Tampere University

Ali Gohar

ROBUST SOUND EVENT DETECTION IN REAL WORLD ENVIRONMENT

Master of Science Thesis
Information Technology and Communication Sciences
Examiner: Prof. Tuomas Virtanen and Toni Heittola
November 2020

ABSTRACT

Ali Gohar: Robust Sound Event Detection in Real World Environment
Master of Science Thesis
Tampere University
Master in Information Technology
November 2020

Sound present in our daily life contains very useful information and can help in solving multiple problems. Sound event detection (SED) in real life can be used to build products for security, safety and human convenience. Deep neural networks are quite common and perform very well for such tasks. A lot of work has already been done using deep neural networks and recurrent neural networks to tackle various problems.

Considering the informative events for nurses in home care centers and nursing homes, we finalized four target classes (door, water tap, yelling, human fall) to be focused in this thesis. These sound events can also be used in other places e.g home security and public places. Considering other frequent sounds in the daily environment, we took speech, music and coughing as non-target classes. Despite four target classes all other sounds will be considered as background noise and will be ignored. This thesis aims to synthesize a dataset containing all these sound events and train various models to achieve higher accuracy for target classes.

Target and non-target class data along with ambience sounds are collected from the internet to create real-life soundscapes. Soundscapes are created in four folds to use the cross-validation technique during training of the model. To ensure the privacy of the speaker, we decided to extract audio embeddings from these soundscapes and use them for further process. Three different types of audio embeddings (openl3, kumar2018, zhao2020) are extracted and compared to find the optimal one for this task. Supervised learning acoustic detection models are trained and validated on this synthesized data to find the finest model for these events.

We found that embeddings extracted from Zhao2020 and provided to various classification models outperform the other two types of embeddings extraction models on our synthetic dataset. We achieved the best results by combining a recurrent neural network with max polling.

Keywords: Sound Event Detection, Recurrent Neural Network, Soundscape Synthesis

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

PREFACE

The conceptual knowledge acquired by the Data Engineering and Machine Learning program is best manifested in this project. I sincerely express my gratitude to my supervisor Professor Tuomas Virtanen for providing me an opportunity to work on this project with a prestigious research group. I would like to thank Toni Heittola and Shuyang Zhao for providing technical support during the implementation process. Moreover, my sincere thanks go to the complete team of the project for discussion and guidance.

Last but not the least, I could not have achieved these things without the support of my family, so thanks to my family for assisting me in my whole life.

Tampere, 30th November 2020

Ali Gohar

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AUC	Area Under Curve
BCE	Binary Cross Entropy
CASA	Computational auditory scene analysis
CRNN	Convolution Recurrent Neural Network
DCASE	Detection and Classification of Acoustic Scenes and Events
DNN	Deep Neural Network
FFNN	Feed Forward Neural Network
GRU	Gated Recurrent Unit
IoT	Internet of Things
LSTM	Long Short-Term Memory
MFCC	Mel-frequency Cepstral Coefficients
ML	Machine Learning
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
SED	Sound Event Detection
SNR	Signal to Noise Ratio

1 INTRODUCTION

Our daily life environment contains multiple complex sounds that are generated from various sources. These sounds have very much variation in terms of the sound source, loudness level, temporal behavior and location as shown in Figure 1.1. The sound could be generated by humans, animals, machinery and nature at different loudness having a wide range of background to event ratio. The sound source could be stationary or moving depending on the situation, so the same sound source could produce different sounds on the base of the movement. Transient, intermittent and continuous are different temporal behavior of sounds and all these sounds are present in our daily life. Gun shot is a transient sound that occurs for a very short time, water tap running could be considered as continuous sounds and footsteps is an intermittent sound and occur periodically.

The human auditory system is trained and automatically works to recognize and put various sounds in a different category. We gather tremendous information from various sound events and use that in our daily routine for taking different decisions. If we want to develop a smart system that could interact with humans and provide useful information, then the system should be able to recognize the surrounding sounds. Computational auditory scene analysis (CASA) [1] is a field to use these sounds and convert them into meaningful information with the help of machines. Source separation, sound event detection and acoustic scene recognition come under the umbrella of CASA.

Source separation is a task to separate a specific event from mixed audio (e.g vocal from music), sound event detection is to determine and classify each sound in a mixture and acoustic scene recognition is to only categorize the environment (e.g park, office, home). These task could be used separately or jointly depending on the situation and problem. For example, in the case of SED, acoustic scene recognition can narrow down the output results and provide high accuracy [2]. Similarly, source separation can also be used jointly with SED to first separate the specific target event and then classify the event [3]. In this thesis, we will be focusing only on sound event detection in a real-world environment.

Sound event detection could be done in two different ways: recording-level detection and frame-level detection. Recording-level detection is also known as audio tagging where we are not interested in the onset and offset of the sound event, but we only want to determine the sounds in the recording. Frame-level detection can determine the onset and offset of the sound event to pinpoint the exact location of the event. These two tasks could be used jointly or standalone depending on the situation. During my work of this thesis, I would be focusing on solving the audio tagging problem but we will also consider frame-level detection.

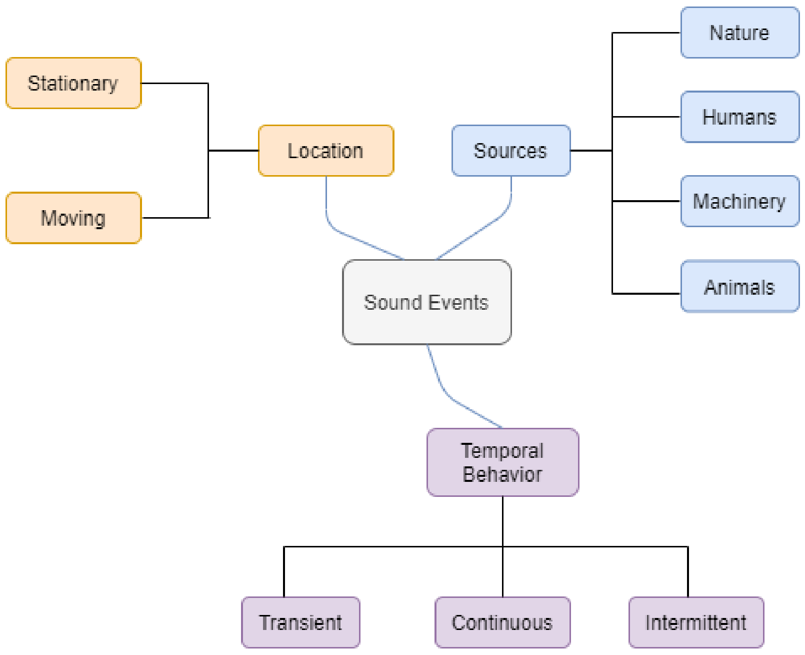


Figure 1.1. *Categories of sound events*

1.1 Motivation

The internet of things (IoT) gave birth to a new era of smart cities to assist mankind and it opens the doors for various new technologies. Machine learning (ML) is one of the hot topics these days and running an ML algorithm on edge devices e.g Raspberry Pi and Jetson Nano, becomes a common practice. By using these technologies we can provide security, convenience, comfort and entertainment to the people. Sound event detection can also be used in these applications to provide audio surveillance, urban sound analysis, healthcare monitoring and multimedia event detection. At the simplest level, SED could provide information about sound events in a specific environment, and by using that information we can produce semantic interpretation about the activities in that environment.

The simple architecture of sound event detection is shown in Figure 1.2, where the microphone will record the sounds and through some API's send the data to the server for further processing. The server will receive the audio and detects various sound events on the base of the machine learning algorithm, and then this information will be used for further data analysis. The data analysis part will produce semantic interpretation on the base of events and produce triggering alarm and notification for users.

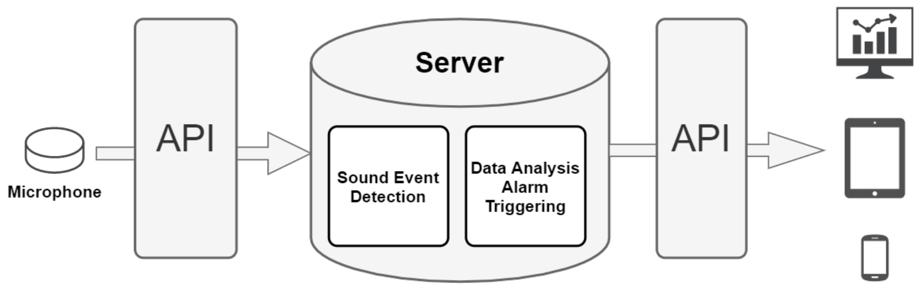


Figure 1.2. *SED system architecture*

Computational analysis of sound scenes and events [4] chapter 12 gave examples of using SED in a real-life environment, let us discuss some of them here. Regarding security and safety, **window break** in an empty apartment could be an indication of an intruder in the home and can be tackled by the SED system. SED system will recognize that sound and play a previously recorded audio, turn the light on and send the notification of activity to the owner. Similarly, **smoke alarm** in an unoccupied apartment could be a great threat but can be handled easily by immediately sending an alert to the owner.

Concerning comfort and convenience, **baby cry** at night can disturb both parents at night and this issue can be solved by SED. The sound will be detected by the SED system and soothing lullaby will be played by the sound system. In case the baby does not sleep again and keeps crying for a certain interval then the SED system will send a notification to one of the parents without waking the other. In the same way, **dog bark** in an empty home can be recognized and notified from the SED system. The owner can talk to the dog through intercom or can also check the home camera feed to verify everything is fine.

The elderly world population is increasing day by day and those people may not get a healthier life causing concern for government and society [5]. A sound event detection system can also play a vital role in health care monitoring to timely tackle any accident with patients. Human fall, call for help and other daily activity sounds can be recognized by the SED system and save the older people from some serious problems. Concerning human fall seriousness, Droghini et al. create a dataset of human fall and propose a method for one-shot human fall detection in their paper [6].

Audio-based multimedia event detection is also an interesting application, where you can get useful information from sound events to explain the activity or environment in the video. Currently searching a video is based on text instead of actual events in that video. Audio-based multimedia event detection can solve this problem and open a new era of searching techniques.

All these interesting applications of sound event detection motivates me to work in this area and try to solve one of the problems. Considering the seriousness and importance of different applications, I choose to work on the healthcare monitoring problem. During this thesis, I will focus in recognizing sounds related to healthcare monitoring.

1.2 Objective

The objective of this thesis is to design a supervised learning model to detect polyphonic sound events that could be useful in health care monitoring. From the literature review as well as considering informative events for nurses, we finalized four target classes (door, water tap, yelling and human fall) and three non-target classes (speech, music and coughing). Despite four target classes all other sounds will be considered as background noise. For training and evaluating the supervised learning model, we need a huge amount of annotated data containing our target and non-target classes. Getting such data is a problem and if we record this type of data then annotation of that data will be a big task, so we decided to synthesize the dataset.

I will collect the individual samples of all required classes as well as ambience from the internet and annotate those samples. The samples will be divided into four non-overlapping folds, so that it can be used for training and evaluation. These individual samples folds will be used to synthesize the real-life soundscapes dataset according to our requirements. The benefits of synthesizing data will be a huge amount of annotated data with a controlled number of target and non-target classes. We will try a python-based sound synthesizing tool, Scaper [7], to produce soundscapes similar to real life.

Computational analysis of sound scenes and events [4] chapter 12 also gives information about ethical issues in privacy and data protection. Privacy is one of the important things in any project and especially if we want to use real-life recordings. Audio data collection in someone's home or any other place for research or commercial purpose falls under the data protection laws and we need to consider it seriously. Normally, people do not want to share their audios with others, so converting the audio to some privacy-preserving features (embeddings) and use that for SED is an important task. We will consider various embeddings models (Openl3, Kumar2018, Zhao2020) in this thesis to tackle this Privacy preserving issue.

For selecting a model, I will mainly consider deep neural network (DNN) and recurrent neural network (RNN). A simple DNN model will be taken as the base model for SED and then we will continue towards more advanced models to improve the performance. I will do experiments with different models as well as various embeddings, mentioned above, to see which embeddings model working good with which classification model on our synthetic dataset.

As discussed above sound event detection could be done into two different ways, and I will experiment considering both frame-level and recording-level. For the audio tagging problem, one possible approach could be using RNN where all frames will be sent to RNN in sequence and then the prediction is made on the basis of last frame. As I will be synthesizing the data with strong labels, so I will do experiments by using both strong and weak labels during training to see the difference in both approaches. Our main focus in this thesis will be to solve the audio tagging problem and make a prediction for a one-minute recording. The embeddings for a complete recording will be extracted and given to the sound event detector that will make frame-level prediction. The results from the frame-level could be combined to form a recording-level prediction output as shown in Figure 1.3.

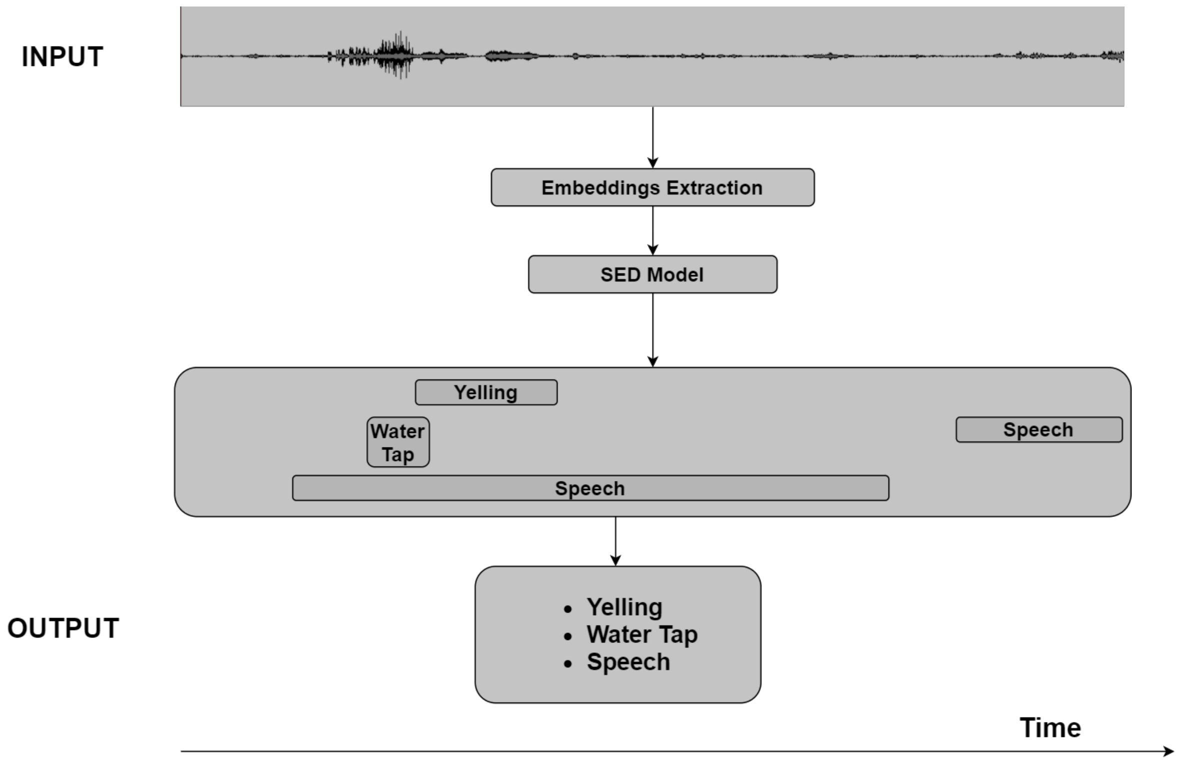


Figure 1.3. *Audio tagging for polyphonic sound event detection*

2 BACKGROUND

2.1 Deep Neural Networks

2.1.1 Feed Forward Neural Network

Feedforward neural networks also known as multilayer perceptron are considered as deep learning foundation and get popularity in the last decade. Frank Rosenblatt invented the perceptron algorithm in 1958 that become the base for deep learning models. Recurrent neural networks and convolution neural networks are complex cases of feedforward neural networks and are mainly used for supervised learning. Simple neural networks consist of input, output and hidden layers with multiple neurons in each layer. It takes an input in the form of a fixed-length vector multiply that with some weights and add a bias to produce an output. The output of each neuron is calculated as

$$y = \sigma(w^t x + b), \tag{2.1}$$

where x denotes the input vector, w is the weight vector, b represents the bias and σ stands for non-linear activation function. Non-linear activation functions are used to create the mapping between input and output of the neural network. Sigmoid, softmax, ReLU and tanh are commonly used activation functions and their equation are shown in the Table 2.1.

