Proceedings of the Institute of Acoustics

AWARENESS SYSTEM FOR HEADPHONE USERS

J W Kay University Of Salford, Manchester, UK
B M Fazenda University Of Salford, Manchester, UK, correspondence:b.m.fazenda@salford.ac.uk

1 INTRODUCTION

Hearing is arguably one of the most important senses in providing situational awareness and maintaining personal safety. There has been recently a marked increase of headphone use in public environments where, arguably, one's safety might be jeopardized by the occlusion and masking caused by listening to music over headphones. The Journal of Injury Prevention published an article on this issue demonstrating a clear association between the increasing ownership of portable media devices with headphones and pedestrian injuries from 2004 to 2011[1].

Some research has been devoted to this problem and a number of applications have been developed in an attempt to ameliorate the use of headphones in urban environment[2,3,4,5,6]. The aim of this project is to create a system that increases the auditory awareness of headphone users to their surrounding environment, whilst listening to music. The principle of operation uses the built in microphone on smartphones to capture the surrounding environment and perform an analysis to detect target audio events. The audio event detection developed here targets vehicle horns and household smoke alarms. Once the presence of one of these events is detected by the acoustical monitoring device, the external audio is passed through to the headphones in an attempt to increase the auditory awareness of the user. Audio event detection is achieved with a machine learning algorithm based on neural networks that has been trained to detect the specific sound events.

2 DESIGN

The platform of choice for implementation in mobile applications is the iOS. The development and training of neural networks took place in Matlab. Xcode was used for the development of the mobile app.

2.1 Feature Extraction for Machine Learning

Figure 1 shows a flow diagram demonstrating the training and implementation of the neural networks. This is processed via an offline algorithm in MATLAB used to calculate the neural network weightings. Essentially the MATLAB scripts and functions are placed into an exclusive folder that contains the sample database for training an individual neural network. This database must consist of a sensible set of WAV files that have been edited and separated into 'good' and 'bad' examples for the neural net to train on.

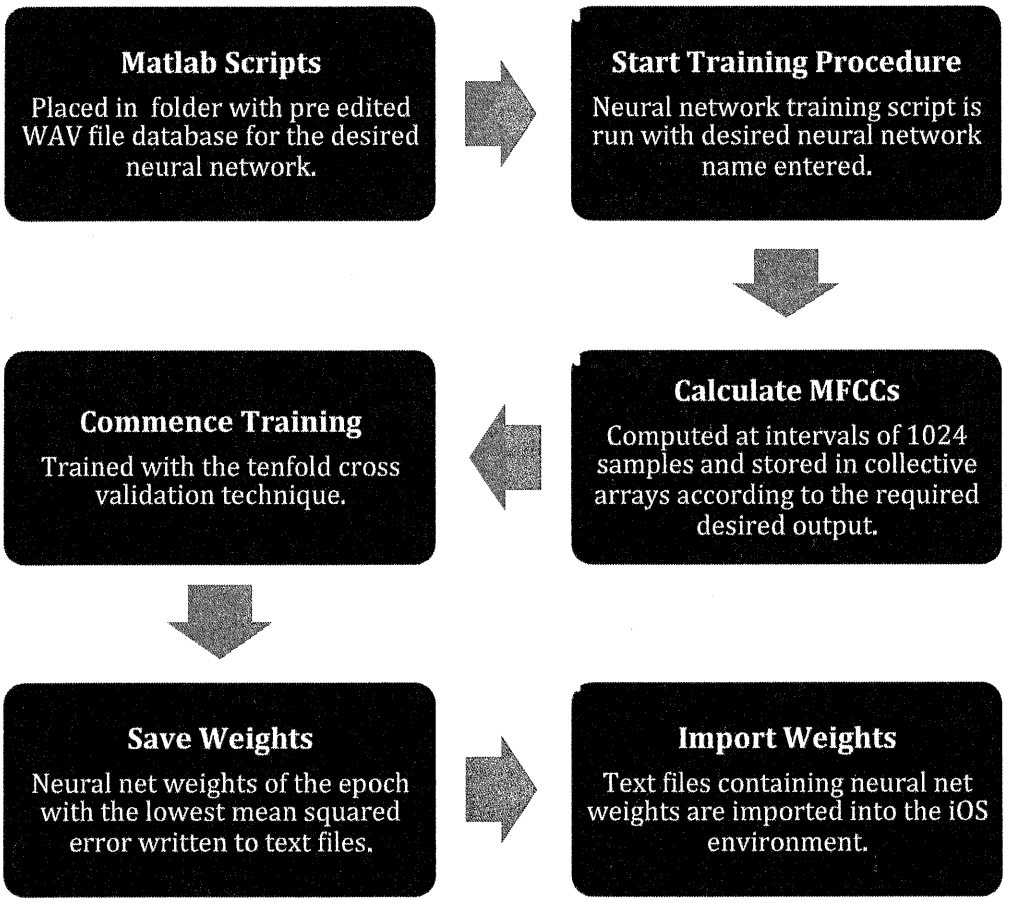

Figure 1: Generalised MATLAB offline procedure.

