Audio Event Detection

ELEC-E5510 Speech recognition project

Group 11: Anand Umashankar 765837, Anssi Moisio 474694

18th December 2019

1 Introduction

Automatic audio event detection, also called sound event detection (SED) or acoustic event detection, is the task of identifying events in an acoustic signal. These events can be of any sort, for example a car passing by or a dog barking. In the SED task, an audio signal is usually converted into labels of the corresponding sound events present in the auditory scene. In contrast to audio *context* recognition, SED aims to isolate discrete incidents from audio rather than identifying the broad acoustic environment, e.g. an auditorium or a street. However, detected sound events can be used also as mid-level-representation in the audio context recognition task (Heittola, 2019). Acoustic event detection can also be distinguished from acoustic event classification. According to Temko et al. (2006), acoustic event detection refers to both localisation of events in audio samples and identifying the event, whereas acoustic event classification refers only to the latter task. In practice, localisation of the event means finding the onset and offset times of an event in a continuous audio signal, whereas event classification means identifying the class of events that the isolated event is an instance of.

Sound event detection has many applications including indexing and retrieval from multimedia databases (Wang et al., 2016) and surveillance (Foggia et al., 2015). Goetze et al. (2012) developed an application of SED in health care, aimed for older people that require support in their daily lives. By monitoring the sound events as a part of a larger emergency monitoring system that incorporates also various other sources of information, possible situations of emergency can be identified and appropriate actions can be taken. For example, the SED module in this system could detect a scream of the user, or a sound of the user falling down, and the system would call for help based on an assessment of the situation.

The approach to solving a SED problem depends on the properties of the task. There can be a lot of variation between different SED tasks, including differences in the event classes to be identified (e.g. events in an office environment or events in a street or both), and different kinds of audio data (isolated events or a continuous signal). Compared to speech, sound events are usually more varied in all of their spectral and temporal features, speech being usually one sound event class. This can make it tricky to, for example, select the parameters for the feature extraction. The ontology of sound events is more open-ended than that of speech sounds. There is a closed set of phonemes in a language, and a range of sounds that can be produced by the human vocal tract, whereas the source of a sound event can be any phenomenon in the environment. On the other hand, this makes the sound event classes often quite different from each other in their (e.g. spectral) characteristics compared to speech sounds, which should make classifying them easier.

In this work, we implement and evaluate audio event classifiers, that is, we do not address the problem of isolating the events but only classify events that have already been isolated. We use data from the Audio Set (Gemmeke et al., 2017), which consists of annotated audio samples from YouTube videos. We implement two different classifiers for the task: a support vector machine system and a convolutional neural network system. The Python code for this project can be accessed from Aalto GitLab[1] by Aalto students and staff (Aalto GitLab users).

[1] `https://version.aalto.fi/gitlab/moisioa3/sed-project`

2 Literature study

In this section we describe three classifier types that have been used in automatic sound event detection: hidden Markov models (HMM), support vector machines (SVM), and deep neural networks (DNN). Initially, we experimented with HMMs in this project, but later restricted our experiments only to SVMs and DNNs due to lack of time and the fact that SVMs and DNNs are simpler to use.

2.1 Hidden Markov Models

Hidden Markov models are based on the idea that a system goes through sequences of hidden states which cause observations. Each observation has a probability of being generated by each hidden state, and each hidden state pair has a transition probability t_{ij} that describes how probable it is to move from state s_i to state s_j. For detecting sound events, one way to use HMMs is to train one HMM for each class of sound events. After learning the parameters, the HMM will estimate the probability that a sound sample (observation sequence) is caused by an instance of this class of sound events.

Ma et al. (2006) used a HMM-based approach to classify acoustic environments into 12 different classes (e.g. office, street). They trained one HMM per class of acoustic environment, using the Viterbi segmentation, state-based clustering, and Baum-Welch re-estimation. They used three-second-long samples of sound from which they extracted 39-dimensional feature vectors. These features include 12 MFCCs, a one-dimensional log energy term, and the velocity (also called differential or delta) and acceleration (delta-delta) features for each of these dimensions. They divided the audio samples into nine states after evaluating different numbers of states from 1 to 19 and finding 9 to be the optimal number of states.

After evaluating their model, they note that acoustic environment models are vulnerable to prominent sound events. For example, telephone ringing in an office can overwhelm all other sounds in the environment for a period of time, possibly causing incorrect classification of the environment. They suggest a hierarchical model where the HMMs are divided onto two levels: environments (e.g. office, street) and sound events (e.g. telephone ringing, car passing). The environment level is classified first, and if the classification is uncertain sound events can be used as additional cues. The HMM classifier works reasonably well on their data set for both acoustic environment classification and sound event classification, achieving overall accuracy values of 75% and 85%, respectively.