Table 2.1. *Activation functions*

Activation Function	**Equation**
$Sigmoid(x)$	$\frac{1}{1+\exp^{-x}}$
$Softmax(x)$	$\frac{\exp^{z_i}}{\sum_{j=1}^{k}\exp^{z_i}}$
$Tanh(x)$	$\frac{\exp^{x}-\exp^{-x}}{\exp^{x}+\exp^{-x}}$
$ReLU(x)$	$max(x, 0)$

The choice of activation function at the output layer mainly depends on the task. Sigmoid and softmax both give output between 0 and 1, which is considered as probabilities of classes. Sigmoid is used for multilabel tasks where several classes can occur at the same time while softmax is used for multiclass classification when only one class is active.

Training of neural networks is finding optimal parameters (weights, bias) for each layer to minimize the loss. If x is the input to network and y-pred is the predicted labels and y-act is the ground truth labels then the loss is calculated as loss (y-pred, y-act). Loss is minimized by using various kinds of techniques and the gradient descent algorithm is a favorite one.

Gradient descent is an iterative process and used to find the minimum of a function by updating the value. There are different variants of gradient descent e.g batch gradient descent, stochastic gradient descent and mini-batch gradient descent. In batch gradient descent the weights of layers are updated after showing the complete training dataset to the network and this is called one epoch. This method is a slow process and learning is also slow in that case, but it is not computationally expensive. Stochastic gradient descent updates the weights of layers after every example, and it makes this method to learn faster by updating the weights more frequently. For mini-batch gradient descent we divided the whole dataset into mini-batches and weights are updated after passing each mini-batch. Based on dataset size and task we choose one of the technique to train our system.

2.1.2 RNN

In feedforward neural network the information flows only in one direction without any feedback. The neurons in each layer do not provide any information to other neurons in the same layer. That means, in the case of sequence input, every output is only dependent on that moment input without considering the contextual information and it can be seen in Figure 2.1.

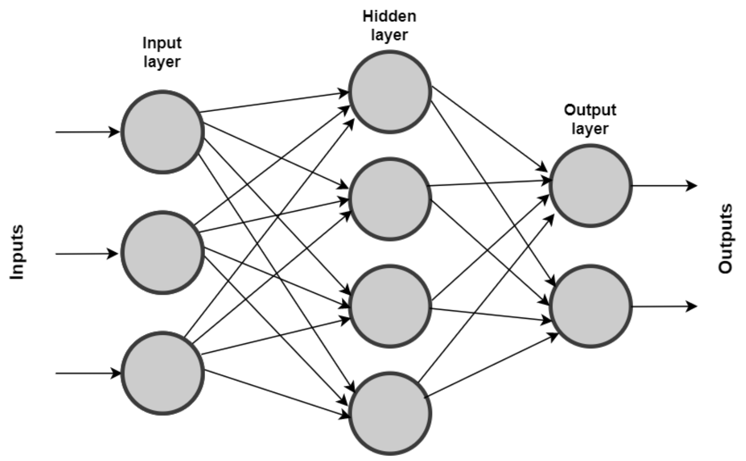


Figure 2.1. *Feedforward neural network*

Contextual information could be very informative and useful in some of the machine learning tasks such as speech recognition and audio detection. To use contextual information in a sequence input recurrent neural network (RNN) play a vital role. In RNN, the output of the current instance depends on current input as well as previous output as shown in Figure 2.2.

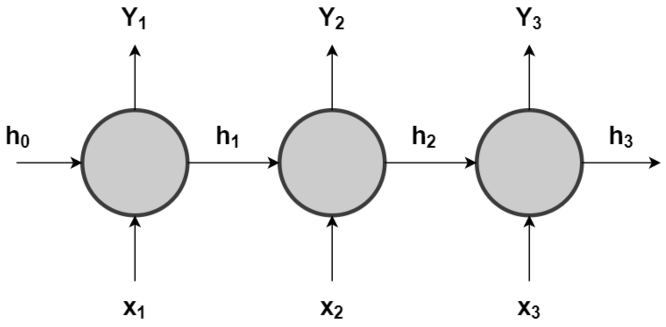


Figure 2.2. *Recurrent neural network*

First, it takes X_1 as input and produces Y_1 as output along with h_1 which together with X_2 is used as input for the next step. Similarly, h_2 along with X_3 is used as input for further step and it keeps on going like that to remember the contextual information. For calculating the output of the current state the formula is

$$Y_t = f(h_{t-1}, X_t), \tag{2.2}$$

where Y_t, X_t and h_{t-1} is the current output, current input and previous hidden vector. As we see current input could be dependent and get some useful information from past context, in some cases future context can also provide useful data. To utilize future contextual information bidirectional RNN (BRNN) is the key to success. Each layer is divided into two separate layers in BRNN, one reads the forward sequence and the other takes care of the backward sequence. This way the network will have both past and future context of the input sequence to predict the output.

Gradient vanishing and exploding is one of the common problems in RNN which are being handled in long short-term memory (LSTM) networks [8]. A simple neuron in RNN is replaced by units known as LSTM memory blocks (Figure 2.3). LSTM uses backpropagation to train the model and consists of three gates (input gate, output gate, forget gate). Hidden activation in RNN is replaced by the following equations [9]

$$\begin{aligned}
i_t &= \sigma(W^{xi}x_t + W^{hi}h_{t-1} + W^{ci}c_{t-1} + b^i) \\
f_t &= \sigma(W^{xf}x_t + W^{hf}h_{t-1} + W^{cf}c_{t-1} + b^f) \\
c_t &= f_t c_{t-1} + i_t \tanh(W^{xc}x_t + W^{hc}h_{t-1} + b^c) \\
o_t &= \sigma(W^{xo}x_t + W^{ho}h_{t-1} + W^{co}c_t + b^o) \\
h_t &= o_t \tanh(c_t)
\end{aligned} \tag{2.3}$$

where i_t, f_t, c_t and o_t represents the input gate, forget gate, memory cell and output gate activation respectively, b^* are the bias terms and W^{**} are the weight matrices. The input gate controls the information that is added to C, the forget gate regulates the erasing of information and the output gate manages what information will be produced at the output of the LSTM memory block. To obtain bidirectional long short-term memory (BLSTM) network we just replace the simple neuron in BRNN with the LSTM memory block and get past and future context information.

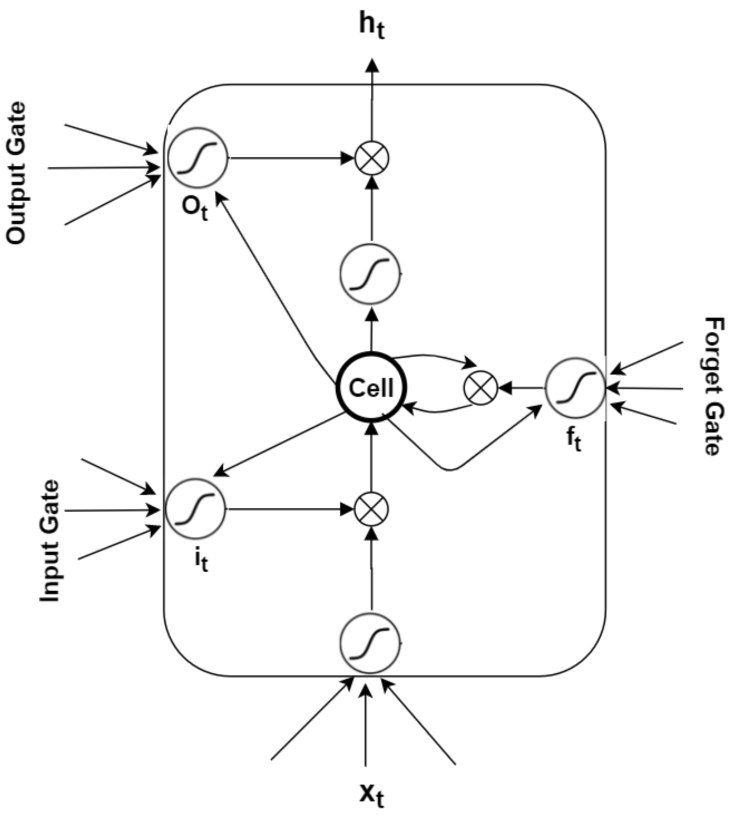


Figure 2.3. *LSTM block*

Gated recurrent unit (GRU) [10] is a simplified version of LSTM with a fewer number of parameters. GRU does not have a cell state vector C and output gate, so they only have an update and reset gate to control the flow of information. GRU gives comparable output accuracy to the LSTM with fewer computations and is considered as favorable in sound event detection. The equations for various gates are defined as

$$\begin{aligned}
z_t &= \sigma(W^z[h_{t-1}, x_t] + b^z) \\
r_t &= \sigma(W^r[h_{t-1}, x_t] + b^r) \\
n_t &= \tanh(W^{in}x_t + b^{in} + r_t(W^{hn}h_{t-1} + b^{hn}) \\
h_t &= (1 - z_t)n_t + z_t h_{t-1}
\end{aligned} \tag{2.4}$$

where z_t, r_t and h_t are respectively the update gate, reset gate and output, W^{**} are the weights matrices and b^{**} are the bias terms.

2.2 Pooling Techniques

As discussed earlier, sound event detection can be divided into two subtasks: determine which events are present in recording (audio tagging) and specify the onset and offset of the event. Multiple instance learning [11] is a common framework for audio tagging tasks where instances are grouped in bags and we make prediction for every bag. A recording in SED is considered as a bag while its frames are interpreted as instances. Neural networks makes a prediction on every frame about the presence of events and then those frame-level predictions are combined with the pooling method to get the prediction for a complete recording. Wang et al. compare five different pooling techniques [12] for sound event detection as shown in the Table 2.2.

Table 2.2. *Five various pooling functions and their gradients*

	Pooling Function	**Gradient**
$Average\ Pooling$	$y = \frac{1}{n}\sum_i y_i$	$\frac{\partial y}{\partial y_i} = \frac{1}{n}$
$Max\ Pooling$	$y = max(y_i)$	$\frac{\partial y}{\partial y_i} = \begin{cases} 1, if y_i = y \\ 0, Otherwise \end{cases}$
$Exp.\ Softmax$	$y = \frac{\sum_i y_i \exp y_i}{\sum_i \exp y_i}$	$\frac{\partial y}{\partial y_i} = (1 - y - y_i).\frac{\exp y_i}{\sum_j \exp y_j}$
$Linear\ Softmax$	$y = \frac{\sum_i y_i^2}{\sum_i y_i}$	$\frac{\partial y}{\partial y_i} = \frac{2y_i - y}{\sum_j y_j}$
$Attention$	$y = \frac{\sum_i y_i w_i}{\sum_i w_i}$	$\frac{\partial y}{\partial y_i} = \frac{w_i}{\sum_j w_j}, \frac{\partial y}{\partial w_i} = \frac{y_i - y}{\sum_j w_j}$

All mentioned pooling techniques are use for calculating recording-level probability output y, where y_i and w_i are the frame-level probabilities and weights respectively. For attention pooing function the weights w_i are learned with a softmax-based fully connected layer. Linear softmax pooling was performing better for audio tagging as well as localization in their experiments, but they also mentioned this could not be an ultimate choice and other polling technique like attention or adaptive pooling can perform better.

2.3 K-Fold Cross-Validation

In the case of limited data, machine learning models are evaluated through cross-validation technique [13]. Cross-validation is a kind of resampling procedure in a way that two sets should not overlap. This procedure has a single parameter K which represents the number of splits from the data. A standard approach to use K-fold cross-validation have the following steps:

- Divide the data into K splits called folds
- Use K-1 splits for training and one remaining split for validation and store the results
- Repeat the process by taking every split as a validation fold
- Summarize the model skill by averaging the results from all folds

Four fold cross-validation is shown in Figure 2.4, where every fold data is used as test data.

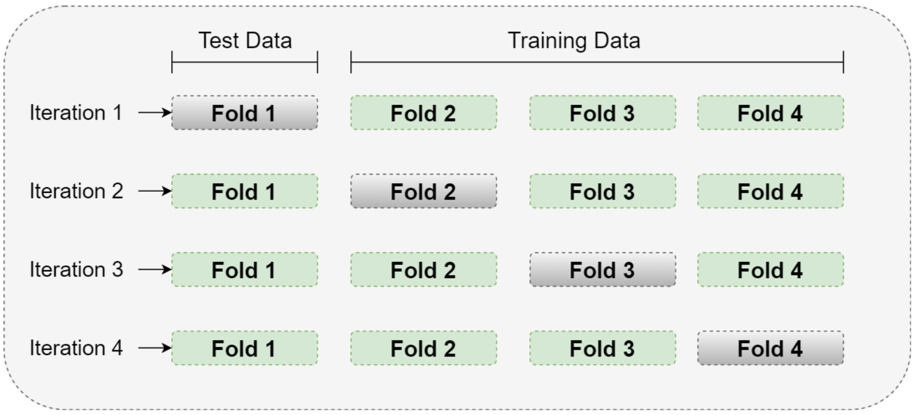


Figure 2.4. *4-fold cross-validation*

2.4 Evaluation Metrics

Classification accuracy is measured by taking the ratio of the number of correct predictions to the total number of predictions. Accuracy is appropriate if the dataset is balanced concerning the number of events of different classes. In the case of an imbalanced number of target class samples accuracy becomes unsuitable measure. Accuracy could be 90 or 99 % with an unskillful model depending on how extreme the dataset is imbalanced. Let's consider an example where our model have to look at cases and find out the fraud and non-fraud cases. If we only look at the accuracy of that model, that is 99.9%, we

		Predicted	
		Non-Fraud	Fraud
Actual	Non-Fraud	998	0
	Fraud	1	1

can say the model is performing very well. On the other hand, if we consider actual fraud cases which model misclassified, then we can think of the cost that this misclassification posed. Precision, recall, error rate and AUC metrics could be used to evaluate the performance of your model and to overcome the problem mentioned above. For calculation of precision, recall and F1 score the number of true positive, false positive and false negative are calculated as:

- If an event is detected in the data and it is also present in ground truth of that data then event is regarded as true positive
- If an event is detected in the data and it is not present in ground truth of that data then event is regarded as false positive
- If an event is not detected in the data but it is present in ground truth of that data then event is regarded as false negative

Precision calculates how much precise our model is and out of total predicted positive how many of them are actually positive. In cases where costs of having false-positive is high precision is a good measure.

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \tag{2.5}$$

Recall, on the other hand, talks about out of total actual positive how many of them are captured as positive from our model. The recall is a good measure in cases where the cost of false negative is high as it was in the above virus or fraud example.

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} \tag{2.6}$$

Precision and recall confront two different problems and when we want to have balance between these two quantities we use **F1 score**. F1 score is a harmonic mean between precision and recall and is considered a good measure of evaluation in case of an imbalanced dataset.

$$F1 = 2 * \frac{precision * Recall}{Precision + Recall} \tag{2.7}$$

One other metric that was used in detection and classification of acoustic scenes and events (DCASE) 2013 for evaluation was the error rate, and it can be used to increase the confidence on your system. It is calculated by considering the number of insertion, deletion and substitution.

$$Error\ Rate = \frac{Substitution + Deletion + Insertion}{ReferenceEvents} \tag{2.8}$$

These metrics can be computed segment-based and event-based as described by Mesaros et al. in [14]. In a segment based metric, the active or inactive status of an event is predicted on a fixed short interval and then compared with ground truth labels of the same interval. On the other hand, event-based metrics are used to compare the predicted output and ground truths event by event.

The receiver operating characteristic curve (ROC) and area under the curve (AUC) can also be used for classification accuracy. The true positive rate and false positive rate are plotted at a different threshold to make ROC, while AUC is used to summarize it into a single number. In Figure 2.5 we can take a look at TP vs FP rate at different classification thresholds and the curve is formed in the two-dimensional area. If we calculate the area under that curve then we will get one value which is AUC. AUC is a good measure to estimate the quality of your model irrespective of the value of the classification threshold.