2.1.1 Mel Frequency Cepstral Coefficient Extraction

Most of the detection of events is based on the analysis of the Mel Frequency Cepstral Coefficients (MFCC). The perceptually based mel filter bank was achieved by designing triangular filters at the desired centre frequencies and stored into a data matrix. A WAV file is read into an array and then undergoes pseudo streaming analysis for MFCC extraction by moving through the data with a block/buffer size of 1024 samples. The 1024 sample size is a requirement to ensure that the offline processes done in the MATLAB environment synchronise with the iOS live application (where a buffer size of 1024 is enforced). Each block of data is processed with a Hanning window before the FFT is calculated.

The positive spectrum of the FFT output is then multiplied with the coefficients from the mel filter bank matrix. At this stage there are 20 numbers where each one is a filter output that corresponds to the sum of its filtered spectral components. These are then squared to represent the energy output of the filters and converted to logarithmic values.

A logarithmic representation of the filter bank energies compresses the dynamic range of the values and simulates a response which is more perceptually relevant. This also means that frequency estimates are less sensitive to input level variations (movement of microphone in relation to sound source)[7]. One difference from the standard way to calculate MFCCs is that the logarithmic

representation is squared. This simple calculation helps to desensitise the MFCCs to intrusive low frequency artefacts[8].

The discrete cosine transform (DCT) of these logarithmic values is then calculated. Only the first ten output values of the DCT are stored, which is enough to detect speech and represent a smoother spectrum[9].

2.1.2 Spectral Centroid Extraction

In order to achieve better accuracy by reducing false positive detection for the smoke alarm neural network, additional information about the acoustic environment was used alongside the network output. For this neural network, a clear choice was to use the spectral centroid calculation (Equation 1), since all household smoke alarms dominate the 3kHz frequency range[6]. The spectral centroid provides an indication of spectral balance and is determined by:

$$\text{centroid} = \frac{\sum f(n)x(n)}{\sum x(n)} \quad (1)$$

where $f(n)$ are the frequencies and $x(n)$ are the linear amplitudes (Patterson).

2.2 Neural Network Creation and Training

To enable flexibility when training for a particular desired sound, the MATLAB scripts and functions were designed in order to read the contents of the folder in which they are placed. A requirement for the training of the neural network is a pre-edited database of audio samples. A batch of files was recorded or generated to act as the training set for the neural network. This included segregating positive and negative examples for a particular target event. Once edited, these files were tagged with 'GOOD' or 'BAD' for neural network training. The neural network training procedure is displayed in Figure 2.

After a user defined neural network name, the script reads through all the audio files and calculates the MFCCs. The neural network layer size has been hardcoded to a hidden layer of six neurons[10]. The neural net weights and offset will default to a small random number with a maximum possible value of 0.25. If required, it is possible for the user to use previously trained neural net weights instead of random numbers; however this presents the risk of over-fitting the network to the training database[10,11].

For training, the N fold cross-validation technique was used. For this method, it is important that the training and test set for each fold gives a statistically balanced representation of the signals[12]. This means that the MFCCs for a particular/unique sound should never appear only once on each epoch of training. A simple solution to this was to use the function randperm and generate a numerically random integer array of equal length to the smallest array of MFCCs, with no repetition of numbers. This array is then used as a cell index to access the MFCC arrays, ensuring that particular MFCCs appear more than one in each epoch.

A loop of ten iterations is then in progress to begin the tenfold cross validation training. On each training fold, a test set of MFCCs is reserved corresponding to 10% of the database and since the location index was shuffled previously, this is done in consecutive blocks of the index.

At this stage in the program, a random tenth of the database has been removed and isolated for the purpose of neural network validation. The function randperm is then again used to generate the location index of the training sets before commencing a nested loop that runs through this index, training one positive example and one negative example. A positive/negative example at this point

consists of a set of the ten MFCCs previously computed rather than direct computation from the audio files.

After the loop has trained the network on the database in a random manner, it is then tested on the secluded validation set. The mean squared error (MSE) is then calculated and stored as a way to gauge the success of the neural network weights for each fold. Following the completion of a fold, the weights and offset are saved to text files corresponding to the training epoch.

