `classifySound` function uses YAMNet to classify audio segments into sound classes described by the AudioSet ontology. This function preprocesses the audio so that it is in the format required by YAMNet and post processes YAMNet's predictions with common tasks that make the results more interpretable. Due to the small size of our dataset and the considerable time is taken for computation and training, we will use this efficient pre-trained Neural Network for our analysis.

Some features of YAMNet: *(Source: MATLAB Documentation/yammnet)*
The network has 86 layers. There are 28 layers with learnable weights: 27 convolutional layers and one fully connected layer. Preprocessing includes resampling of the audio data to 16 kHz and casting to single precision. Buffer it into L overlapping segments where every segment is 0.98 seconds and is overlapped by 0.8575 seconds. A spectrogram is computed using magnitudes of the Short-Time Fourier Transform using a 25 ms periodic Hann window with a 10 ms hop and a 512-point DFT. The audio is now represented by a 257-by-96-by-L array, where 257 is the number of bins in the one-sided spectra and 96 is the number of spectra in the spectrograms. A Mel spectrogram is computed by mapping the spectrogram to 64 Mel bins covering the range 125-7500 Hz. A stabile log Mel spectrogram is estimated by applying log (Mel-spectrum + 0.001) where the offset is used to prevent taking a logarithm of zero. These features are then framed into 50%-overlapping examples of 0.96 seconds, where each sample covers 64 Mel bands and 96 frames of 10 ms each Pass the 96-by-64-by-1-by-L array of Mel spectrograms through YAMNet to return an L-by-521 matrix. The output from YAMNet corresponds to confidence scores for each of the 521 sound classes over time.

The identified sound regions that overlap by 50% or more are consolidated into single regions. The region start time is the shortest start time of all sounds in the group. The region end time is the largest end time of all sounds in the group. The function returns time stamps, sound classes, and the mean and maximum confidence of the sound classes within the region in the results table.

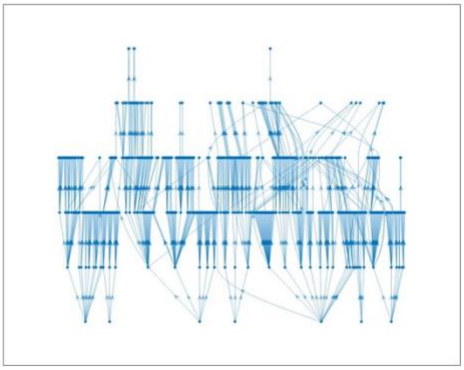

Figure 10: The graph of YAMNet's sound class ontology

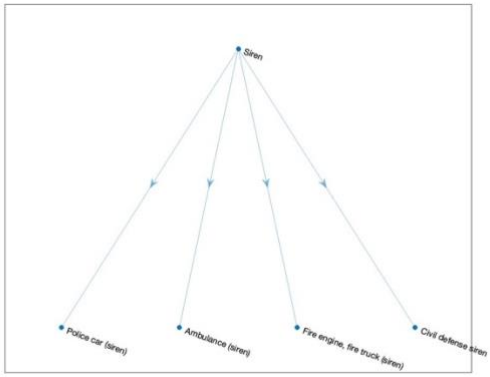

Figure 9: The subgraph for Siren sounds

RESULTS AND OBSERVATIONS

Performance Evaluation:

After the training and evaluation stage, a confusion matrix is generated to view the performance. A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. It allows the visualization of the performance of an algorithm. It will enable easy identification of confusion between classes, i.e., one class is commonly mislabeled as the other. The confusion matrix obtained after training and evaluation is given below:

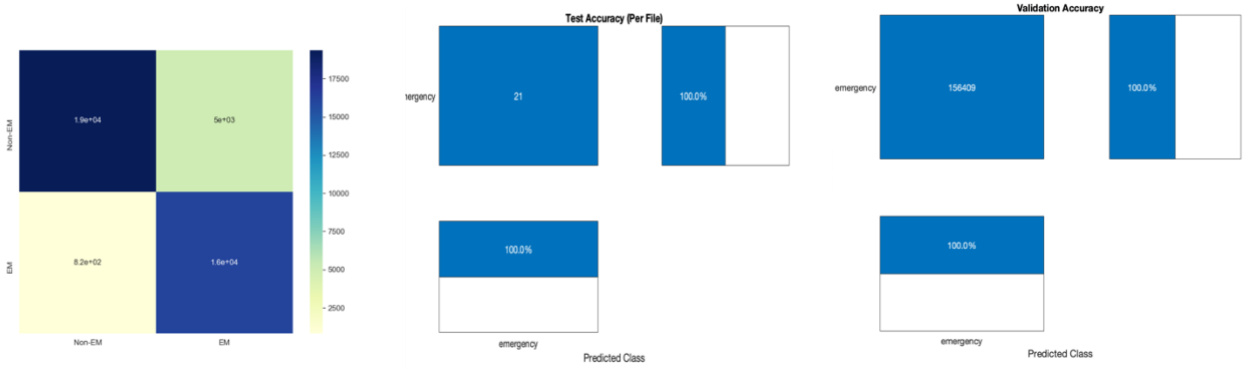

Figure 11: Confusion matrices generated for training and validation

From the results, we can see that not enough adequate data was used to train the network. Therefore, giving a lower accuracy. The ideal matrix observed should have darker regions on the diagonal ends representing true positives and lighter regions on the remaining squares denoting low false positives.

When networks are trained for deep learning, the training progress can be monitored. By plotting various metrics during training, the training progress can be viewed. Information regarding the network accuracy and whether the network is starting to overfit the training data can be obtained. `trainNetwork` creates a figure and displays training metrics at every iteration. Each iteration estimates the gradient and an update of the network parameters. If validation data is specified, then the figure shows validation metrics each. The figure plots the following:

- Training accuracy - Classification accuracy on each individual mini-batch.
- Smoothed training accuracy - Smoothed training accuracy, obtained by applying a smoothing algorithm to the training accuracy. It is less noisy than the unsmoothed accuracy, making it easier to spot trends.
- Validation accuracy — Classification accuracy on the entire validation set.
- Training loss smoothed training loss, and validation loss — The loss on each mini-batch, its smoothed version, and the loss on the validation set, respectively

Output from YAMNet:

The program was able to successfully classify the sirens of emergency vehicles giving high scores in classification. The output results contain a table of identified sound classes. The audio plot is segmented into classified intervals. The timestamps are separated and stored in the Sound Table. A final table of results with the calculated score of each sound class is also computed.