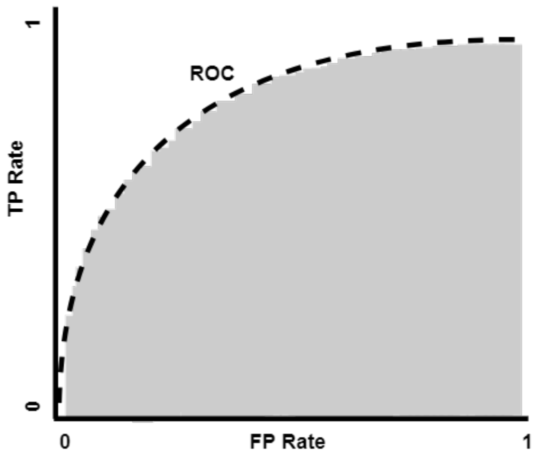


Figure 2.5. *Area under ROC curve*

Jaccard index could also be one other evaluation metric that could be used to measure similarity and dissimilarity between sets. This is easy to interpret as in the case of multiple classes one number could show the performance of the method. It is calculated by taking the ratio of the size of the intersection to the size of union of sample sets as shown in equation 2.9.

$$J(A,B) = \frac{A \cap B}{A \cup B} \tag{2.9}$$

2.5 Sound Event Detection

Real-life sounds have multiple events happening at the same time and that makes the task a little bit complicated. Cakir et al. [15] used multi-label deep neural networks for polyphonic sound event detection. The recordings are divided into frames and features are extracted from each frame to detect sound events in that specific frame. They used maxout activation function in hidden layers and sigmoid activation for the output layer. The output is binarized to 0 or 1 with the help of specifying a certain threshold. Log mel-band energies are used as a feature in experiments and F1 scores were used as an evaluation metrics.

Parascandolo et al. [9] proposed bi-directional LSTM for SED in real-life recordings. As discussed in DNN part above, they support the benefits of using RNN over FNN to avoid smoothing steps at the end because RNN has the ability to remember previous states. They use log mel-band as features and normalize them by subtracting the mean. Acoustic features are mapped to class activity by using multilabel bi-directional LSTM along with several hidden layers. To avoid overfitting issue they use an interesting approach of data augmentation. Time stretching, sub-frame time-shifting and block mixing transformations are applied to the features in freq-domain. For evaluation, they use frame-wise and 1 s block average F1-score and get considerable improvements in results as compared to previous FNN base results.

Cakir et al. [16] proposed the convolutional recurrent neural networks approach for polyphonic sound event detection. They use CNN for the extraction of higher-level features from audio and then RNN is used to keep contextual information. As discussed earlier, feedforward neural networks for SED lack temporal context as well as frequency and time invariance. These two problems are tackled in their paper with CRNN where CNN's addresses the frequency and time invariance problem by learning various filters and RNN deal with temporal context issue by integrating information of consecutive frames.

They trained the model using binary cross-entropy as a loss function and adam [17] as gradient descent optimizer. Segment-based F1 score and segment-based error rate were taken as evaluation metrices and experiments were done on four different datasets. For the experimental setup, the Gaussian mixture model and feedforward neural network were used as baseline models while standalone CNN and RNN were considered as advanced models for comparison. CRNN performed better on all four datasets and considered as a valid approach for polyphonic sound event detection, however, the performance of CRNN depends on the size of annotated data. As future work, they proposed a semi-supervised training method and transfer learning as a potential candidate to overcome annotated data limitation.

Xu et al. won the 1st position in large-scale weakly supervised SED in detection and classification of acoustic scenes and events (DCASE) 2017 challenge. They proposed [18] a CRNN approach with a learnable gated linear unit (GLU) and temporal attention on frames for audio tagging and localization. Instead of ReLU activation GLU is used to control Information flow between convolution layers for audio classification. Time-frequency (T-F) unit is attended when the GLU gate value is close to 1 while it is ignored if the gate value is near to 0. This way unrelated events are ignored and required sound events are attended. GLUs defines as

$$Y = (W * X + b) \odot \sigma(V * X + c), \tag{2.10}$$

where $\odot$ denotes an element-wise product, σ is a sigmoid function and $*$ represents the convolution operator. X in the equation is a T-F representation input, b and c are biases while W and V are convolution filters.

To predict the temporal location of sound events they proposed an attention-based approach. The RNN output is feed to FNN with sigmoid activation function and frame-level classification is performed there. RNN output is also sent to FNN with softmax activation function which produces weights for events and considers only salient frames for different classes. Element-wise multiplication is done between the outputs of both parallel layers and then the average is taken across the time axis to get the final output. This approach is a unified approach for audio tagging as well as weakly supervised SED.

For the training machine learning algorithm, we need a huge amount of data and that is the limitation in SED classification. Audioset [19] is the main dataset available at this time with weakly labeled data but certain events are inherently rare and decrease the usefulness of DNN. Kumar et al. [20] proposed knowledge transfer using a convolution neural network for sound event detection and scene classification. They train a CNN model on the audioset and use that model in domain adaptation as well as in the task adaptation scenario. For training with Audioset, they used 13 convolution layer model to produce frame-wise output which is then converted to recording-level output by global max pooling. This recording-level output is then used to calculate the loss and optimize the model. To transfer the learning they use 11 convolution layers from the 1st method as it is and change the last two layer in three different manners. To adapt the different tasks they use different target datasets (ESC-50) and only update the last two layers during training. They achieve 83.5% accuracy on the ESC-50 dataset which was higher than human accuracy on this dataset. Their methods of transfer knowledge are also helpful for understanding the relation between sound events and scenes.

Wang et al. [21] proposed deep recurrent neural networks for audio-based multimedia event detection. Audio in multimedia often contains useful information about the environment and sometimes that information is not even present in visual content. They classify audio frames into "noisemes" (semantic units) and performed frame-level noiseme classification and clip-level event detection. For feature extraction, they took MFCCs as a low-level feature and then use a sliding window to get a variety of statistics over low-level features. One interesting thing they used in their work is the Adaptive learning rate, which changes its value on basis of validation error rate. They concluded recurrent neural networks (RNNs) could utilize temporal information for frame level and clip-level classification in multimedia event detection.

2.6 Robustness

Reverberation in acoustics needs to be considered seriously while doing SED. Reverberation is created when sounds reflect from different surface. Normally the sound decays because of absorption due to various objects in space (furniture, air and people) and that is the reason reverberation is high in empty places. It becomes noticeable when actual sound is stopped but the reflection continues due to slow reduction of amplitude with time. These reverberating sounds could become challenging for detectors, and sometime late reverberations can be categorize as sound event. Dereverberation could also be used to remove effects of the reflections before providing the audio to the detector or it can also be handled at the detector side.

Noise and other environmental sounds are also needed to take care of while doing sound event detection. If there is some kind of noise in the data then you have to make sure the data is properly divided into train and test, so that model will also learn the effect of noise. Sometimes environmental sounds are very similar to some of the target sounds and could be very confusing. In case of such sounds, you should include those sounds in training data to teach the model about ignoring those sounds.

Impulse response measurement can also help in making the sound event detection system more robust and efficient. Different rooms and places have different effects on audio recordings and to make a system for a specific environment you need to know the effect of that place. Impulse response measurement is a good way to have idea of different places and then use that information for designing the system. For example in the case of training with synthesized data you need to make your data to be similar to real-life recordings. To achieve similarity with real-life recording you can measure the impulse responses of actual rooms and then convolve them with synthesized data as

$$Output\ Recording = SR * RIR, \tag{2.11}$$

where $*$, SR and RIR represents the convolution, synthesized recording and room impulse response respectively. For measuring impulse responses we play an excitation signal from speaker and record that from microphone as shown in Figure 2.6. The original and recorded signal are used to recover the impulse response of a room. Maximum length sequence (MLS), inverse repeated sequence (IRS), time-stretched pulses and sinesweep [22] could be used as an excitation signal for impulse response measurement.

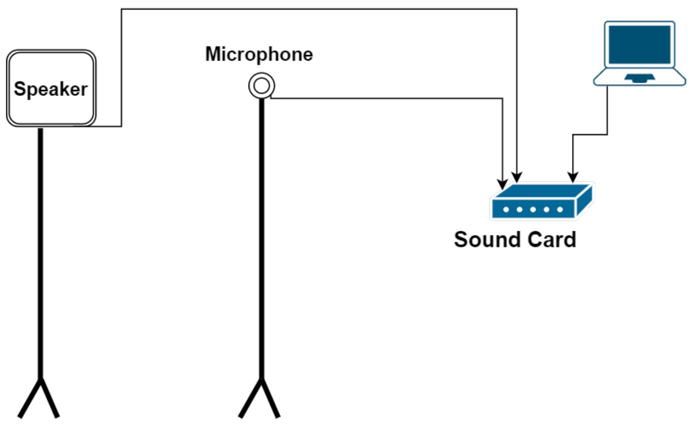


Figure 2.6. *Impulse response measurement setup*

3 METHODS

3.1 Dataset Creation

3.1.1 Data Collection

Freesound Data
Freesound provides a huge amount of audio data with a variety of samples that could be used to create a database for research purposes. For collecting samples of target classes (door, water tap, yelling, human fall) I started searching on this website. To gather individual samples, I searched with different key words for every class e.g yelling is searched with scream, call for help, yell, cry and pain. After searching these samples, I go through each audio samples manually and listen to them before selecting that sample. Door and water tap samples are very easy to find and we gather a variety of samples in these classes. For door sounds we consider open, close and open-close sounds along with variation in types of the door (old, new, main door, inner door). The yelling class has very broad meaning, and we define this class as abnormal voices or the sounds that would not be present in a typical scenario. Human fall samples are a bit rare and difficult to find online but we managed to get some of them from freesound. We find a paper [6] where they are detecting human fall and have recorded the data, so we contact them and get their data for our experiments. This way we gather around 100 samples for each target class.

One-shot human fall detection [6] dataset was recorded in three different rooms (classroom, recording room and auditorium). Real human fall was performed in recording studio and auditorium, while in the classroom they use a manikin doll instead of humans. When we listen to the sounds recorded in the recording studio, they are very low, and we cannot hear them without amplification. When we amplify the sounds it also increases the noise in the recording up to unacceptable levels, so we cannot use these recording sounds. On the other hand, recordings in the auditorium were a little bit louder so I listened and chose some of them for our use. I chose those sounds which have some frequencies above 1000 Hz in the spectrogram. Initially, we thought we would only choose sounds from real human fall, but later on, we decided to consider manikin doll fall for our dataset.

VoxCeleb1

Speech is considered as one of the non-target events for our dataset and we want some kind of real speech instead of reading a book. We choose the voxceleb1 [23] dataset for speech, it contains 1251 celebrities voices in total. It seems more natural speech as compared to reading books because celebrities are talking and telling about themselves. We have metadata available for this dataset in a CSV file. The following information is available in the metadata file.

VoxCeleb1 ID	VGGFace1 ID	Gender	Nationality	Set

LibriSpeech

LibriSpeech [24] is a corpus of approximately 1000 hours of 16 kHz read English speech. Reading books seems more like television news, so we can use that for TV. This dataset also has metadata available. The following information is available in the metadata file.

ID	SEX	SUBSET	MINUTES	NAME

MUSAN

This corpus [25] provides data for music/speech discrimination, speech/non-speech detection, and voice activity detection. The corpus is divided into music, speech, and noise portions. In total there are approximately 109 hours of audio. All files in this corpus fall under a creative commons license or are in the USA public domain.

Music

Music is annotated for the presence or absence of vocals and by genre(s). It contains western art music (e.g., Baroque, Classical, and Romantic) and popular western music genres (e.g., Country, Hip-Hop, Jazz, etc)

Fma

This folder contains files at 16 kHz sampling rate with the mono channel. The length of the files is between the 50 seconds and 11 minutes. On average most files vary between 3 and 5 minutes.

Fma-western-art

This folder contains files at a 16 kHz sampling rate with a mono channel. The length of the files is between 50 seconds and 7 minutes. On average most files are around 3 minutes long.

Hd-classical

This folder contains files at a 16 kHz sampling rate with a mono channel. The length of the files is between 1 and 16 minutes. On average most of the file's length is between 2 and 6 minutes.

Rmf
This folder contains files at a 16 kHz sampling rate with a mono channel. On average most of the files are 2 and 4 minutes long.

US-Gov
Speech in this folder is from US-Gov people discussion and does not have annotation file. The files length varies between 2 and 10 minutes but most of the files are between 9 and 10 minutes long. Some of the files were up-sampled at 16 kHz sampling rate, so I have removed those files from this folder.

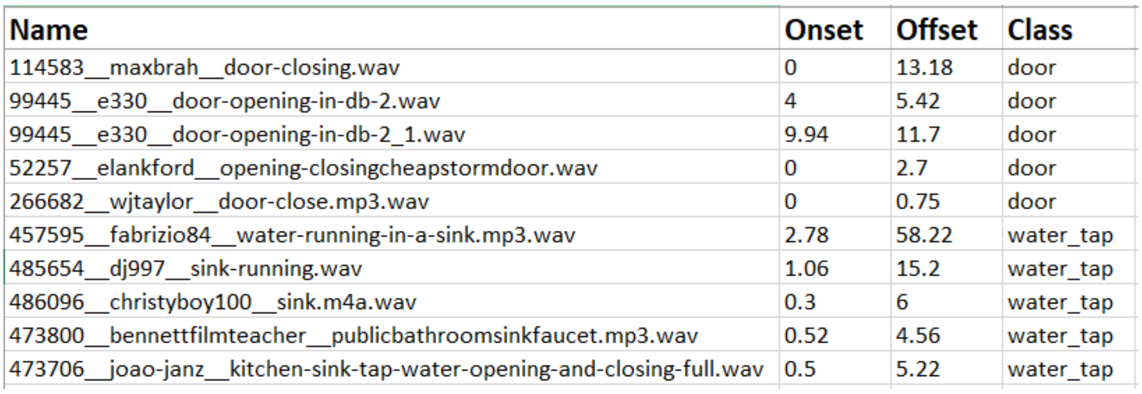
Speech
This data is more like reading a book that could be used as TV. It also contains an annotation file with identification, gender and nationality of the speaker.

ID	GENDER	NATIONALITY

Ambience files are also required to combine them with the above-mentioned sound events to create real-life soundscapes. The criteria for selecting ambience files is to have a single holistic sound that should be coming from distant. Wind, snoring, typing on computer and clock ticking sounds are some examples of ambience sounds that are present in real-life environment. These files are taken from multiple sources such as downloaded from freesound, DCASE 2016 [26] office sounds and some real-life recordings.

3.1.2 Data Preparation

Annotation
Annotation is the first part of data preparation, where we need to point out the starting point (onset) and ending point (offset) of the sound event in a recording file. Annotation normally takes too much time, so I need to find ways to somehow reduce the time. I use a semi-automatic process that requires some empirical tuning of the class-specific thresholds to get good enough rough estimates and then manual screening and fine-tuning of the timing. It is faster than annotating from scratch, but it still takes time. PyaudioAnalysis library is used for this purpose, and silenceRemoval function in audioSegmentation fits a classifier with the most and least energetic frames and then predicts the activity for the rest of the frames. We created an excel sheet by adding the onset, offset and class information w.r.t the name of files as shown in Figure 3.1. This onset and offset information will be extracted from the sheet before adding a target event in soundscape.