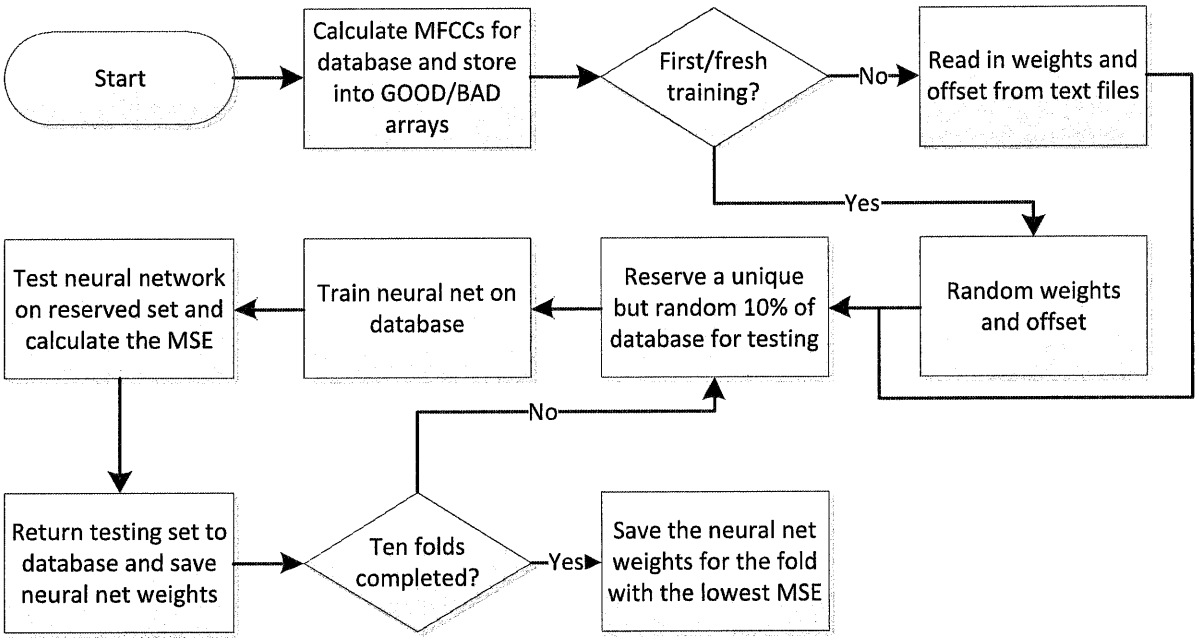

Figure 2: A flow diagram representing the general neural network training procedure.

This process iterates ten times and the fold which has generated the lowest MSE has its weights written to the final text files. This was easily achieved since a 'back pocket' technique was used – storing the weights generated for the most recent lowest result. The MSE over the training cycle for the horn neural network is shown in Figure 3.

Each network was trained on approximately 3000 sets of 10 MFCCs (10 per frame), consisting of equal amounts of 'good' and 'bad' samples for homogeneous statistical training. Although the success of a neural network is not guaranteed and rather dependent on the training dataset it is provided with, their flexibility and upgradable abilities are a great compensation for this.

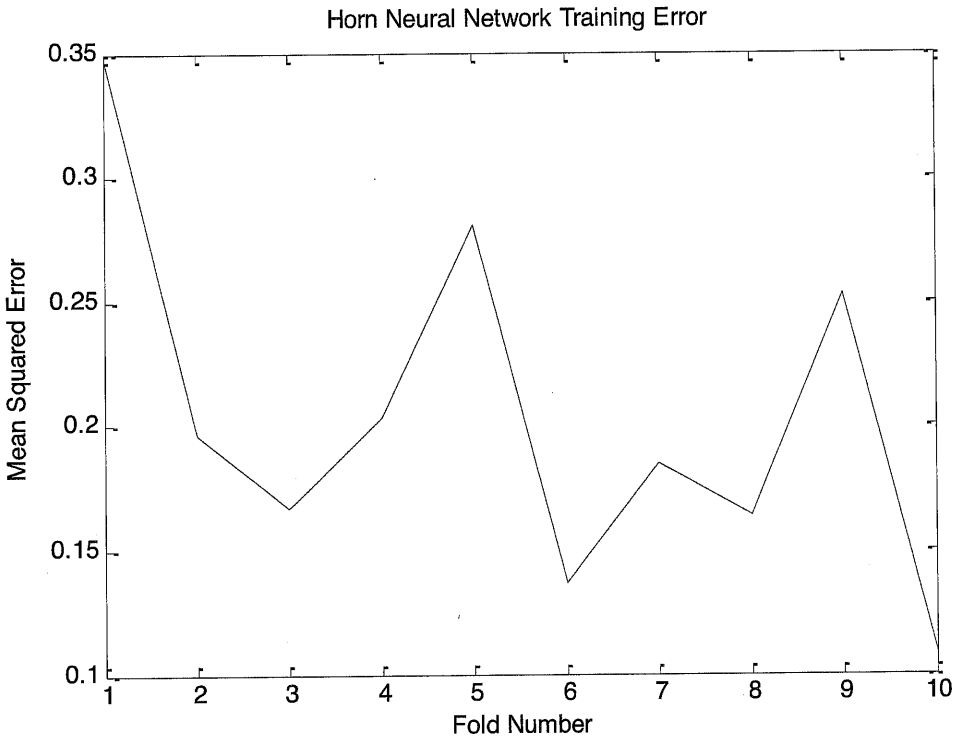

Figure 3: The MSE over the horn neural network training epochs.

3 IMPLEMENTATION IN IOS

3.1 General Implementation

The implementation of the neural network detection algorithm in the mobile app follows the same steps as described for implementation in Matlab. A generalised overview of the process structure for the iOS application is given here and represented in Figure 4.

After the application is initialized it starts retrieving the data from the active microphone, whether built-in on the handset or in the headphone cable, if plugged in. The application works in real time and continuously accesses the audio received from the microphone in frames of 1024 samples which is predetermined by the audio management setup in the iOS. This frame size corresponds to an analysis time window of approximately 23ms at a sampling frequency of 44.1kHz.

Neural network weights are retrieved from comma separated text files which have been stored in the general application directory. MFCC calculation is performed directly after the microphone data has been assembled into a float value array and the result is compared to the optimized outputs of the neural net. A gate delay has been set after neural net trigger to avoid spurious detections based on short events. This can be set by the user in the app as will be described below and the app will warn of a detected event only if the neural net is triggered for a period of time larger than the gate settings.

If the neural net has been triggered successfully, the application has to share audio output with an existing audio session (such as the music player). At this stage a new routine instantiates a new buffer of output samples which is populated with data from the microphone and the music player mixed together.