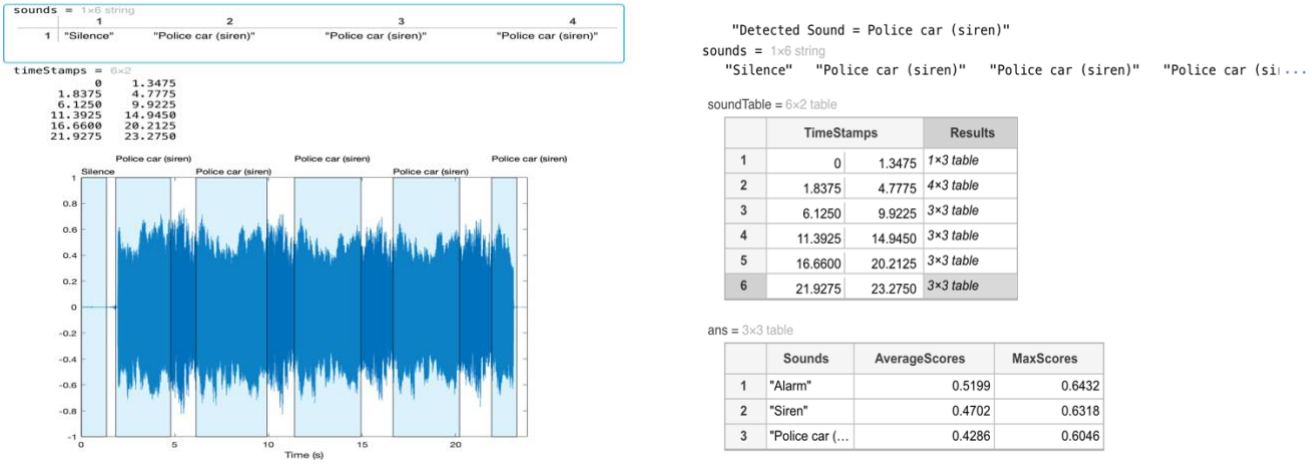

Figure 12: Output after YAMnet classification

A Word map is also generated by the in-built function, in which the most recognized sound class appears larger, while other identified classes appear smaller. We can see that Siren sounds are recognized well.

Figure 13: Generated Wordmap

Challenges Faced:

- Non-Availability of a proper data set for testing and evaluation. The project was implemented with a mix of data obtained from manual recording and from online repositories. Thus, the accuracy of the classifier would be affected.
- Adapting a standard audio file with proper dimensions would have made the code more efficient. A lot of troubleshooting was required to fix errors caused by incorrect size dimensions of the files. Proper database management is necessary to implement the project effectively.
- Long computational times due to limited hardware resources.

CONCLUSION

The implemented project was successful in detecting and classifying the sirens of emergency vehicles. The ambulance siren sound signals were detected with sufficient accuracy by the classifier. The system can be significantly improved with a more extensive data set and efficient implementation techniques.

Project Contributions:

The project was proposed after a detailed literature survey of papers and articles relating to siren detection techniques. Various methods were analyzed and compared. The method that offered the most advantages and feasible implementation methods was selected. The reference papers provided a lot of insight on efficient methodologies and theory. Background knowledge of signal processing and neural networks was necessary and consumed a large amount of the time spent on the project. Theoretical information was sourced from these papers and cited.

The project relied heavily on MATLAB documentation and examples for coding and troubleshooting. Deep learning and Audio Toolbox contained sufficient tools and resources to implement and practice projects. The final code for training and evaluation was written using base examples for other MATLAB examples. For the analysis with YAMnet, the code was written from scratch, using the built-in functions and pre-trained neural network.

Future Scope:

This project can be extended for real-time detection and implemented in traffic intersections on a small-scale embedded system unit like a Raspberry Pi. The software can be implemented in Python more efficiently and accurately. This unit along with the traffic microcontroller, can pre-empt traffic lights to guide emergency vehicles to pass through. The entire system will be cost-effective and easy to implement.

REFERENCES

[1] A. Jose, "Jose, A. Traffic Light Pre-emption control System for Emergency Vehicles Miss.," *internationaljournalssrg.org.*

[2] Y. K. M. S. Takuya Miyazakia, "Ambulance Siren Detector using FFT on dsPIC," in *1st IEEE/IIAE International Conference on Intelligent Systems and Image Processing 2013*, 2013.

[3] W.-S. W. H.-C. C. M.-S. H. a. C.-P. L. Jiun-Jian Liaw, "Recognition of the Ambulance Siren Sound in Taiwan by the Longest Common Subsequence," in *2013 IEEE International Conference on Systems, Man, and Cybernetics*, Taipei, 2013.

[4] L. P. E. D. R. L. L. P. D. F. Meucci, "A REAL-TIME SIREN DETECTOR TO IMPROVE SAFETY OF GUIDE IN TRAFFIC ENVIRONMENT," in *16th European Signal Processing Conference (EUSIPCO 2008)*, Lausanne, Switzerland, August 25-29, 2008.

[5] W.-H. T. Van-Thuan Tran, "Acoustic-Based Emergency Vehicle Detection Using Convolutional Neural Networks," *IEEE Access, Volume 8, 2020,* pp. 75702-75713, 2020.