Name	Onset	Offset	Class
114583__maxbrah__door-closing.wav	0	13.18	door
99445__e330__door-opening-in-db-2.wav	4	5.42	door
99445__e330__door-opening-in-db-2_1.wav	9.94	11.7	door
52257__elankford__opening-closingcheapstormdoor.wav	0	2.7	door
266682__wjtaylor__door-close.mp3.wav	0	0.75	door
457595__fabrizio84__water-running-in-a-sink.mp3.wav	2.78	58.22	water_tap
485654__dj997__sink-running.wav	1.06	15.2	water_tap
486096__christyboy100__sink.m4a.wav	0.3	6	water_tap
473800__bennettfilmteacher__publicbathroomsinkfaucet.mp3.wav	0.52	4.56	water_tap
473706__joao-janz__kitchen-sink-tap-water-opening-and-closing-full.wav	0.5	5.22	water_tap

Figure 3.1. *Annotation file structure*

Sampling Rate

We downloaded the data from various platforms e.g. freesound.org (target sounds), openslr.org (LibriSpeech) and robots.ox.ac.uk (VoxCeleb1). Sampling rate was different for various files. Freesound data have a sampling rate of 44.1 kHz, 48 kHz and 96 kHz but some of our datasets (non-target sounds) have a sampling rate 16 kHz. We decided to resample all the data at 16 kHz. There is one more problem in the dataset which is about the upsampling of some files. When we look at the sampling rate for some files it shows 16 kHz, but the spectrogram was showing 8 kHz. For those upsampled files we decided to look manually and remove them from the dataset.

Folds Creation

For cross-validation purpose, we discussed to have four folds of synthesized data with non-overlapping samples. I look at every class samples as well as ambience files and randomly divide all of them into four different folds. For creating folds, I also keep in mind the target class samples from the same source should be assigned to the same split. The structure of every fold is similar to the one shown in Figure 3.2. These four splits of samples will be used to create the respective four split of soundscapes.

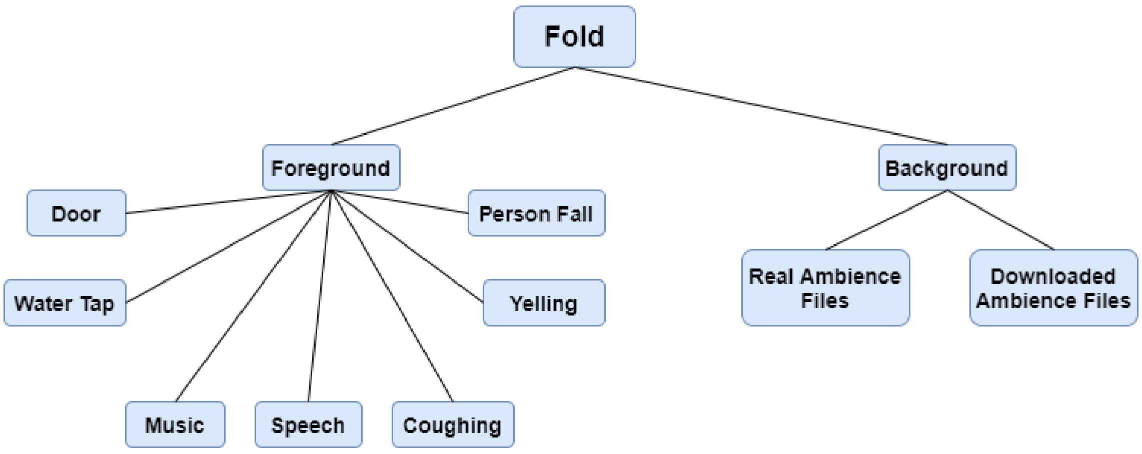


Figure 3.2. *Each fold structure in four folds data*

Real-life Soundscape Synthesis

We have to create mixtures of different sounds to simulate the real-world environment. We have audio samples of target and non-target classes in the form of four splits as mentioned above. For creating mixtures we consider an open-source tool Scaper [7] that is developed by Salamon et al.

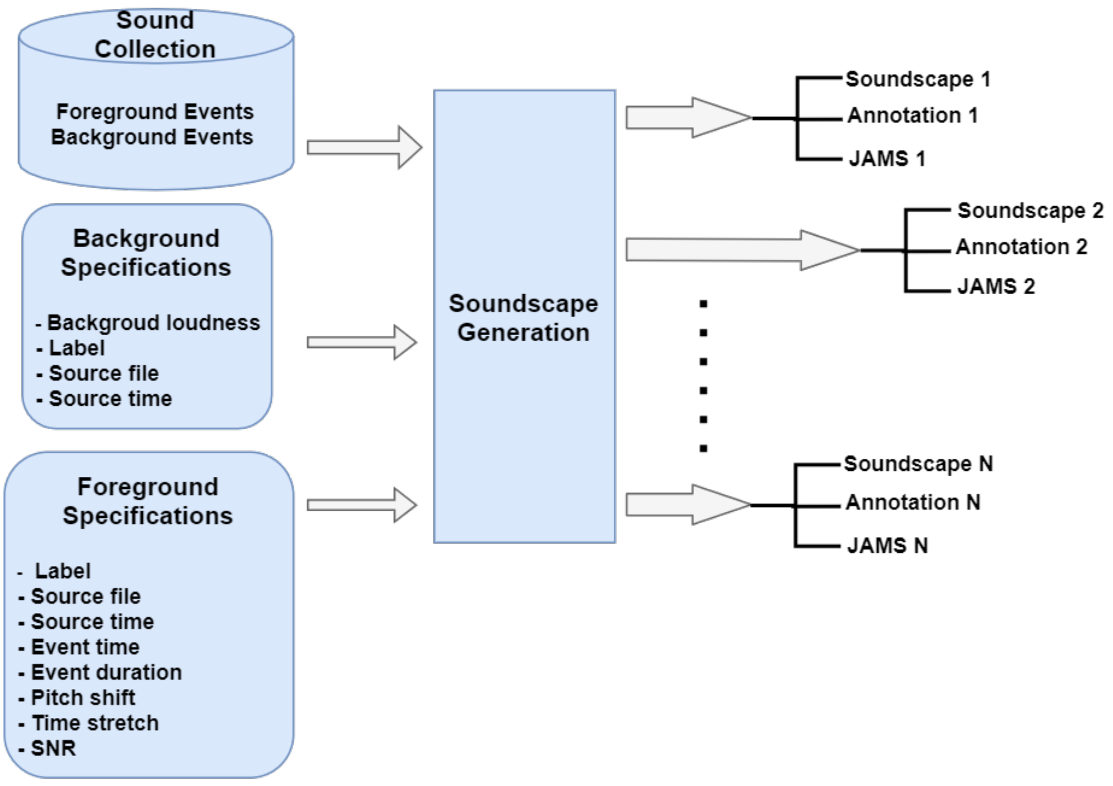


Figure 3.3. *Scaper synthesis pipeline*

Scaper creates mixtures with background and foreground audios as shown in Figure 3.3. There are two different directories with background and foreground names, which contain sub-directories of different classes. You can select the source file randomly from these directories or you can also specify some specific directory or a file. In our case, we put ambience sounds in the background directory while all other classes data in the foreground directory. We randomly select a file from the foreground directory (target-event) and then use annotation.csv file to find the onset and offset. Event time and event duration information in Figure 3.3 is provided with help of onset and offset of that event. Non-target events are added randomly with a variable length that can vary between 0 and 60 seconds.

To create a mixture with different number of sound events, we follow two different approaches:

- In the first approach, we randomly generate a number between 0 and 6 for adding total events in the ambience file. Out of these total events, 65% of events were dedicated to the non-target class and 35% to target ones. There was an issue in that approach, we do not have control over the occurrence of each class, but we were only controlling the target and non-target distribution. In most of the files, it adds music more as compared to speech, but we want to have some kind of balance between different classes, so we moved to the 2nd approach.

- In the second approach, we do not put any restriction on the total number of events, but we set the probability for the occurrence of every event. The probabilities for various events are decided considering the occurrence of those events in the real environment as shown in Table 3.1. This way we can control any event by just changing the probability for that event.

Table 3.1. *SNR ranges with probability of occurrence of various number of events in one file*

Event	**SNR**	**No. of Events**				
		0	1	2	3	4
Door	0->+20	0.5	0.45	0.05	0	0
Water Tap	-10->+20	0.5	0.45	0.05	0	0
Yelling	0->+20	0.5	0.45	0.05	0	0
Falling	0->+30	0.5	0.45	0.05	0	0
Speech	-5->+20	0.5	0.125	0.125	0.125	0.125
Music	-10->+10	0.5	0.45	0.05	0	0
Coughing	-5->+20	0.5	0.45	0.05	0	0

We also have the capability to choose the level of background loudness as well as signal to noise ratio (SNR) while adding foreground. Initially, we set the background loudness to -50 LUFS for all soundscapes, but to make more variation in the data we change this value to be a random number between -40 and -50. For selecting SNR values while adding foreground events we follow two different approaches:

- Initially, we consider generating mixtures by selecting a random number from -10, -5, 0, 5 and 10 for SNR. For some classes like human fall and door -10 and -5 SNR was a bit low, so we decided to use 0, 5 and 10 for such classes.
- After observing some samples, we change the SNR value ranges for every class. We set low SNR values for music because music in nature is a bit loud and when the music is added to soundscape at higher SNR, then it becomes difficult to recognize other events at the same time. SNR for the door, yelling and a person falling events is set a bit higher because these events are quite in nature and for negative SNR these events become impossible to recognize. The reason for having negative SNR for some events is to consider the scenario when some sounds will come from other

rooms. Specific SNR ranges for different classes are shown in Table 3.1

Scaper generates output file at 44.1 kHz sampling rates, but we decided to use 16 kHz as discussed above. Scaper's core.py file was changed to achieve a 16 kHz sampling rate for output files.

Feature Extraction

Openl3 [27] is an open-source tool for deep audio embedding, and it is an improved version of L3-Net. It is trained using self-supervision and exploiting the audio and video correspondence on a subset of AudioSet [19]. The video and audio subnetwork extracts the respective feature and then correspondence is created between them by the fusion layer.

We were using PyTorch [28] so we extract the openl3 model for PyTorch and use that pre-trained model to get embeddings. We load the audio at 48 kHz and convert that into log-mel spectrogram with the help of librosa [29]. Log-mel spectrogram features are then reshaped to meet the dimension requirement of openl3. This model returns two objects, one is T-by-D dimension embeddings, where T is the number of frames and D represents the number of features in each frame. Openl3 supports two different numbers of features in each frame 6144 and 512. In our synthetic dataset each audio is one minute long and we get 60 frames and each frame have 512 features.

Kumar2018: Kumar et al. [20] proposed knowledge transfer methods and extracting meaningful features from audio. They showed the learned features with the proposed CNN model reach human-level accuracy on the ESC-50 dataset. They also use extracted features for acoustic scene classification experiments and get good results. It is trained using the AudioSet [19] dataset to produce embeddings, and works on 44.1 kHz sampling frequency. We also consider this embeddings model for our task to see the classification accuracy. The initial procedure for embeddings extraction is the same as openl3, like loading the audio and get a log-mel spectrogram representation of audio from librosa. It gives 28-by-1024 dimension embeddings for one minute audio, where 28 are the number of frames and 1024 are the features in each frame.

I also receive a different version of the kumar2018 model that was trained with 16 kHz data. We consider this model because our synthetic dataset is 16 kHz. We refer to these embeddings in our work as kumar2020 and it produces 29-by-1024 dimension embeddings for one minute audio.

Zhao2020: This is also an embeddings extraction model that is based on gated CNN and trained using AudioSet [19]. The embeddings are extracted per frame of log-mel spectrogram and change point detection is performed on embeddings. It works on 16 kHz sampling frequency and produces embeddings. For one minute audio it produces 239-by-128 dimension embeddings, which means every frame is 0.25 s resolution.

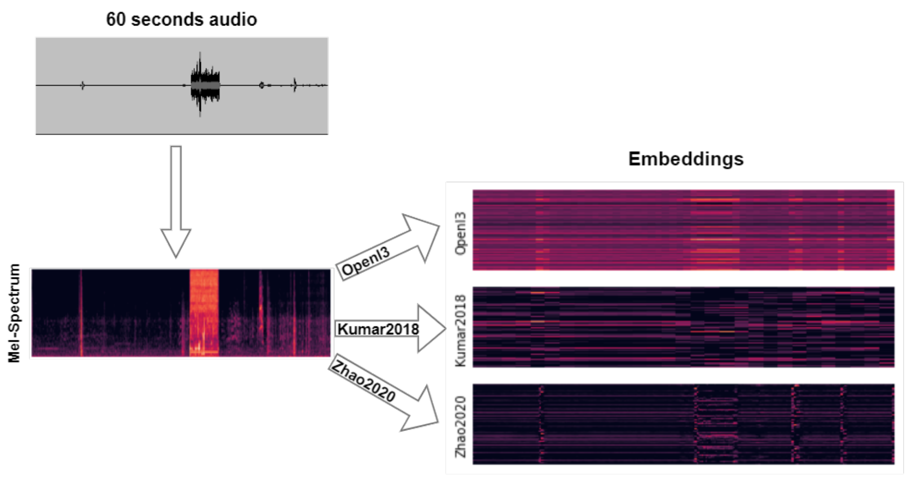


Figure 3.4. *Openl3, kumar2018 and zhao2020 embeddings illustration*

Label Extraction

As discussed above, the Scaper generates soundscape as well as an annotation file that contains the event name along with the onset and offset of that event. We use that file to create ground truth labels for our data. Based on our embeddings, the number of frames for one minute audio vary, so we have to tell our label extraction system the number of frames in one audio file. The label extractor divides the 60 s by the number of frames to get the time resolution of each frame. Labels are extracted for each frame by considering the respective annotation file information. Labels are created for all seven classes (door, water tap, person fall, yelling, speech, music and coughing). For each frame a 1-by-7 vector is generated and if the specific event is present in that frame we put 1 for that event otherwise 0. For openl3 we generate 60-by-7, for kumar2018 28-by-7 and 239-by-7 dimension label matrix for zhao2020.

Embedding Database Creation

To create an embeddings database, we extract the embeddings and labels from the audio data, as mentioned in previous sections, and save the data into the HDF5 file along with the information of various folds. HDF5 file is chosen to create a database because it has a key-value pair structure to save the data. The key-value structure makes the system quite fast by making access to a certain value possible with the help of a key. The purpose of database creation is to avoid extracting the embeddings, a time consuming task, whenever you want to train a model. Once we get the database file then we can use that file for all of our experiments. For saving the data we created the HDF5 file structure as shown in Figure 3.5

In the main file, we have three subfolders data log, feature and label. Data log folder contains four datasets with the key as the name of the respective fold and each value of that key keeps the record of the audio file names belongs to that specific fold. The feature folder contains a separate dataset for each audio file where the key of that dataset is the

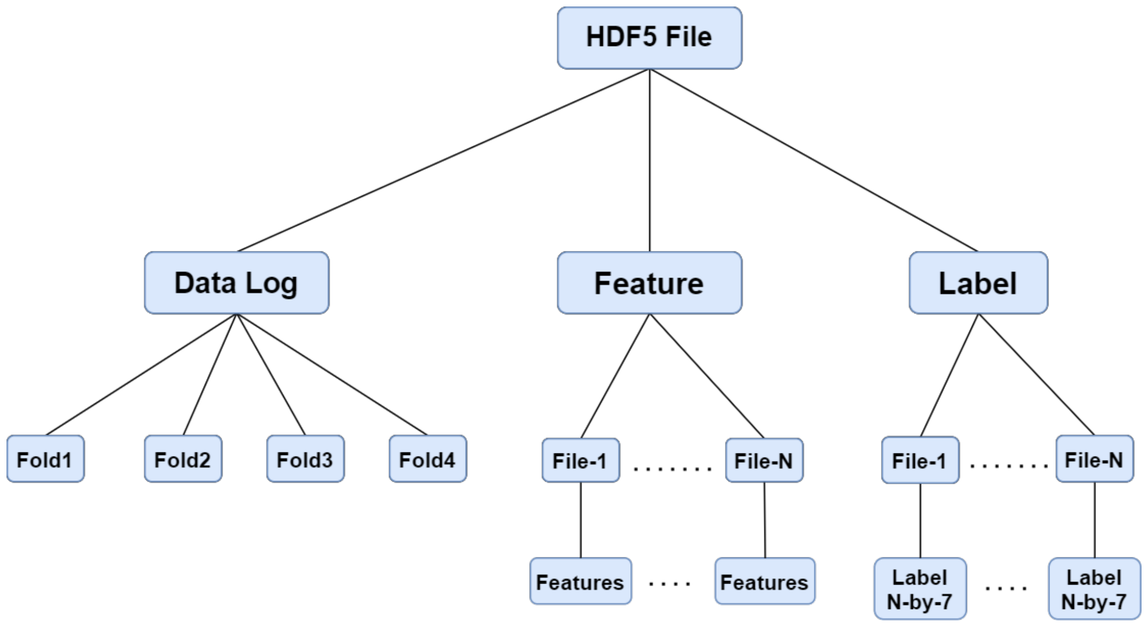


Figure 3.5. *Embeddings database structure*

name of the file and the value is the embeddings of that file. The label folder also stores the distinct dataset for each audio file, and the file name is used as the key of that dataset while the extracted labels are saved as the value of that dataset.