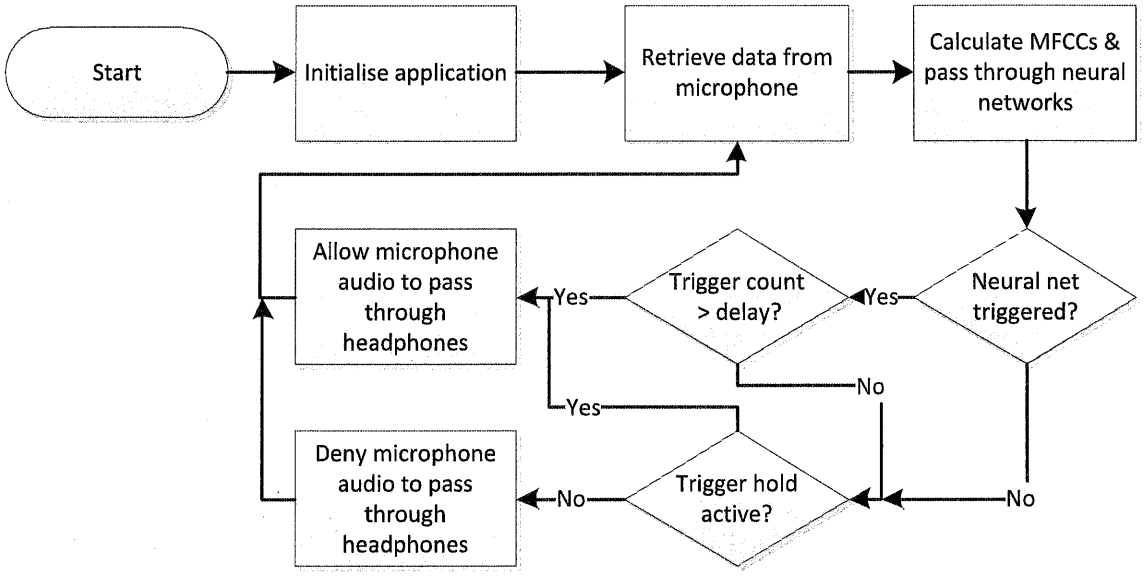

Figure 4: A flow diagram representing the general iOS application structure.

3.2 User Settings in the Application

Aspiring towards a more computationally efficient app, an effort was made to try and remove redundant processing. A user can switch off the detection of audio events (Horn or Alarm) in the settings menu, which disables the calculations for a given neural net. The user can also set a threshold for MFCC RMS values, which will disengage neural network calculations when idling or in the presence of acoustic information that is evidently dissimilar to the target sound, i.e. switching off detection of tram horns when at home.

Due to the dynamic and sporadic nature of environmental sound, there will be times when the neural nets fire without the presence of the correct acoustic stimuli. The key feature of the target sounds is the consistency it has over time. A gate with a primitive hysteresis feature was thus developed to monitor the outputs of the neural networks. This means that the gate appreciates how the neural networks previously behaved for a finite length in time[13]. Once a neural net output exceeds a threshold for a given number of consecutive frames, the application then deems it probable that it has indeed been acoustically stimulated by the sound of interest and initiates the mixture of external audio with media audio. At such times a hold count is also activated to ensure the sound is held open for a sensible period also removing any potentially irritating volume fluctuations. This hold characteristic additionally aids the hysteresis feature because when a neural net detection drops below the threshold, it may be picked up again with successive neural net firing and reset the hold count. Signals with a long duration may encounter this scenario and with such a component, be less likely to drop in and out of confirmed detection and audio mix. The values for gate delay and hold are default to 209ms and 1.161s respectively. This is a customisable way to reduce the number of false positive detections but more study is required to identify optimal values for particular events. The application also provide an option to pause the media player audio instead of merely mixing the external sound with the media player when an event is detected.

3.3 Application Performance Feedback and Learning

There is a bottleneck in the implementation which is related to its training phase, where the neural net will only behave as well as it is trained. During the development of this project, the audio sample databases for each neural network was limited to what was realistically possible to acquire prior to implementation. For this reason a feedback element was implemented where the application has the possibility of learning from its mistakes via user feedback. There is a selectable 'learning' mode which instead of providing an auditory and written alert to the user, has an active running buffer storing two-second recording data for any detections. If detection is triggered, the user is presented with an alert menu to select its validity, i.e. correct/incorrect. Both the section of audio detected and user validation are saved as a text file on the device. These files take up an almost unnoticeably small amount of storage, so there is little concern for the limit on feedback quantity. The text files can then be exported from the device via a simple upload button, which automatically appends the files to an email and sends to a pre-selected account.

4 RESULTS AND DISCUSSION

4.1 Introduction

For the validation tests in Section 4.2, it should be noted that only one neural network was active at any one time. Whilst the application allows simultaneous neural network activity, the inappropriate nets for each test were deactivated to give a detection rate true to the target sound.

4.2 Detection

For the following validation tests, testing audio was captured previously onto the device simply using the onboard recording app. This way, any artefacts introduced by the recording apparatus, its hadndling or location were taken into account in the recorded audio file. The recordings taken with the device, for validation purposes, were then played back from a computer into the headphone awareness application via a mini jack adapter that assumed the role of an inline microphone on a headset.

Each test consisted of two validation recordings, one where the device was held freely in the environment without any obstructions to the microphone (unobstructed) and another where the device was placed inside the pocket of an item of clothing (obstructed). The latter test took place to give an indication as to what impediment clothing material obstruction has on the performance.