[6] Y.-C. Y. W.-H. T. Van-Thuan Tran, "DETECTION OF AMBULANCE AND FIRE TRUCK SIREN SOUNDS USING NEURAL NETWORKS," in *Proceedings of 51st Research World International Conference, Hanoi, Vietnam, 26th -27th July 2018*, 2018.

[7] A. Y. Y. M. Dean Carmel, "Detection of Alarm Sounds in Noisy Environments," in *25th europen Signal Processing Conference (EUSIPCO)*, 2017.

[8] S. Dey, "Emergency Vehicle Detection," https://github.com/sheelabhadra/Emergency-Vehicle-Detection, 2019.

[9] D. E. C. O. a. N. C. M. Agustín Soto Otálora, "METHODS FOR EXTRACTION OF FEATURES AND DISCRIMINATION OF EMERGENCY SIRENS," *ARPN Journal of Engineering and Applied Sciences,* pp. 1525-1532, MARCH 2017.

[10] P. A. H. V. ,. S. B. H. R. K. Binish Fatimah, "An automatic siren detection algorithm using Fourier Decomposition Method and MFCC," in *11th ICCCNT 2020*, IIT - Kharagpur, 2020.

[11] P. S. T. P. a. S. M. Dharma Rane, "Detection of Ambulance Siren in Traffic," National Institute of Technology Goa , Goa, 2019.

[12] S. S. Mohit Jha, "Design Of Fuzzy Logic Traffic Controller For Isolated Intersections With Emergency Vehicle Priority System Using MATLAB Simulation," Jabalpur Engineering College, Jabalpur, M.P., India.

[13] S. C. A. R. S. S. F. Beritelli, "An Automatic Emeregency Signal Recognition System for the Hearing Impaired," University of Catania, Viale , Catania, 2006.

APPENDIX

Source Code:

– MATLAB Code to generate spectrum of an audio signal:

```
sirenAudio = 'Recording (25).wav'; %Load Audio file
[x,fs] = audioread(sirenAudio); %Sample and Store

y = x(:,1);
n = length(y)
sound(y,fs)
t = (0:n-1)/fs

plot(t,y)
xlabel('Sample Number')
ylabel('Amplitude')

z = fft(y);
plot(t,z)
xlabel('Sample Number')
ylabel('Amplitude')

spectrogram(y,128,120,128,1e3) %To plot the spectrogram
```

– MATLAB Code for training and evaluation of LSTM-RNN for detection:

```
clc
close
clear

datafolder = fullfile(pwd);
%myFolder =
('Users/jeffinjohnson/Documents/MATLAB/Examples/R2021a/Project529/emergenc
y');

fs=44100 %Sampling freqiency
duration = 0.5;
N = duration*fs;
i=1;

% Load Audio Data
ads =
audioDatastore(datafolder,'FileExtension',{'.wav','.mp4'},'IncludeSubfolde
rs',true);
data = readall(ads)

%sound(data,fs)
%info

%melSpectrogram(data(:,1),fs)
```

```
%title('last audio clip spec')

% Separate Train and Validate Data
audioTrain = [data(1:70,1)];
%audioTrain = str2double(audioinTrain)
%audioTrain = cell2mat(audioTrain)
labelsTrain = [data(1:70)];

audioValidate = [data(71:end,1)];
%audioValidate = str2double(audioValidate);
labelsValidate = [data(71:end)];


%Extraction of Audio Features
aFE = audioFeatureExtractor( ...
    "SampleRate",fs, ...
    "Window",hamming(round(0.03*fs),"periodic"), ...
    "OverlapLength",round(0.02*fs), ...
    "mfcc",true, ...
    "mfccDelta",true, ...
    "mfccDeltaDelta",true, ...
    "pitch",true, ...
    "SpectralDescriptorInput","melSpectrum", ...
    "spectralCentroid",true, ...
    "spectralSlope",true);

featuresTrain = extract(aFE,audioTrain);
[numHopsPerSequence,numFeatures,numSignals] = size(featuresTrain)
%Format for 1st layer of NN
featuresTrain = permute(featuresTrain,[2,1,3]);
featuresTrain = squeeze(num2cell(featuresTrain,[1,2]));

numSignals = numel(featuresTrain)

[numFeatures,numHopsPerSequence] = size(featuresTrain{1})
%Extract validation feature and format
featuresValidation = extract(aFE,audioValidate);
featuresValidation = permute(featuresValidation,[2,1,3]);
featuresValidation = squeeze(num2cell(featuresValidation,[1,2]));

%Define layers for Neural Network
layers = [ ...
    sequenceInputLayer(numFeatures)
    lstmLayer(50,"OutputMode","last")
    fullyConnectedLayer(numel(unique(labelsTrain)))
    softmaxLayer
    classificationLayer];

options = trainingOptions("adam", ...
    "Shuffle","every-epoch", ...
```

```
    "ValidationData",{featuresValidation,labelsValidate}, ...
    "Plots","training-progress", ...
    "Verbose",false);
%Train Network
net = trainNetwork(featuresTrain,labelsTrain,layers,options);