3.2 DNN SED Techniques

3.2.1 Frame-level Prediction

As discussed earlier, all three embeddings extraction models (openl3, kumar2018 and zhao2020) produced a T-by-D dimension embeddings, where T is the number of frames and D is the number of features in each embeddings frame. We consider doing some experiments with the frame-level output of all embeddings to see the difference in DNN and RNN classification models. We cannot compare the frame-level output of embeddings with each other because the time resolution of a single frame is different in all embeddings. We planned to compare a feedforward neural network model and RNN model within each type of embeddings. From the family of recurrent neural networks, we consider GRU for our experiments due to less number of parameters. Now onward, when we refer to RNN that will mean GRU.

Single Layer Feedforward Neural Network

Deep neural network shows promising results in acoustic event recognition [30], so we also consider a single layer neural network model for predicting the output of every embeddings frame. The input of the model is the number of features in one embeddings frame while the output is the number of total target events. The features in one embeddings frame are different for different embeddings models e.g openl3 has 512, kumar2018 has 1024 and zhao2020 has 128. We created a single layer FFNN model as shown in Figure 3.6 for frame-level predictions.

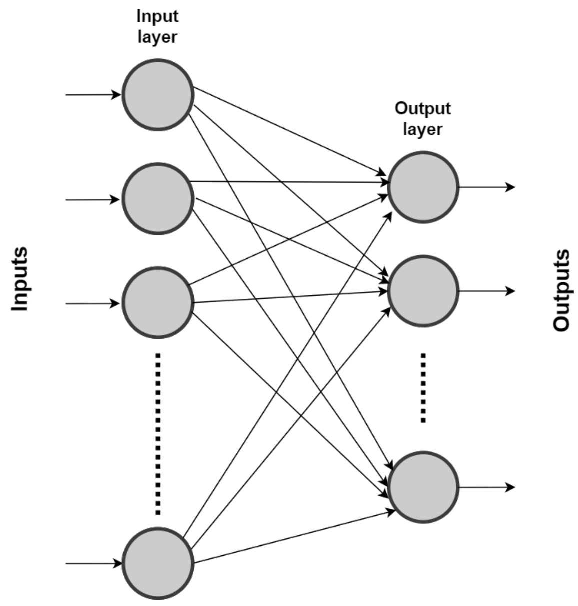


Figure 3.6. *Single layer FFNN model*

Multi-Layer Feedforward Neural Network

After obtaining results from a single layer FFNN model, we moved to a multi-layer FFNN as shown in Figure 3.7. For example, in the case of openl3 embeddings, the input layer will go from 512 to 128 dimension hidden layer, 1st hidden layer to 2nd hidden layer the dimension goes from 128 to 32 and then at the end we go to 7 neuron output layer. One dimensional batch normalization and ReLU are applied after 1st and 2nd hidden layer and sigmoid non-linearity is used at the output layer.

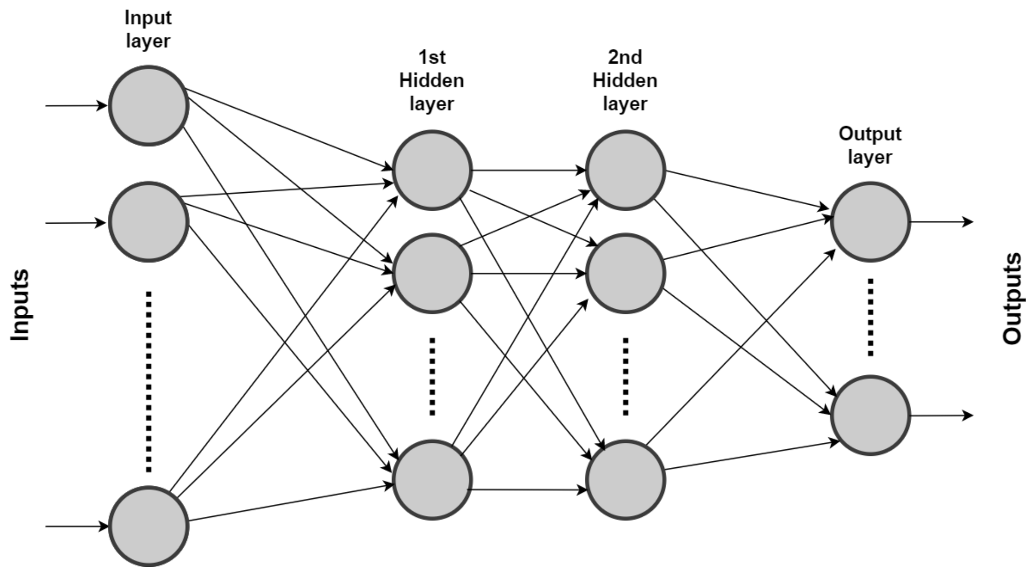


Figure 3.7. *Multi-layer FFNN model*

Uni-Directional RNN

The embeddings frames are sent to the RNN layer to get the benefit of contextual information. X_1 and X_2 are the first and 2nd embeddings frame producing the frame-level output Y_1 and Y_2, while X_t is the last input frame and give Y_t as an output. The output of each frame (Y_1, Y_2 and Y_t) of the RNN layer is then passed through a sigmoid-based output layer to get the frame-level output.

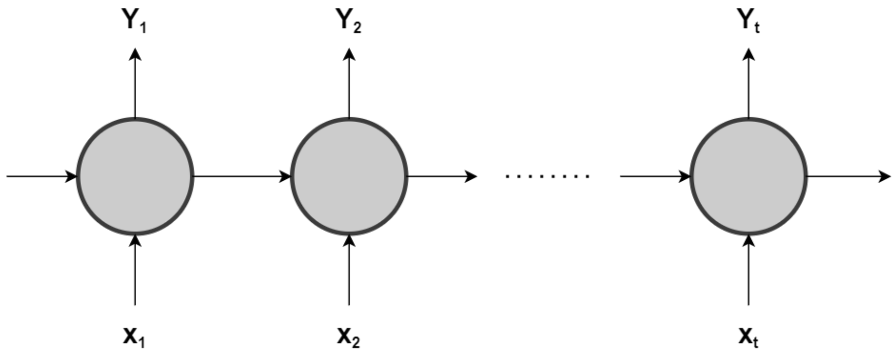


Figure 3.8. *Uni-directional RNN model*

3.2.2 Recording-level Prediction

Considering real-life scenarios of using SED in health care monitoring, where prediction will be done for the resolution of one minute or longer, we think to do experiments for the prediction of one minute. In frame-level predictions, a 1-by-7 output vector is generated for every frame, while in recording-level predictions a 1-by-7 vector is generated for a complete recording. Our synthetic data produce one minute long soundscapes, and we try different models for producing output labels for every recording. Recording-level predictions also make it easier to compare various embeddings and their performance for SED.

Single Layer Feedforward Neural Network

We start recording-level sound event detection by taking a single layer FFNN model that has an input layer and output layer. All the embeddings frames from one minute audio are concatenated to make a long big vector. This concatenated vector of one minute recording is used as input to our model. For one minute audio, various embeddings have a different number of features in each frame, so when concatenated the size of the input layer varies based on the type of embeddings. As we have seven classes in our synthetic dataset, so the output layer always has seven neurons. Sigmoid non-linearity is used at the output layer to get the probabilities for seven classes.

Uni-Directional RNN

As discussed above, when we want to use contextual information then RNN is the best choice. I took a uni-directional RNN model for the experiment, where a single layer of

RNN is used. The number of features and the number of hidden units are kept the same in this layer. As shown in Figure 3.8 embeddings frames (X_1, X_2 until X_t) from audio are passed to RNN and then instead of taking the output of all frames only the last frame output Y_t is taken for further process. The last frame output Y_t also has the information that is passed from the 1st frame due to the capability of RNN. The last frame output is then given to the sigmoid-based output layer to generate the probabilities of output classes present in the full recording.

Bidirectional RNN

We took bi-directional RNN [31] for the next experiment, where forward and backward time steps are used for predicting the current input. The last embeddings frame output from the forward layer is concatenated with 1st frame output of backward layers as shown in Figure 3.9. This concatenated vector is then given to the sigmoid-based output layer for generating predictions.

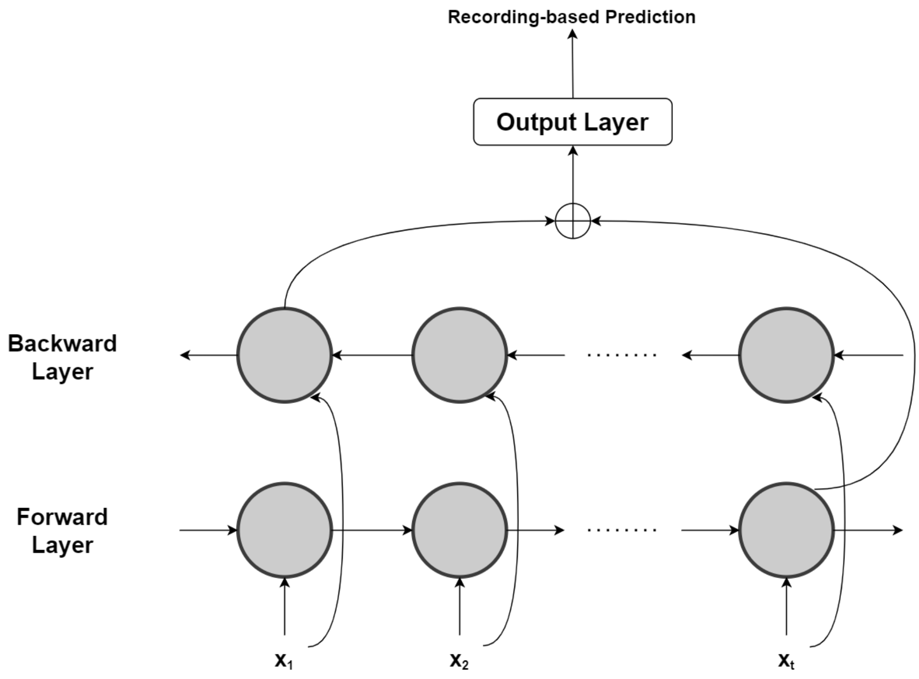


Figure 3.9. *Bi-directional RNN for recording-based prediction*

Residual Connection with RNN

Inspired by the idea of using the residual connection in [32] [33], which has shown good performance results, we also think to try it in our work. We created two layers of RNN as shown in Figure 3.10, where embeddings frames X_* are given to the first RNN layer and processed. Each frame output of the 1st RNN layer is then added with the respective input frame and then given to the 2nd RNN layer as input. Each frame's output Y_* is the addition of the 2nd RNN layer's output with the respective input frame. As we were doing recording-based prediction, so we took the last frame output Y_t and passed through a sigmoid-based output layer to get probabilities of classes.

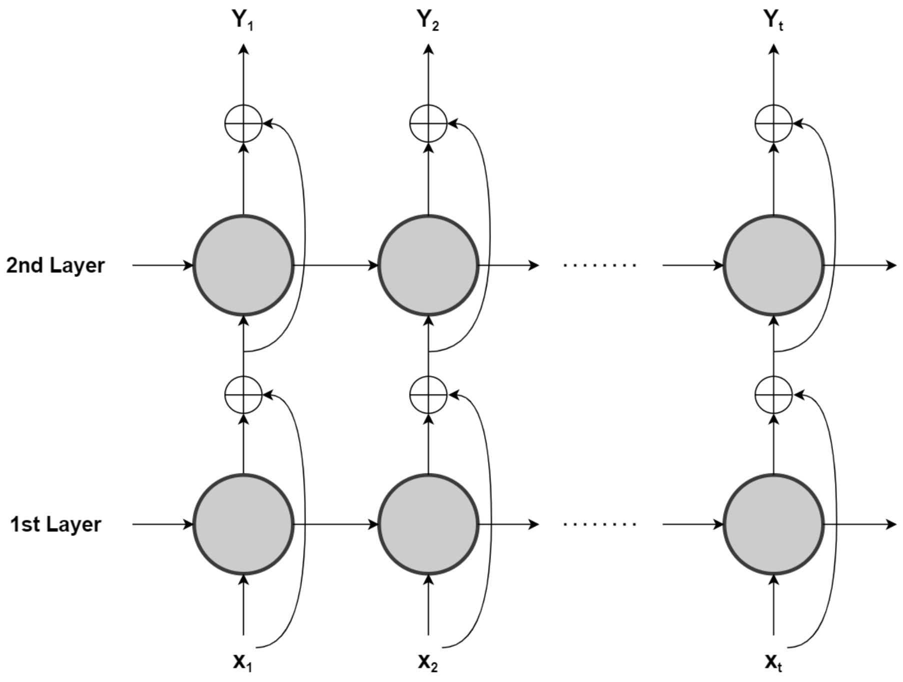


Figure 3.10. *RNN with residual connection*

Attention

The attention model [34] [35] also gains popularity for multiple instance learning problems [11]. They used a trainable weights measure for each class and then these weights along with the classification output probabilities of frames are used to produce the final output. As shown in Figure 3.11, p_x is the frame-wise classification probability for every frame, w_x is the frame-wise weights measure and x belongs to the specific frame of X. The final prediction will be the expectation of p_x w.r.t the weight of w_x.

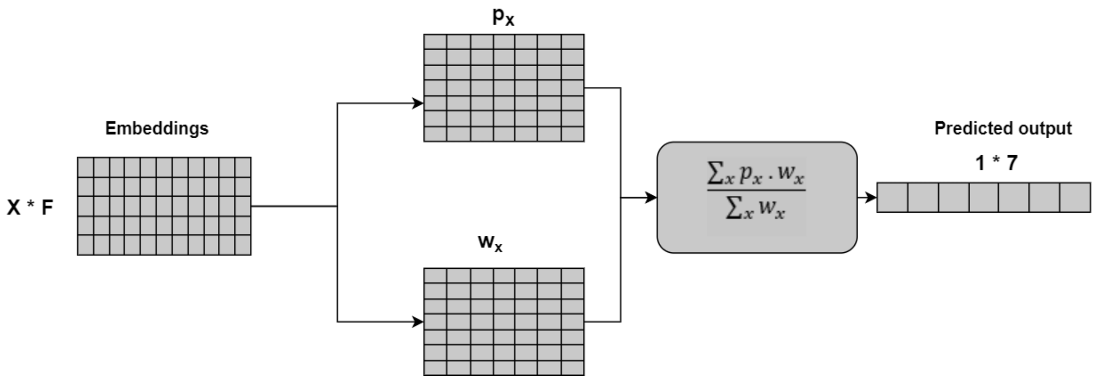


Figure 3.11. *Attention model*

RNN with Pooling

We also use RNN with various pooling techniques to find out the performance of this model. For this model, the input embeddings are passed to the RNN layer, then instead of taking only the last frame output, we take the output of all the frames. Frame-level outputs are then combined with help of pooling to form recording-level predictions. As discussed in [12] and also shown in Table 2.2 five different pooling functions, we consider trying all of them. We use all five pooling functions with RNN as explained above and generate results to select the best performing one for our case.

Train with Strong Label
All of the above methods are generating recording-level prediction labels, and those weak labels are used during training. Loss is calculated on recording-level predicted and ground truth weak labels, and weights are updated on the base of that loss. We try out training with strong labels because our synthetic data have frame-level ground truth labels. In this method, the model generates the probabilities for every frame and then the loss is computed on the base of those strong labels. For evaluation, recording level weak labels are generated on test data by applying pooling. The reason to use weak labels during validation is to compare the results with previously done experiments.