4.2.1 Horns

The horn validation recordings were made at a location in Manchester Piccadilly Gardens, where there was approximately a 50 metre radius around the device that had a direct line of sight to three main tram line tracks. The tracks passed by the device closely and encompassed a diverse set of angles with respect to the device. Each recording was ten minutes in length.

Unobstructed

A horn detection timeline is illustrated in Figure 5 to visually flag the responsiveness of the application to the environmental audio when the device is located outside of clothing. Initial observation of this graph indicates that the application is not impractically over sensitive to acoustic stimulation, especially since a ten minute time period in a busy city centre is displayed.

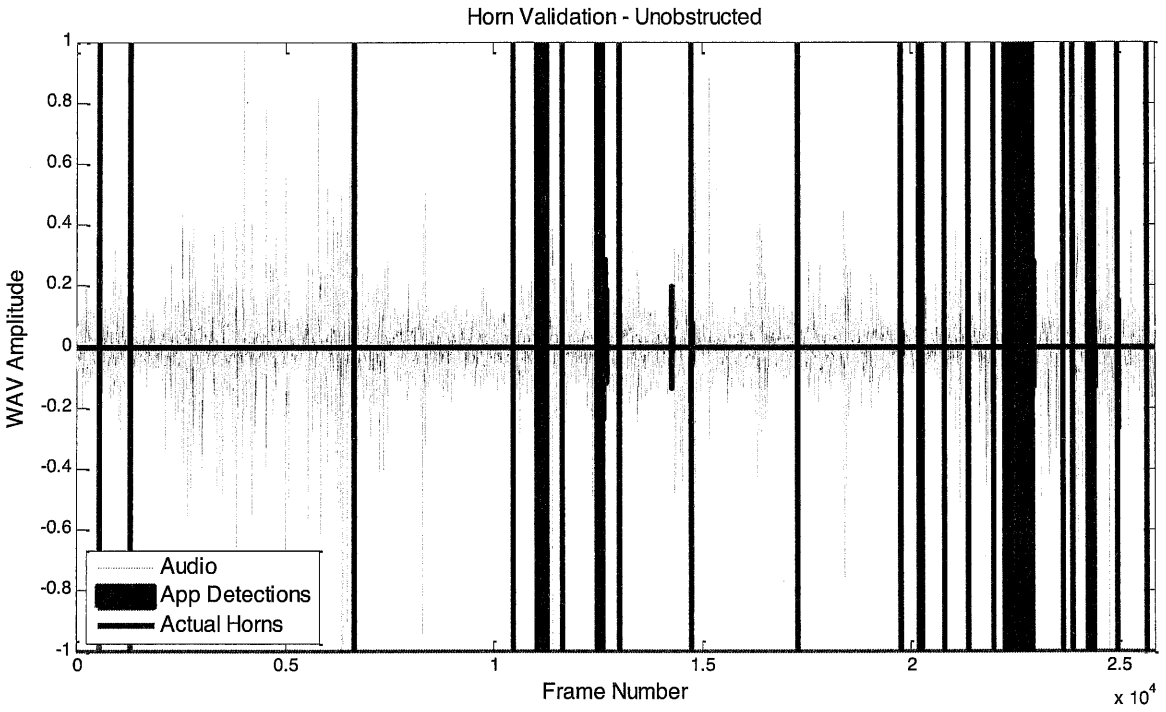

Figure 5: Visual representation of the application detections (red) compared to the actual horns (blue) and general environmental sounds (pale blue).

Out of the 29 detections, 17 were actually correctly identified as being horns. This gives a success rate of 58.6% which is quite low, especially when comparing to the results of another proposed traffic warning system, with success rates of 94%[5].

The application only missed the detection of one horn, which means that the success rate is affected mainly by false positives. The horn event missed was of low pitch, with spectral frequency dominance at around 430Hz but, more significantly, of duration 150ms, which is below the 209ms gate delay and hold setting described in the user settings section. It would, of course, be possible to alter the sensitivity values of the application but, arguably, this can be seen as unnecessary since horns with a very short duration do not tend to couple with a sense of urgency[5].

The majority of the false positive detections were triggered by tonal tram track screeching as the trams travelled around bends. These false positives can be explained by observing the comparative spectrograms between a standard tram horn and a tram track screech, as shown in Figure 6.

The frequency spectrum of a typical tram horn is dominated by energy in the 500Hz and 1500Hz regions. This is consistent through time and gives a tonal characteristic to the sound. Similar frequency qualities are also present in the tram track screech spectrogram, for example from 1.2 seconds onwards. Here the audio consists mainly of energy at approximately 750Hz and 1750Hz. This lasts steadily for more than 400ms which is ample time for the application to analyse and act upon the characteristics of this audio. The spectrogram also indicates that use of the spectral centroid frequency to try and negate the false positive detection would be ineffective since both sounds occupy a very similar spectral arrangement.