```

– MATLAB Code to train and validate the Emergency Vehicle Classifier:

```
clc
close all
clear
%Load recorded files
datafolder = fullfile(pwd);

ads =
audioDatastore(datafolder,'FileExtension',{'.wav','.mp4'},'IncludeSubfolde
rs',true,'LabelSource','foldernames');
%Separate the dataset for training and validation (80%/20%)
[adsTrain, adsTest] = splitEachLabel(ads,0.8);

adsTrain

trainDatastoreCount = countEachLabel(adsTrain);

adsTest

testDatastoreCount = countEachLabel(adsTest);
%Play sample data and reset
[sampleTrain, dsInfo] = read(adsTrain);
%sound(sampleTrain,dsInfo.SampleRate)
reset(adsTrain)
%Pre-process
fs = dsInfo.SampleRate;
windowLength = round(0.03*fs);
overlapLength = round(0.025*fs);
%Feature extraction for train Data
features = [];
labels = [];
while hasdata(adsTrain)
    [audioI,dsInfo] = read(adsTrain);
    audioIn = audioI(:,1);
    melC =
mfcc(audioIn,fs,'Window',hamming(windowLength,'periodic'),'OverlapLength',
overlapLength);
    f0 =
pitch(audioIn,fs,'WindowLength',windowLength,'OverlapLength',overlapLength
);
    feat = [melC,f0];


    label = repelem(dsInfo.Label,size(feat,1));
```

```
38
39      features = [features;feat];
40      labels = [labels,label];
41  end
42
43  %Standardize to remove bias
44  M = mean(features,1);
45  S = std(features,[],1);
46  features = (features-M)./S;
47  %Classifier specifications
48  trainedClassifier = fitcknn( ...
49      features, ...
50      labels, ...
51      'Distance','euclidean', ...
52      'NumNeighbors',5, ...
53      'DistanceWeight','squaredinverse', ...
54      'Standardize',false, ...
55      'ClassNames',unique(labels));
56
57  k = 5;
58  group = labels;
59  c = cvpartition(group,'KFold',k); % 5-fold stratified cross validation
60  partitionedModel = crossval(trainedClassifier,'CVPartition',c);
61
62  validationAccuracy = 1 -
    kfoldLoss(partitionedModel,'LossFun','ClassifError');
63  fprintf('\nValidation accuracy = %.2f%%\n', validationAccuracy*100);
64  %Generate Confusion matrix
65  validationPredictions = kfoldPredict(partitionedModel);
66  figure
67  cm = confusionchart(labels,validationPredictions,'title','Validation
    Accuracy');
68  cm.ColumnSummary = 'column-normalized';
69  cm.RowSummary = 'row-normalized';
70
71  %For Validation Data
72  features = [];
73  labels = [];
74  numVectorsPerFile = [];
75  while hasdata(adsTest)
76      [audioI,dsInfo] = read(adsTest);
77      audioIn = audioI(:,1);
78
79      melC =
    mfcc(audioIn,fs,'Window',hamming(windowLength,'periodic'),'OverlapLength',
    overlapLength);
80      f0 =
    pitch(audioIn,fs,'WindowLength',windowLength,'OverlapLength',overlapLength
    );
81
```

```
    feat = [melC,f0];
    numVec = size(feat,1);
    label = repelem(dsInfo.Label,numVec);
    numVectorsPerFile = [numVectorsPerFile,numVec];
    features = [features;feat];
    labels = [labels,label];
end
features = (features-M)./S;
%Testing and Validation of classifier
prediction = predict(trainedClassifier,features);
prediction = categorical(string(prediction));
figure('Units','normalized','Position',[0.4 0.4 0.4 0.4])
cm = confusionchart(labels,prediction,'title','Test Accuracy (Per
Frame)');
cm.ColumnSummary = 'column-normalized';
cm.RowSummary = 'row-normalized';
r2 = prediction(1:numel(adsTest.Files));
idx = 1;
for ii = 1:numel(adsTest.Files)
          r2(ii) = mode(prediction(idx:idx+numVectorsPerFile(ii)-1));
          idx = idx + numVectorsPerFile(ii);
      end
      %Generate confusion chart
      figure('Units','normalized','Position',[0.4 0.4 0.4 0.4])
      cm = confusionchart(adsTest.Labels,r2,'title','Test Accuracy (Per
File)');
      cm.ColumnSummary = 'column-normalized';
      cm.RowSummary = 'row-normalized';