Train with 10 Seconds Resolution Weak Labels
We checked both extreme cases, training with one minute resolution weak labels as well as training with strong labels. We consider doing training with the resolution something in between these two values. We generate 10 s resolution weak labels for ground truth and also create a RNN model to return 10 s resolution weak labels. These 10 s resolution weak labels are generated by applying max-pooling over the results of multiple frames. Training is done with these 10 s weak labels, but the evaluation is done on a one minute resolution to make the results comparable with previous results.

Train with Four Target Classes
As explained earlier, in our synthetic dataset we have 7 classes, 4 targets and 3 non-target. All methods explained above are trained by considering 7 classes and making a prediction for non-target classes as well. Considering healthcare monitoring in a real-life, detecting four sound events (door, water tap, yelling and human fall) are important, so we have a test by reducing the number of predicted classes to 4 (target classes). The dataset remains the same but we dropped the labels of 3 non-target classes and only take target classes labels for training and evaluation. In this method, all non-target classes are considered as background noise and the model will focus only on target classes. The purpose of this experiment is to find out how much performance improvement we get for reducing the number of classes, and is it really worth it to lose the information related to non-target classes.

3.3 Data Loader and Training

We have our data in the form of four non-overlapping folds for cross-validation purpose. We decided to use every fold for testing one by one and remaining for the training of our model. Mini-batch approach is adopted to train the model where batch size was decided by the user. We created a data loader class that loads the folds information from the data log folder and creates a list of train and test files. The data loader class also contains a get-next-batch method that loads the embeddings and labels of a specific mini-batch and sends it for training. The same method also keeps track of the number of batches and when the last mini-batch is sent for training, then it shuffles the training data for the next epoch.

As our problem is a classification problem, so binary cross-entropy (BCE) loss is being used which is optimized for this kind of task. We use Adam [17] optimizer as gradient descent optimizer during the training of our model. As this problem is a multi-label and multi-class problem where multiple events can happen at the same time, so sigmoid is used at the output layer to get the probabilities of every event. The threshold is applied at the output probabilities to get the final predicted labels. To finalize the number of epochs during training on our synthetic data, we run multiple tests by setting the number of epochs 100, 200, 300 and 400. On our synthetic dataset 200 epochs is the optimal number because the best model is always achieved before that.

To train the model 'Training' class is created that contains multiple methods for various tasks. Train method in the training class is used to run four times considering every fold as validation data. Mini-batch is taken from training data and given to the model to get the predicted labels. Loss is calculated with predicted and ground truth labels, and propagated backward after every batch. At the time of last mini-batch evaluation is done with test data and results are calculated. The result of current epoch is compared with the best result of previous epochs and if the current epoch result is better, then the best result is updated. The model is saved whenever the best results are found. At the end of the training, we get four best trained models one for every fold.

For evaluation, we use jaccard index, precision, recall and F1 score. Jaccard index is calculated jointly for all classes and one single number shows the overall credibility of the model. Precision, recall and F1 score are calculated for every class and give the detailed performance overview of individual classes.

The training class also contains load-best-model and test methods. We can load the already saved model with help of this load-best-model method and utilize the test method to get the performance of every fold. The test data method also writes the results in a CSV file for the record.

4 RESULTS AND DISCUSSION

4.1 Frame-level Prediction Results

Frame-level predictions are done to see the difference between deep neural networks and recurrent neural networks. We cannot compare the frame-level results of one embeddings model with other embeddings model because the time resolution of each embeddings frame is different in all embeddings models. We generate the following results for each type of embeddings.

Openl3
Openl3 embeddings model generates a T-by-D dimension embeddings, where T is the number of frames and D is the number of features in each frame. It produces 60 embeddings frames for one minute audio, so every embeddings frame belongs to 1 s time resolution. We use each frame (1 s segment) results for training and evaluation. We tried three different models as shown in Table 4.1 and see the trend as expected. Jaccard score increase when we go from the single layer FFNN model to the multi-layer FFNN model and finally to the RNN model. If we look at the individual F1 scores of classes then we can see RNN is performing better for our task as compared to simple neural network models. The F1 scores for the door and human fall class was not that good with openl3, but some of the classes like music and water tap was performing well.

Table 4.1. *F1 and jaccard scores for openl3's single embeddings frame*

Model	Jaccard	F1 Score						
		Water tap	Door	Yelling	Human Fall	Music	Speech	Coughing
Single layer FFNN	0.406	**0.790**	0.014	0.236	0.059	0.690	0.554	0.033
Multi-Layer FFNN	0.435	0.738	**0.084**	0.325	0.021	0.702	0.623	0.075
RNN	**0.531**	0.787	0.080	**0.503**	**0.221**	**0.838**	**0.694**	**0.143**

Kumar2018
Three experiments performed again with kumar2018 embeddings and the conclusion is the same. In terms of jaccard and individual F1 score, RNN performed better as compared to other models as shown in Table 4.2. Jaccard score increases 5.3% by moving from single layer FFNN to RNN model. Kumar2018's each embeddings frame time resolution is 2.14 s as it is generating 28 embeddings frames for one minute audio.

Table 4.2. *F1 and jaccard scores for kumar2018's single embeddings frame*

Model	**Jaccard**	**F1 Score**						
		Water tap	**Door**	**Yelling**	**Human Fall**	**Music**	**Speech**	**Coughing**
Single layer FFNN	0.588	0.755	0.429	0.600	0.366	0.840	0.775	0.519
Multi-Layer FFNN	0.579	0.748	0.396	0.617	0.194	0.836	0.773	0.469
RNN	**0.642**	**0.773**	**0.484**	**0.656**	**0.473**	**0.886**	**0.814**	**0.585**

Zhao2020

Zhao2020 produces 239 embeddings frames for one minute audio so each frame belongs to 0.25 s resolution. Considering the results across various models, the conclusion is the same as the other two embeddings model, and RNN performs better. From the single layer FFNN model to RNN it shows a 16.8% increase in jaccard scores.

Table 4.3. *F1 and jaccard scores for zhao2020's single embeddings frame*

Model	**Jaccard**	**F1 Score**						
		Water tap	**Door**	**Yelling**	**Human Fall**	**Music**	**Speech**	**Coughing**
Single layer FFNN	0.497	0.799	0.218	0.396	0.284	0.763	0.686	0.224
Multi-Layer FFNN	0.541	0.814	0.326	0.483	0.352	0.792	0.724	0.345
RNN	**0.666**	**0.839**	**0.470**	**0.723**	**0.432**	**0.889**	**0.777**	**0.570**

4.2 Recording-level Prediction Results

A comparison of the embeddings model is made on basis of one minute recording level results. We performed multiple experiments with all three embeddings model by taking various classification models. All these experiments were done to select embeddings model that perform better for our task. Let us now discuss the results of experiments.

Single Layer Feedforward Neural Network

The single layer FFNN is considered as a base model for recording-level predictions. This model results show very minor difference in all three embeddings model as can be seen in the Table 4.4.

Table 4.4. *Recording-level F1 and jaccard scores with single layer FFNN*

Embedding	**Jaccard**	**F1 Score**						
		Water tap	**Door**	**Yelling**	**Human Fall**	**Music**	**Speech**	**Coughing**
Openl3	0.541	0.630	**0.672**	0.656	**0.698**	0.791	0.770	**0.675**
Kumar2018	**0.559**	0.710	0.595	**0.668**	0.652	**0.887**	**0.834**	0.659
Zhao2020	0.552	**0.770**	0.654	0.639	0.629	0.877	0.831	0.556

Uni-directional RNN

The recording-level results from uni-directional RNN shows kumar2018 and zhao2020 performing better as compared to openl3. Kumar2018 and zhao2020 jaccard and F1 scores for all classes are almost similar or have minor differences.

Table 4.5. *Recording-level F1 and jaccard scores from uni-directional RNN*

Embedding	Jaccard	F1 Score						
		Water tap	Door	Yelling	Human Fall	Music	Speech	Coughing
Openl3	0.574	0.865	**0.666**	0.610	**0.707**	0.850	0.782	0.664
Kumar2018	0.629	0.787	0.650	0.753	0.661	**0.906**	**0.914**	**0.704**
Zhao2020	**0.630**	**0.887**	0.656	**0.754**	0.633	0.906	0.893	0.672

Bidirectional RNN

When we move from uni-directional RNN to bidirectional RNN only zhao2020 shows a little improvement but the other two embeddings almost remains the same. Zhao2020's jaccard score improved due to improvement in the water tap, door, yelling and music scores as can be seen in Table 4.6.

Table 4.6. *Recording-level F1 and jaccard scores from bidirectional RNN*

Embedding	Jaccard	F1 Score						
		Water tap	Door	Yelling	Human Fall	Music	Speech	Coughing
Openl3	0.580	0.872	0.667	0.644	**0.686**	0.841	0.804	0.654
Kumar2018	0.629	0.785	0.666	0.747	0.685	0.916	0.893	**0.718**
Zhao2020	**0.647**	**0.886**	**0.671**	**0.757**	0.662	**0.926**	**0.893**	0.682

Residual Connections with RNN

Residual connections in RNN is used to see the effect on performance and results are calculated on one minute recording-level. This method does not show much improvement as compared to bidirectional RNN. Openl3 and kumar2018 recording-level scores go down with this model and zhao2020 results show minor improvements.

Table 4.7. *Recording-level F1 and jaccard scores from residual connections with RNN*

Embedding	Jaccard	F1 Score						
		Water tap	Door	Yelling	Human Fall	Music	Speech	Coughing
Openl3	0.556	0.833	0.599	0.626	0.684	0.811	0.772	0.668
Kumar2018	0.619	0.778	0.656	0.732	0.667	0.897	**0.908**	**0.699**
Zhao2020	**0.653**	**0.911**	**0.679**	**0.765**	**0.684**	**0.898**	0.884	0.699

Attention

Results improved by using an attention model for recording-level predictions. Across different embeddings, zhao2020 performed the highest and openl3 as the lowest. Overall kumar2018 and zhao2020 results improved as compared to previous experiments. Jaccard score for zhao2020 improved 5.1% from the previous highest score and 3.9% for kumar2018.

Table 4.8. *Recording-level F1 and jaccard scores from attention model*

Embedding	**Jaccard**	**F1 Score**						
		Water tap	**Door**	**Yelling**	**Human Fall**	**Music**	**Speech**	**Coughing**
Openl3	0.560	0.820	0.650	0.649	0.679	0.797	0.777	0.647
Kumar2018	0.668	0.825	0.702	0.777	0.728	**0.906**	**0.917**	0.723
Zhao2020	**0.704**	**0.913**	**0.757**	**0.789**	**0.783**	0.886	0.893	**0.745**

RNN with Pooling

Zhao2020 remains on top of the list by 70.4% recording-level jaccard score. We did an experiment by applying max pooling over the frame-level results of RNN to get recording-level results. RNN with max pooling display promising results and improved jaccard score for all embeddings as compared to their previous highest scores. Jaccard score of zhao2020, kumar2018 and openl3 improved 3%, 1.3% and 2.9% respectively.

Table 4.9. *Recording-level F1 and jaccard scores from RNN with max pooling*

Embedding	**Jaccard**	**F1 Score**						
		Water tap	**Door**	**Yelling**	**Human Fall**	**Music**	**Speech**	**Coughing**
Openl3	0.589	0.887	0.663	0.641	0.696	0.853	0.807	0.673
Kumar2018	0.681	0.834	0.696	0.768	0.735	**0.928**	0.920	0.751
Zhao2020	**0.734**	**0.927**	**0.757**	**0.804**	**0.805**	0.916	**0.923**	**0.770**

As we get favorable results from RNN with max pooling, so we decided to test some other pooling techniques as shown in Table 2.2. We tried various pooling techniques mentioned in Table 4.10 with zhao2020 embeddings because it was on top of the list in our previous results. RNN with Max pooling and exponential softmax pooling gave better results as compared to other pooling. Overall, if we look at jaccard score max pooling technique remains on top with a 73.4% score.

Table 4.10. *Recording-level F1 and jaccard scores of zhao2020 with various pooling techniques*

Pooling	Jaccard	F1 Score						
		Water tap	Door	Yelling	Human Fall	Music	Speech	Coughing
RNN-Maxpool	**0.734**	**0.927**	**0.757**	0.804	**0.805**	**0.916**	**0.923**	0.770
RNN-Expnential Softmax	0.729	0.925	0.750	**0.809**	0.786	0.907	0.917	**0.784**
RNN-Linear Softmax	0.445	0.675	0.497	0.649	0.689	0.673	0.753	0.696
RNN-Average Pooling	0.643	0.862	0.690	0.746	0.675	0.914	0.885	0.699

kumar2018 vs kumar2020

Comparison is made between kumar2018 and kumar2020 to see the effect on recording-level results with RNN max pooling. Kumar2020 performed better and shows 2.3% increase in jaccard score along with an increase in F1 scores for most of the classes, and can be seen with bold face in the Table 4.11.

Table 4.11. *Recording-level F1 and jaccard scores comparison of kumar2018 and kumar2020*

Embedding	Jaccard	F1 Score						
		Water tap	Door	Yelling	Human Fall	Music	Speech	Coughing
Kumar2018	0.681	0.834	0.696	0.768	**0.735**	0.928	**0.920**	0.751
Kumar2020	**0.704**	**0.932**	**0.726**	**0.792**	0.726	**0.936**	0.910	**0.766**

After doing all these experiments, we shortlisted RNN with max pooling and residual RNN with max pooling model for further experiments. Regarding embeddings, we decide to proceed with zhao2020 and kumar2020.

Train with Various Resolution Labels

We performed multiple experiments by varying the training technique and keeping the evaluation technique same. We train multiple models with various time resolution labels (1 minute, 10 s and 0.25 s strong labels) and then use these models to generate recording level results on our test data. Zhao2020 embeddings are used in all of these experiments and the RNN with max pooling as classification model.

Table 4.12. *Comparison of using various resolution labels during training by using RNN-maxpool as classifier and zhao2020 embeddings*

Label Resolution	Jaccard	F1 Score						
		Water tap	Door	Yelling	Human Fall	Music	Speech	Coughing
1 Minute weak Label	0.734	0.919	**0.765**	0.809	**0.796**	**0.923**	**0.921**	0.775
10 seconds label	**0.734**	**0.919**	0.756	**0.820**	0.789	0.902	0.910	**0.797**
Strong Labels	0.699	0.899	0.733	0.797	0.758	0.866	0.895	0.770

We find out the 1 minute resolution and 10 s resolution weak labels are performing almost equally well as can be seen by jaccard score in Table 4.12. Although the difference is very small but training with strong labels gave little bit lower results.

We did the same experiments as above but with a different model to verify things. Residual RNN-maxpool also produce the same conclusion as simple RNN-maxpool as shown in the Table 4.13.

Table 4.13. *Comparison of using various resolution labels during training by using residual RNN-maxpool as classifier and zhao2020 embeddings*

Label Resolution	**Jaccard**	**F1 Score**						
		Water tap	**Door**	**Yelling**	**Human Fall**	**Music**	**Speech**	**Coughing**
1 Minute weak Label	0.732	**0.924**	**0.766**	0.796	**0.792**	**0.919**	**0.919**	0.777
10 seconds label	**0.734**	0.922	0.759	**0.812**	0.789	0.891	0.918	**0.796**
Strong Labels	0.701	0.901	0.739	0.805	0.756	0.861	0.893	0.777

Both the above experiments are also done with kumar2020 to see the effect of the changing training method on these embeddings. The conclusion remains identical to zhao2020's embeddings by keeping the 1 minute resolution and 10 s resolution weak labels on top. RNN-maxpool and residual RNN-maxpool results lead to the same opinion as shown in Tables 4.14 and 4.15 respectively.