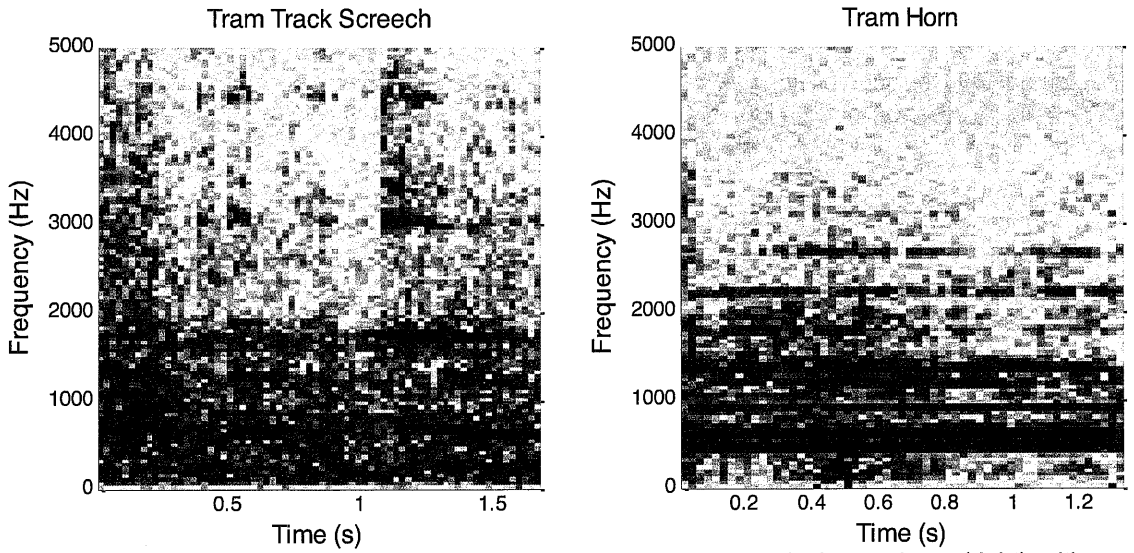

Figure 6: Spectrograms to compare the frequency content of a typical tram horn (right) with a typical tram track screeching sound (left).

Obstructed

Figure 7 shows a section of audio with superimposed detection of events by the neural network. The presence of material obstruction to the microphone evidently decreased the performance success of the application. Out of 31 detections 15 were verified as being correct, giving a success rate of 48.4%.

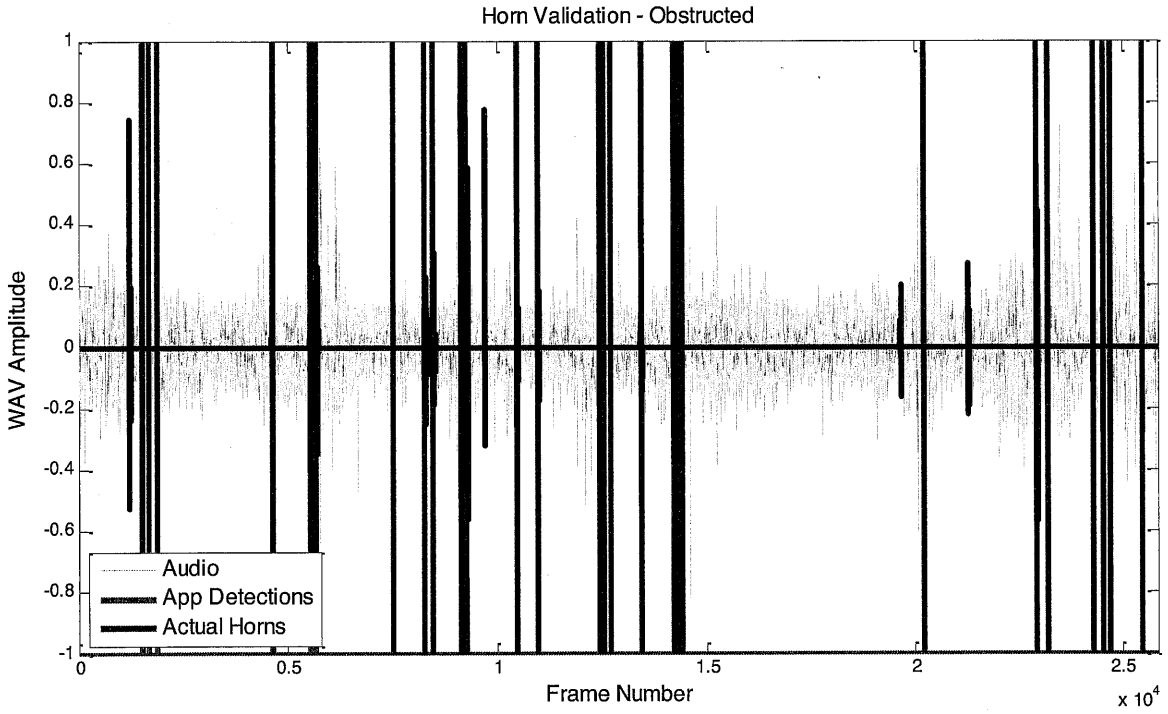

Figure 7: Visual representation of the application detections inside of clothing (red) compared to the actual horns (blue) and general environmental sounds (pale blue).

Again the bulk of the false positive alerts originated from the track screeching of passing trams. This time however, the number of missed horns increased to 4 but the same reservations as to whether these horns were actually important should be applied. The only horn that was not either too short in length or too distant was at frame 1214, where its pitch was unusually low (~400Hz). An update to the neural network training database with this horn would potentially remedy this overlook.

The most impressive aspect of these results is the ability of the application to detect horns in a noisy and busy city environment. Many of the true horn detections were at a distance that can be considered to be of no threat to the user, showing a great deal of sensitivity to this application. This is not something that should be regarded as a hindrance because it would be an almost trivial matter to enforce a threshold level where only horns within a given radius, or with a prescribed level, are likely to be detected.