```

– MATLAB Code for analysis with YAMNet pretrained network:

```
clc
clear
close
%Download and Load YAMNet
downloadFolder = fullfile(tempdir,'YAMNetDownload');
loc =
websave(downloadFolder,'https://ssd.mathworks.com/supportfiles/audio/yamne
t.zip');
YAMNetLocation = tempdir;
unzip(loc,YAMNetLocation)
addpath(fullfile(YAMNetLocation,'yamnet'))
net = yamnet;
%analyzeNetwork(net);
%ygraph = yamnetGraph;
%p = plot(ygraph);
%layout(p,'layered');
%allSirenSounds = dfsearch(ygraph,"Siren");
%ygraphSpeech = subgraph(ygraph,allSirenSounds);
%plot(ygraphSpeech)
```

```
%Read Audio file
[audioIn,fs]=audioread("emergency/EmergencySiren6.wav");
x = audioIn(:,1);

%% Create and Initialize
%sound(x,fs)
%identifiedSound = classifySound(x,fs)
%Classify Sound- call pre-buit function
[sounds,timeStamps] = classifySound(x,fs)
%Plot audio signal
t = (0:numel(x)-1)/fs;
plot(t,x)
xlabel('Time (s)')
axis([t(1),t(end),-1,1])
%To identify and segment and label timestamps
textHeight = 1.1;
for idx = 1:numel(sounds)

patch([timeStamps(idx,1),timeStamps(idx,1),timeStamps(idx,2),timeStamps(id
x,2)], ...
        [-1,1,1,-1], ...
        [0.3010 0.7450 0.9330], ...
        'FaceAlpha',0.2);
    text(timeStamps(idx,1),textHeight+0.05*(-1)^idx,sounds(idx))
end
sampleStamps = floor(timeStamps*fs)+1;
soundEvent = 3;
%Isolate timestamp and play audio
isolatedSoundEvent =
x(sampleStamps(soundEvent,1):sampleStamps(soundEvent,2));
%sound(isolatedSoundEvent,fs);
display('Detected Sound = ' + sounds(soundEvent))
%Generate Sound Table
[sounds,~,soundTable] = classifySound(x,fs);
sounds
soundTable
%Generate Result Table
soundTable.Results{end}
%Generate Word map
classifySound(x,fs)
```