Table 4.14. *Comparison of using various resolution labels during training by using RNN-maxpool as classifier and kumar2020 embeddings*

Label Resolution	**Jaccard**	**F1 Score**						
		Water tap	**Door**	**Yelling**	**Human Fall**	**Music**	**Speech**	**Coughing**
1 Minute weak Label	0.704	**0.932**	**0.726**	0.792	**0.726**	**0.936**	0.910	0.766
10 seconds label	**0.707**	0.932	0.719	**0.795**	0.702	0.911	**0.925**	0.757
Strong Labels	0.695	0.927	0.707	0.787	0.683	0.885	0.916	**0.769**

Table 4.15. *Comparison of using various resolution labels during training by using residual RNN-maxpool as classifier and kumar2020 embeddings*

Label Resolution	**Jaccard**	**F1 Score**						
		Water tap	**Door**	**Yelling**	**Human Fall**	**Music**	**Speech**	**Coughing**
1 Minute weak Label	0.704	**0.929**	0.716	0.790	**0.724**	**0.934**	0.912	0.759
10 seconds label	**0.707**	0.927	**0.729**	0.796	0.702	0.906	**0.917**	**0.771**
Strong Labels	0.699	0.921	0.719	**0.803**	0.699	0.880	0.915	0.764

Train with Four Target Classes

We did an experiment by predicting only four target classes and consider three non-target classes as background. The first experiment is done by taking the one minute weak labels during training and evaluation. The results were not so much different in terms of F1 score for individual classes. We can see water tap, door and human fall F1 score increase if we move to the prediction of four classes, but the improvement is not that significant.

Table 4.16. *Comparison of 7 vs 4 target classes by training and evaluating with 1 minute weak labels using zhao2020*

No. of Classes	**Jaccard**	**F1 Score**						
		Water tap	**Door**	**Yelling**	**Human Fall**	**Music**	**Speech**	**Coughing**
7 Target Classes	**0.734**	0.919	0.764	**0.809**	0.796	0.923	0.921	0.776
4 Target Classes	0.702	**0.924**	**0.768**	0.798	**0.805**			

The same experiment is done by training the model with 10 s resolution weak labels and the conclusion just remains the same as in the above experiment.

Table 4.17. *Comparison of 7 vs 4 target classes by training with 10 s and evaluating with 1 minute weak labels using zhao2020*

No. of Classes	**Jaccard**	**F1 Score**						
		Water tap	**Door**	**Yelling**	**Human Fall**	**Music**	**Speech**	**Coughing**
7 Target Classes	**0.734**	0.919	0.756	**0.820**	0.789	0.903	0.911	0.797
4 Target Classes	0.709	**0.934**	**0.762**	0.816	**0.800**			

Embeddings Model Selection

For extracting privacy-preserving features from audio, we used openl3, kumar2018 and zhao2020 embeddings model. Embeddings extracted from all these models are provided to various classification models to have a comparison among these embeddings. Jaccard score is the cumulative score of all events present in data and one value displays the overall result with one number, so we use it as an evaluation metric for this comparison.

After obtaining the recording-level classification results from various models, we find out that the zhao2020 embeddings model performed better on our synthetic dataset. As shown in Figure 4.1 single layer FFNN model is giving similar outputs for all three types of embeddings, but in all of the other classification models, zhao2020 embeddings outperform the other embeddings models on our synthetic dataset.

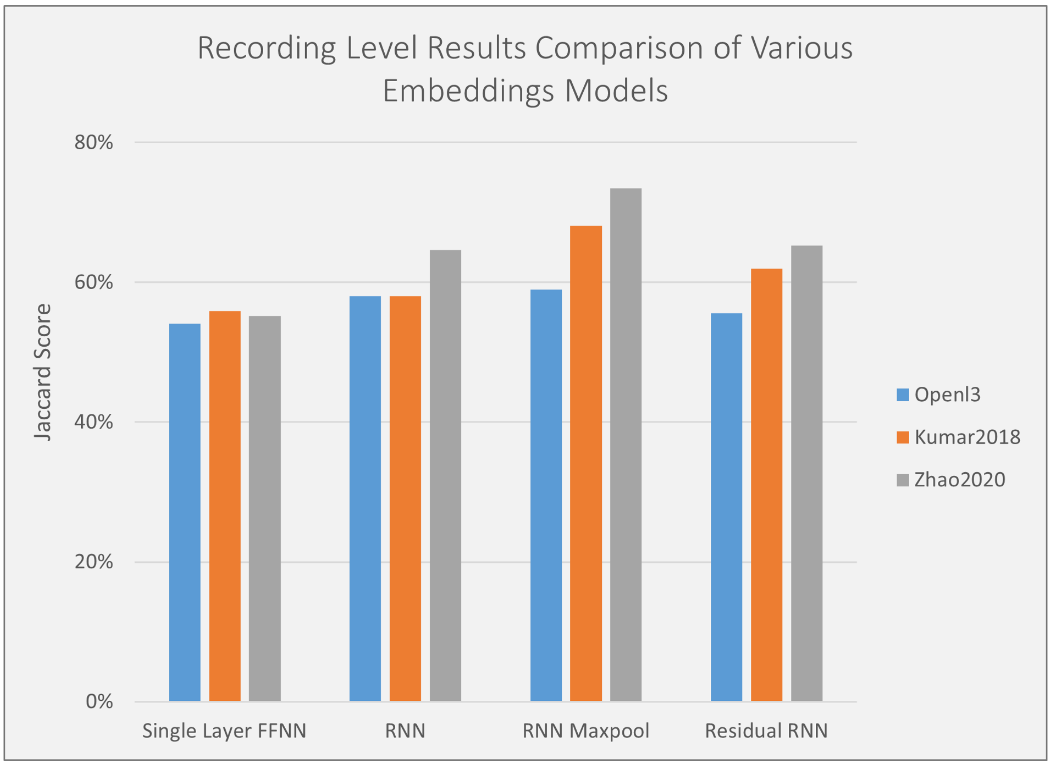


Figure 4.1. *Embeddings model selection based on recording level results*

Classification Model Selection

Zhao2020 is selected as the best performing embeddings extraction model on our synthetic dataset, so we use these embeddings with various classification models. Recording-level results are obtained from multiple classification models to have a comparison among them. RNN with max pooling came out the best performing classification model with embeddings as an input, and give Jaccard score of 73.4% as shown in Figure 4.2.

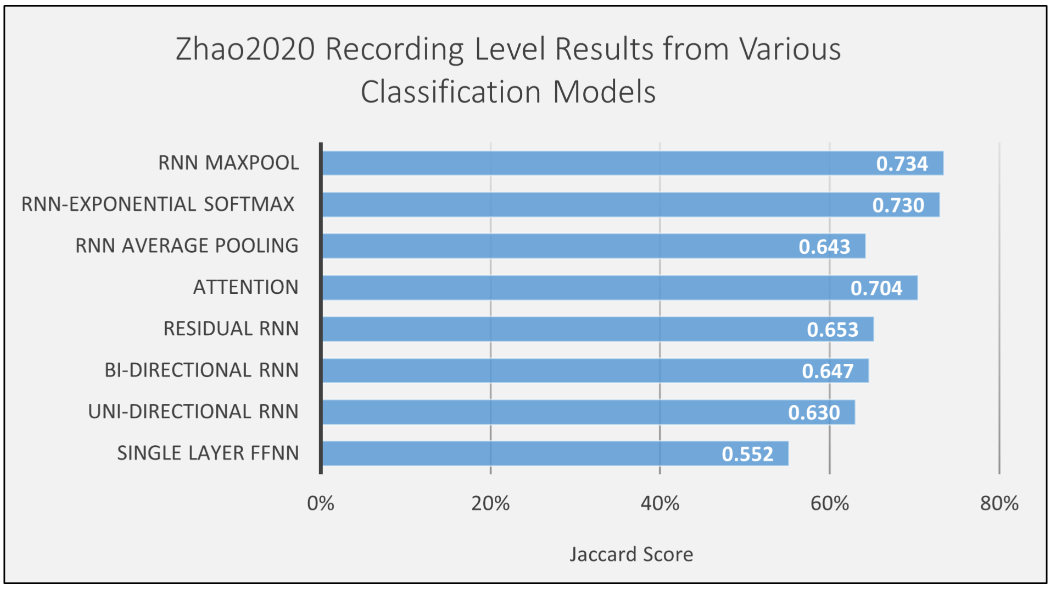


Figure 4.2. *Classification model selection from recording level results*

For a detailed overview of the results concerning different events, we consider the F1 score for each event. Zhao2020 as an embeddings model and RNN with max pooling are selected according to previous given results and now the detailed behavior of various classes will be done with help of the F1 score. Recording level F1 scores of single layer FFNN and RNN maxpool for each event is shown in Figure 4.3, and all events show a significant increase in F1 score as we move from single layer FFNN to RNN maxpool. Water tap, music and speech F1 score reaches above 90%, while all other classes are above 75%.

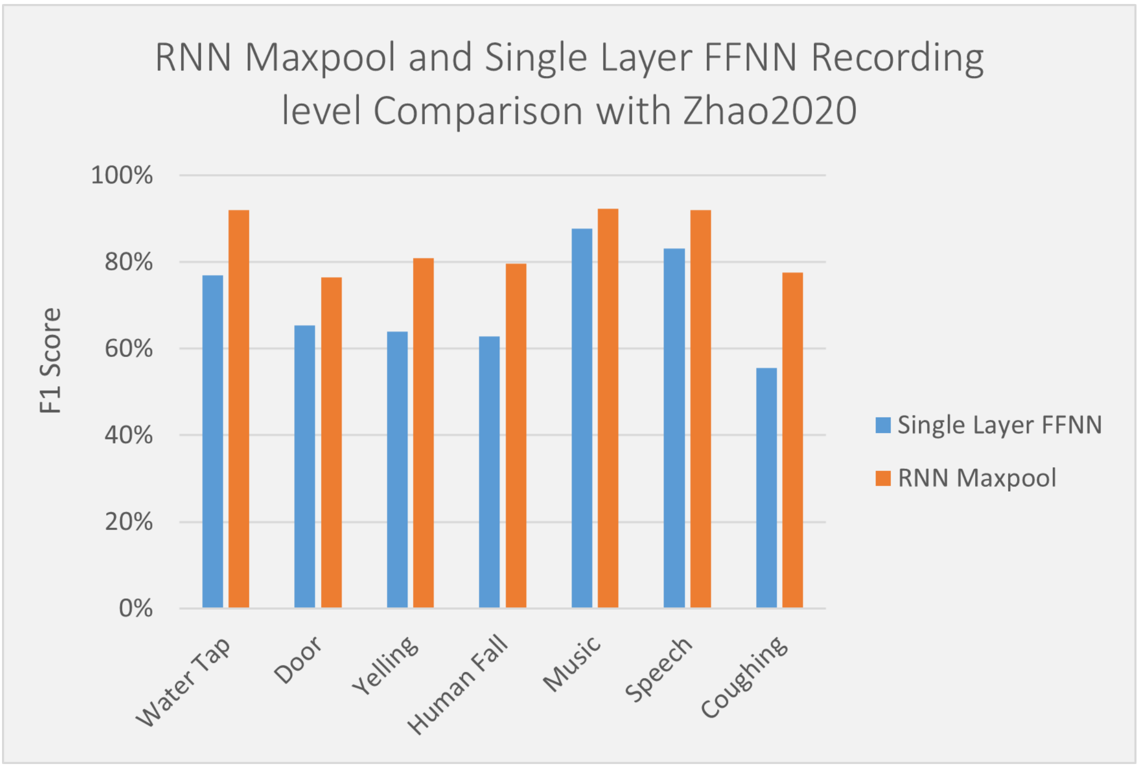


Figure 4.3. *Recording level results comparison of RNN maxpool and single layer FFNN with zhao2020*

5 CONCLUSION

In this thesis, we did sound event detection in a real-life environment by considering the healthcare monitoring problem. We created a synthetic dataset with a door, water tap, yelling, human fall, music, speech and coughing sounds. Door, water tap, yelling and human fall are taken as target classes, while music, speech and coughing are non-target classes. These sounds are quite common in daily life, so this dataset could be used as training material in many applications e.g health care monitoring, home security and activity measurement in a house.

Privacy of the people in real life recording is a serious concern and need to be handled accordingly. For having privacy-preserving features from audio, we consider openl3, kumar2018 and zhao2020 embeddings extraction models. We extract embeddings from these three models and provided them to various classification models to see the difference in performance. After obtaining recording-level results from various experiments, we concluded that embeddings extracted from zhao2020, when given to various classification models performed better as compared to the other embeddings models on our synthetic dataset.

For classification model selection, we did various experiments producing the recording-level outputs. We concluded that on our synthetic dataset, providing embeddings as an input, RNN model with a max pooling performed better as compared to all other classification models. This model makes frame-level predictions and then combine the frame results of one recording by applying max pooling for generating recording level output. For recording-level predictions, this model gives a jaccard score of 73.4% and F1 scores above or equal to 80% for most of the classes on our synthetic dataset.

We did experiments with various training techniques, where we use different resolution labels for calculating loss during the training of the model. Training done during these experiments by using 1 minute resolution weak labels, 10 s resolution weak labels and strong labels, while in evaluation results are calculated on a recording level. Training done with one minute weak labels outperforms the other two training techniques on our synthetic dataset. Considering the real-life scenario of healthcare monitoring, we want to do recording level (one minute) predictions and one minute weak labels training performed well, so we finalize this training technique for our task.

Future Work

The synthetic dataset created in this thesis could be improved by adding impulse responses during the soundscape creation to make the data more realistic. Some more sound events like paper flipping and laughing could be added because these are also common sounds in homes.

This sound event detection model could be used in real-life monitoring of nursing homes and home care centers. The microphones will be installed at the monitoring site that records the audio and sends it to the local server present at the site as shown in Figure 5.1.

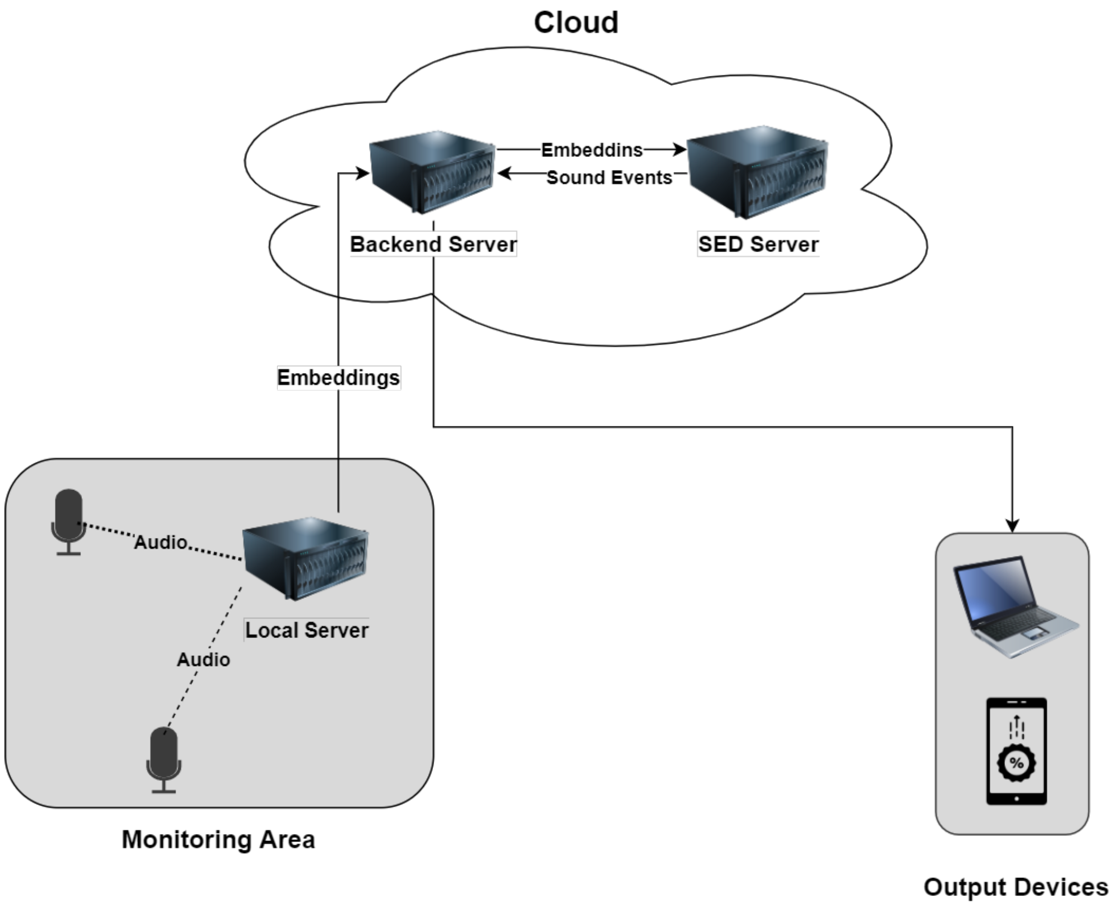


Figure 5.1. *SED system in a real life application*

The local server will process the audio and extract the privacy-preserving features (embeddings) and send it to the backend server on the cloud. The backend server will send the embeddings to the SED system and receive the predictions. The predictions will be sent to a mobile app or web site where a person can see the activity detail.