4.2.2 Smoke Alarms

The two smoke alarm validation recordings were each five minutes in length and took place in a household with common background noises such as speech, radio and television. Every smoke alarm sample was emitted from a ground floor kitchen whilst the detection device changed locations in the house to simulate different receiver distances, two on the ground floor, two on the first floor and four on the second floor. Each time the device was physically in another room but with connecting doors open to be consistent with the recordings located on the second floor. This was important to allow the sound to physically propagate to such a room.

Unobstructed

The application detections are plotted along with the background audio and actual smoke alarm audio in Figure 8. Brief assessment of this graph shows a success rate of 100%.

As a guideline in terms of application sensitivity, the smoke alarm signals at the worst case scenario (receiver on second floor) were on average merely 6dBFS louder than the background noise. To achieve this sensitivity the neural network triggering values were slightly amended to a 0.7 net firing threshold and a 6 frame (~140ms) successive trigger.

Obstructed

A success rate of 87.5% for detecting smoke alarms when the device is inside a piece of clothing is displayed in Figure 9. Similarly to the unobstructed case, there were no false positive detections caused by varying background noise. Instead the reduction in success rate stems from a missed smoke alarm (frame 10166), where the signal at that time was merely 3dBFS larger than the surrounding environmental noise.

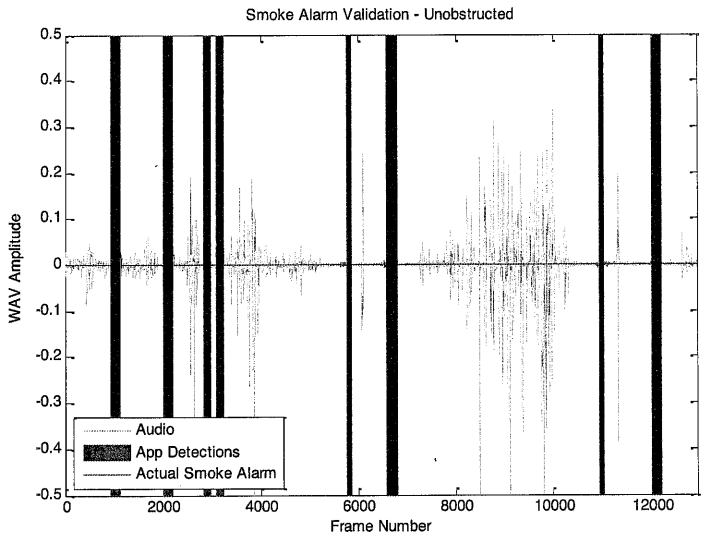

Figure 8: Visual representation of the application detections (red) compared to the actual smoke alarms (blue) and general environmental sounds (pale blue).

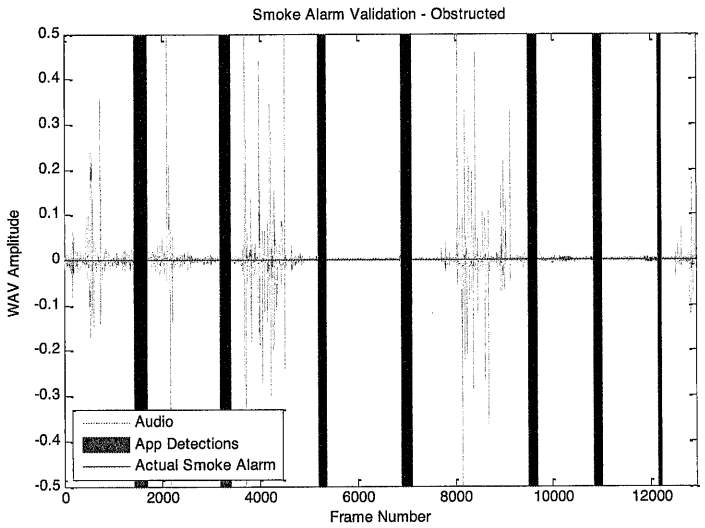

Figure 9: Visual representation of the application detections inside of clothing (red) compared to the actual smoke alarms (blue) and general environmental sounds (pale blue).

The time taken before a smoke alarm sound is confirmed becomes an issue when the device is placed inside clothing, compared to when it is not. The maximum length of time before a confirmed detection outside of clothing was 2.49 seconds, contrasting a much larger maximum of 6.46 seconds when the device was inside clothing. In a real emergency situation a smoke alarm is likely to continue sounding for a longer period than these delay times. For this reason, the time taken before a smoke alarm is detected is currently not of any great implication.

4.3 Hardware Performance

The application is able to run fluently on a fourth generation Apple iPod. Other devices have not been tested at this time but their hardware specifications match/outperform this iPod. Previous generations of the iPod cannot run the application because of the absence of an inbuilt microphone.

The 'Instruments' analysis tool in Xcode permitted detailed analysis of resource usage and activity. The application utilises 6.75MB of the available 256MB of RAM, which is not an area of concern since this figure is so low in comparison. CPU performance however is a different matter with the application using approximately 56% to calculate the MFCCs. With the inclusion of neural network calculations this figure only increases by approximately 1% to 2% per neural network in force, showing their impressive computational efficiency. Issues in performance occasionally arise when an alert message box is displayed on screen, where the CPU usage increases to ~70% but since these are for very short periods of time this is not of great importance at this stage of development.