REFERENCES

[1] Wang, D. and Brown, G. J. *Computational Auditory Scene Analysis: Principles, Algorithms, and Applications*. Wiley-IEEE press, 2006.

[2] Kumar, A. and Raj, B. Audio event and scene recognition: A unified approach using strongly and weakly labeled data. *2017 International Joint Conference on Neural Networks (IJCNN)*. 2017, 3475–3482.

[3] Heittola, T., Mesaros, A., Virtanen, T. and Eronen, A. Sound event detection in multisource environments using source separation. *Machine Listening in Multisource Environments*. 2011.

[4] Virtanen, T., Plumbley, M. D. and Ellis, D. *Computational Analysis of Sound Scenes and Events*. Springer, 2018.

[5] Carone, G. and Costello, D. Can Europe afford to grow old. *Finance and Development* 43.3 (2006), 28–31.

[6] Droghini, D., Squartini, S., Principi, E., Gabrielli, L. and Piazza, F. Audio Metric Learning by Using Siamese Autoencoders for One-Shot Human Fall Detection. *IEEE Transactions on Emerging Topics in Computational Intelligence* (2019), 1–11.

[7] Salamon, J., MacConnell, D., Cartwright, M., Li, P. and Bello, J. P. Scaper: A library for soundscape synthesis and augmentation. *2017 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*. 2017, 344–348.

[8] Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation* 9.8 (1997), 1735–1780.

[9] Parascandolo, G., Huttunen, H. and Virtanen, T. Recurrent neural networks for polyphonic sound event detection in real life recordings. *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2016, 6440–6444.

[10] Chung, J., Gulcehre, C., Cho, K. and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *NIPS 2014 Workshop on Deep Learning*. 2014.

[11] Amores, J. Multiple instance classification: Review, taxonomy and comparative study. *Artificial Intelligence* 201 (2013), 81–105.

[12] Wang, Y., Li, J. and Metze, F. A comparison of five multiple instance learning pooling functions for sound event detection with weak labeling. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2019, 31–35.

[13] Wikipedia contributors. *Cross-validation (statistics) — Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/w/index.php?title=Cross-`

validation_(statistics)&oldid=975192923. [Online; accessed 17-September-2020]. 2020.

[14] Mesaros, A., Heittola, T. and Virtanen, T. Metrics for polyphonic sound event detection. *Applied Sciences* 6.6 (2016), 162.

[15] Cakir, E., Heittola, T., Huttunen, H. and Virtanen, T. Polyphonic sound event detection using multi label deep neural networks. *2015 International Joint Conference on Neural Networks (IJCNN)*. 2015, 1–7.

[16] Cakır, E., Parascandolo, G., Heittola, T., Huttunen, H. and Virtanen, T. Convolutional recurrent neural networks for polyphonic sound event detection. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 25.6 (2017), 1291–1303.

[17] Lecun, Y., Bottou, L., Bengio, Y. and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86.11 (1998), 2278–2324.

[18] Xu, Y., Kong, Q., Wang, W. and Plumbley, M. D. Large-scale weakly supervised audio classification using gated convolutional neural network. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, 121–125.

[19] Gemmeke, J. F., Ellis, D. P., Freedman, D., Jansen, A., Lawrence, W., Moore, R. C., Plakal, M. and Ritter, M. Audio set: An ontology and human-labeled dataset for audio events. *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2017, 776–780.

[20] Kumar, A., Khadkevich, M. and Fügen, C. Knowledge transfer from weakly labeled audio using convolutional neural network for sound events and scenes. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, 326–330.

[21] Wang, Y., Neves, L. and Metze, F. Audio-based multimedia event detection using deep recurrent neural networks. *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2016, 2742–2746.

[22] Holters, M., Corbach, T. and Zölzer, U. Impulse response measurement techniques and their applicability in the real world. *Proceedings of the 12th International Conference on Digital Audio Effects (DAFx-09)*. 2009, 1–5.

[23] Nagrani, A., Chung, J. S. and Zisserman, A. VoxCeleb: A Large-Scale Speaker Identification Dataset. *Proceedings of the Interspeech*. 2017, 2616–2620.

[24] Panayotov, V., Chen, G., Povey, D. and Khudanpur, S. Librispeech: an asr corpus based on public domain audio books. *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2015, 5206–5210.

[25] Snyder, D., Chen, G. and Povey, D. *MUSAN: A Music, Speech, and Noise Corpus*. arXiv:1510.08484v1. 2015. eprint: 1510.08484.

[26] Mesaros, A., Heittola, T. and Virtanen, T. TUT database for acoustic scene classification and sound event detection. *2016 24th European Signal Processing Conference (EUSIPCO)*. 2016, 1128–1132.

[27] Cramer, J., Wu, H., Salamon, J. and Bello, J. P. Look, Listen, and Learn More: Design Choices for Deep Audio Embeddings. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2019, 3852–3856.

[28] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L. and Lerer, A. Automatic differentiation in PyTorch: NIPS Autodiff Workshop: The Future of Gradient-based Machine Learning Software and Techniques. (2017).

[29] McFee, B., Raffel, C., Liang, D., Ellis, D. P., McVicar, M., Battenberg, E. and Nieto, O. librosa: Audio and Music Signal Analysis in Python. *Proceedings of the 14th Python in Science Conference*. Vol. 8. 2015, 18–25.

[30] Gencoglu, O., Virtanen, T. and Huttunen, H. Recognition of acoustic events using deep neural networks. *2014 22nd European Signal Processing Conference (EUSIPCO)*. 2014, 506–510.

[31] Schuster, M. and Paliwal, K. K. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing* 45.11 (1997), 2673–2681.

[32] He, K., Zhang, X., Ren, S. and Sun, J. Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, 770–778.

[33] Su, P., Ding, X.-R., Zhang, Y.-T., Liu, J., Miao, F. and Zhao, N. Long-term blood pressure prediction with deep recurrent neural networks. *2018 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI)*. 2018, 323–328.

[34] Kong, Q., Xu, Y., Wang, W. and Plumbley, M. D. Audio set classification with attention model: A probabilistic perspective. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, 316–320.

[35] Xu, Y., Kong, Q., Huang, Q., Wang, W. and Plumbley, M. D. Attention and Localization Based on a Deep Convolutional Recurrent Model for Weakly Supervised Audio Tagging. *Interspeech*. 2017.

A APPENDIX

Scaper's Soundscape with Label

Scaper generates audio mixture as well as their annotation file as shown in Figure A.1. This is a bit simple scenario where sound events are not overlapping with each other.

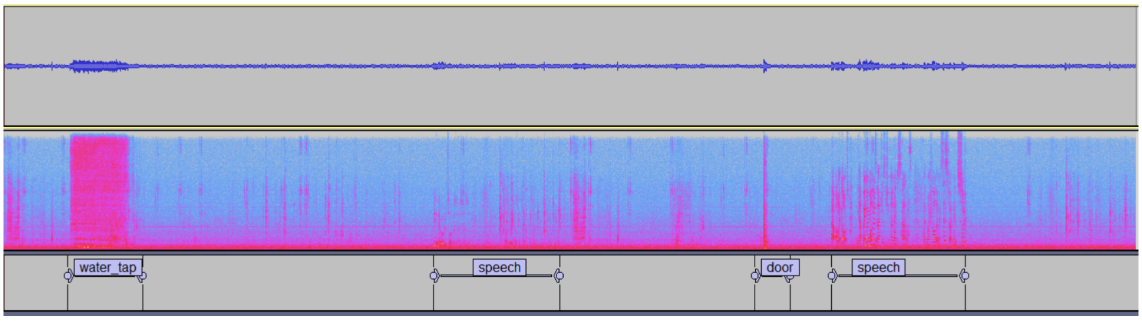


Figure A.1. *Simple scenario of soundscape*

One complex scenario of the audio mixture is shown in Figure A.2 where multiple sound events are happening at the same time.

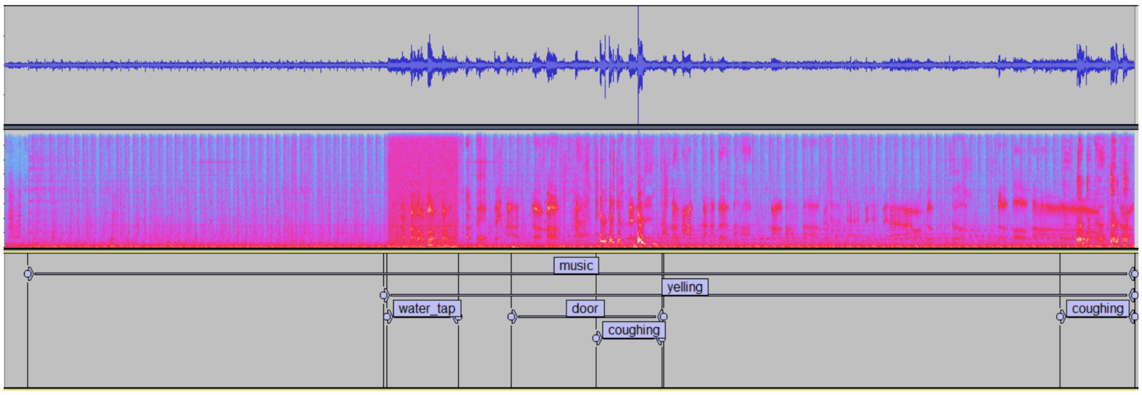
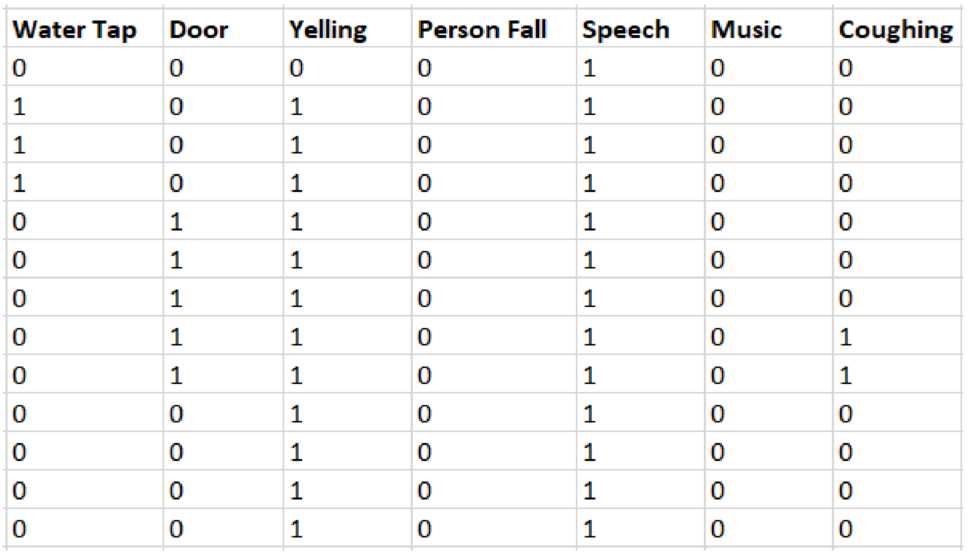


Figure A.2. *Complex scenario of soundscape*

Ground Truth Labels

Frame wise ground truth labels are generated for seven classes with help of an annotation file provided by scaper. The structure of ground truth labels is shown in Figure A.3. The 1 in a frame means that a specific event is present in that frame while 0 indicates the absence of that event.

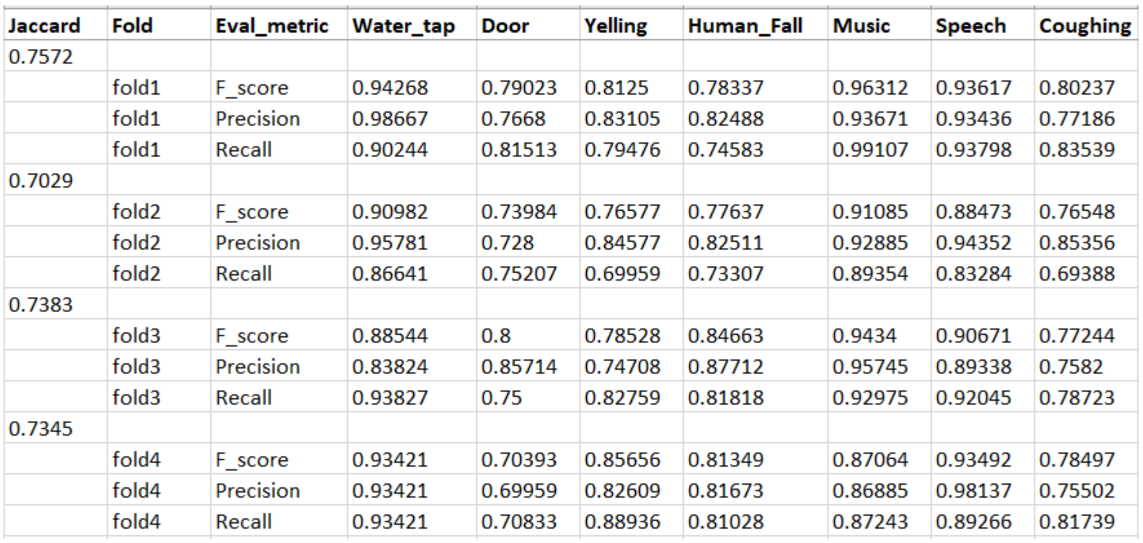


Water Tap	Door	Yelling	Person Fall	Speech	Music	Coughing
0	0	0	0	1	0	0
1	0	1	0	1	0	0
1	0	1	0	1	0	0
1	0	1	0	1	0	0
0	1	1	0	1	0	0
0	1	1	0	1	0	0
0	1	1	0	1	0	0
0	1	1	0	1	0	1
0	1	1	0	1	0	1
0	0	1	0	1	0	0
0	0	1	0	1	0	0
0	0	1	0	1	0	0
0	0	1	0	1	0	0

Figure A.3. *Ground truth labels structure*

Detailed Output Results for One Experiment

We have four folds of data, and we did cross-validation during the evaluation of different models. The detailed overview of the results that we get after one experiment is shown in Figure A.4. The F1 scores, precision and recall are calculated for every event in each fold of data. The results that are considered for evaluation are by taking the average of these four folds results.



Jaccard	Fold	Eval_metric	Water_tap	Door	Yelling	Human_Fall	Music	Speech	Coughing
0.7572									
	fold1	F_score	0.94268	0.79023	0.8125	0.78337	0.96312	0.93617	0.80237
	fold1	Precision	0.98667	0.7668	0.83105	0.82488	0.93671	0.93436	0.77186
	fold1	Recall	0.90244	0.81513	0.79476	0.74583	0.99107	0.93798	0.83539
0.7029									
	fold2	F_score	0.90982	0.73984	0.76577	0.77637	0.91085	0.88473	0.76548
	fold2	Precision	0.95781	0.728	0.84577	0.82511	0.92885	0.94352	0.85356
	fold2	Recall	0.86641	0.75207	0.69959	0.73307	0.89354	0.83284	0.69388
0.7383									
	fold3	F_score	0.88544	0.8	0.78528	0.84663	0.9434	0.90671	0.77244
	fold3	Precision	0.83824	0.85714	0.74708	0.87712	0.95745	0.89338	0.7582
	fold3	Recall	0.93827	0.75	0.82759	0.81818	0.92975	0.92045	0.78723
0.7345									
	fold4	F_score	0.93421	0.70393	0.85656	0.81349	0.87064	0.93492	0.78497
	fold4	Precision	0.93421	0.69959	0.82609	0.81673	0.86885	0.98137	0.75502
	fold4	Recall	0.93421	0.70833	0.88936	0.81028	0.87243	0.89266	0.81739

Figure A.4. *Detailed output results structure for one experiment*