The CPU usage consumes the battery life which may be an area of concern for the application's quality of experience. Fundamentally this is due to the calculation of the MFCCs. An area for improvement that is worth investigation is to code the DCT used in the calculation of the MFCCs more efficiently and remove the redundant repetition of calculations in the loops. This is much like the efficiency difference between the Discrete Fourier Transform and the Fast Fourier Transform.

5 CONCLUSIONS

The aim of this project was to create an application for portable media devices that increases the awareness of headphone users to external sound events. Mel Frequency Cepstral Coefficients were used as the foundation to detect acoustical characteristics of the events, which in turn were analysed by neural networks.

The event detection method produced promising results when tested for validity in real life situations. The success rates are displayed in the following table:

Table 1: Summary of the detection success rates.

Detection Sound	**Success Rate** (%)	
	Unobstructed	*Obstructed*
Vehicle Horns	58.6	48.4
Household Smoke Alarms	100.0	87.5

Whilst the vehicle horn success rates may indicate an underwhelming performance of the detection algorithm, no important horns were actually missed by the application. Instead the reduction in the success rate comes from the false positive detections. In particular, these originated from the loud tonal screeching of tram tracks which rather coincidently contained very similar spectral components in the frequency domain to the target event. It is uncertain at this stage whether an updated neural network training database will provide a solution to the unwanted detection of these sounds. Inclusion of car horns is an obvious upgrade to this neural network.

Encouraging results were also obtained from the household smoke alarm validation testing, where all but one event, across the varying source-receiver separations, were detected. This missed smoke alarm was at a maximum possible separation of two floors in the house with the device placed inside clothing. These results, coupled with the absence of false positive detections, demonstrate a powerful sensitivity in the recognition algorithm that may be further exploited with future refinement.

The app was also developed with future updates in mind, to make a more efficient use of processing. In particular, the option of switching off defined neural nets aims to provide a future option where users can set scenes, such as 'outdoors', 'indoors'. Ultimately, these scenes can be toggled automatically once the app acquires information about the users location from the GPS in the device.

The implementation of a feedback function allows the application to collect two-second audio samples of event detections, along with a validation of the detections determined by the user. This alleviates some of the labour involved in collecting samples for the neural networks to train on and provides the detection algorithm with a promising future, since updates on the neural networks can be obtained from user feedback which, given the potential nature of crowd sourcing in the application, can be very efficient. The application has been designed to expand other neural networks for detection of additional event categories. Other desired target sounds, which may be regarded as beneficial to users, can be trained offline in MATLAB and then deployed to the live application with minimal coding modifications.

Whilst there is much modification and development left to do in further work, the project has demonstrated the capability for an environmental auditory monitoring system to execute in real time on portable devices.

6 REFERENCES

1. Lichenstein, R., Smith, D., Ambrose, J., & Moody, L. (2012). Headphone use and pedestrian injury and death in the United States: 2004e2011. Injury Prevention. 18 (5), p287-290.
2. Basu, S., Clarkson, B., & Pentland A. (2001). Smart headphones: enhancing auditory awareness through robust speech detection and source localization. Acoustics, Speech, and Signal Processing. 5(1), p3361-3364.
3. Essency. (2011). Awareness! The Headphone App. Available: http://www.essency.co.uk/awareness-the-headphone-app/. Last accessed 20th Jan 2013.
4. Wang, T., Cardone, G., Corradi, A., Torresani, L., & Campbell, A.T (2012). WalkSafe: a pedestrian safety app for mobile phone users who walk and talk while crossing roads. In Proceedings of HOTMOBILE.
5. Yoon S., Lim Y., Lim H., & Kim H. (2012). 421 Architecture of Automatic Warning System on Urgent Traffic situation for Headphone Users. International Journal of Multimedia and Ubiquitous Engineering. 7 (2), p421-426.
6. Ellis, D. P. W. (2001). Detecting Alarm Sounds. Available: http://academiccommons.columbia.edu/download/fedora_content/download/ac:148927/CONTENT/crac01-alarms.pdf. Last accessed 15th April 2013.
7. Jurafsky, D. (2009). Speech Recognition, Synthesis, and Dialogue. Available: http://www.stanford.edu/class/cs224s/lec/224s.09.lec9.pdf. Last accessed 10th April 2013.
8. Tyagi, V., & Wellekens, C. (2005). On desensitizing the Mel-Cepstrum to spurious spectral components for Robust Speech Recognition. In International Conference on Acoustics, Speech, and Signal Processing . Philadelphia, March 18-23, 2005. p529–532.
9. Holmes, J., & Holmes, W (2001). Speech Synthesis and Recognition. London: Taylor & Francis.
10. Cary, W. S. S. (2002). Neural Network FAQ. Available: ftp://ftp.sas.com/pub/neural/FAQ2.html. Last accessed 5th Feb 2013.
11. Hertz, J., Krogh, A., & Palmer, R. G (1991). Introduction to the Theory of Neural Computation. California: Addison Wesley Publishing Company.
12. Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In IJCAI'95 Proceedings of the 14th international joint conference on Artificial intelligence. 1995. San Francisco: Morgan Kaufmann Publishers Inc. p1137-1143.
13. Zölzer, U (2002). DAFX:Digital Audio Effects. West Sussex: John Wiley & Sons Ltd. p102-104.