2.2 Support vector machines

SVMs are a supervised learning algorithm. An SVM model divides the space of data points into two subspaces with the hyperplane that maximises the margin between the hyperplane and the nearest data point. Guo and Li (2003) used a SVM model to classify sounds to 16 different classes, such as "crowds", "oboe", and "water". They used 409 audio samples from Muscle Fish data set (Wold et al., 1996) and extracted features including the the total power, subband powers, brightness, bandwidth, and pitch of each frame, as well as the MFCCs. They used a binary tree structure to solve the multiple class problem. The structure is similar to a sports tournament playoff phase, where sixteen teams (classes of sounds) play one-vs-one matches (space dichotomy by an SVM) until all but one team (class) have lost a game. The data point is then classified to the winner class.

2.3 Deep Neural Networks (DNN)

2.3.1 Convolutional Neural Networks (CNN)

Convolutional neural networks are a type of neural network (NN) architecture developed to overcome drawbacks of NNs when dealing with spatial structure data. CNNs were inspired by the processing of human visual cortex. By converting audio signals to spectrograms, CNNs are applied on the spectrogram outputs to classify audio events. A CNN consists of three basic components: convolutional layers, pooling layers, and fully-connected layers. Several works have been published which use CNN for sound event detection. These works have obtained better results than most of the other systems in both term error rate and f-score. (Dang et al., 2017)

Qian and Woodland (2016) have developed very deep convolutional neural networks up to 10 layers deep. Their proposed new model was evaluated on the Aurora4. Their best deep CNN system achieves a word error rate (WER) of 8.81%, and further 7.99% with auxiliary feature joint training. The final joint decoding scheme with LSTM-RNN gives a WER of 7.09%. This is the best published result on the Aurora 4 task, even without feature enhancement or sequence training.

2.3.2 Recurrent Neural Networks (RNN)

Recurrent neural networks (RNNs) are a family of NNs for processing sequential or temporal data. The advantage of CNNs is that they can effectively process spatial data with large width and height. On the contrary, the powerful point of RNNs are that they can scale to much longer sequences. Also CNNs required fixed size input while RNNs can handle sequences of variable length. In recurrent networks, a hidden layer with self connections acts as memory that accumulates information over time from an input sequence. Several works (Schroder et al., 2017) (Parascandolo et al., 2016) (Adavanne et al., 2016) have recently used RNNs for addressing sound event detection tasks. The proposed methods are based on bidirectional LSTM (Long Short Term Memory) or GRU (Gated Recurrent Unit) architectures using log mel-band energies as features. In comparison to DNN and CNN architectures, RNNs work better with sequential input data (Dang et al., 2017).

(Graves et al., 2013) have used combination of deep, bidirectional Long Short-term Memory RNNs. They have achieved state-of-the-art results in phoneme recognition on the TIMIT database. The three main conclusions of their work are:

1. LSTM works much better than tanh for this task. This is because LSTMs are better to handle temporal information.

2. Bidirectional LSTM has a slight advantage over unidirectional LSTM.

3. Depth is more important than layer size.

2.3.3 Convolutional recurrent neural networks (CRNN)

RNNs are neural networks that are specialized for processing time-series data. RNNs can integrate information from earlier time windows and even may have connections that go backward in time, presenting a theoretically unlimited context information. However, an audio input presents its features on both time and frequency domains. While RNNs only work well on time domains, CNNs can apply linear convolutional filters on both time and frequency domains of the local features. In order to take advantage of both approaches, a combination of convolutional networks and recurrent networks often referred to as a convolutional neural network (CRNN) has been adopted for the sound event detection task. CRNN outperforms all previous sound event detection methods (Dang et al., 2017).

3 Methods

We extract different features from the audio files using the Librosa Python library (McFee et al., 2019). We generate the mel band spectrogram, log mel band energies, MFCCs, zero crossing rate, root mean square energy and delta and delta-delta features for each audio sample. Using these features, we train a support vector machine as well as a deep convolutional neural network on a subset of the data set that we download. We create the SVM model using the scikit-learn Python library (Pedregosa et al., 2011), and the CNN model using TensorFlow (Abadi et al., 2015) with the Keras API (Chollet et al., 2015).

4 Data Set

We use annotated audio event samples from the Audio Set (Gemmeke et al., 2017), which is a data set and an ontology of 527 classes of sound events organised in a hierarchy. Most of the samples are 10 seconds long, but some are shorter if the original YouTube video is shorter than 10 seconds. Each sample is from a different YouTube video to avoid bias (Gemmeke et al., 2017). The samples can have more than one label in Audio Set, but we classify them only to one class to make the task simpler for this project. We downloaded about 9700 annotated audio samples in total from the 8 classes we selected for our project: ”Acoustic Guitar”, ”Bark”, ”Bell”, ”Explosion”, ”Laughter”, ”Siren”, ”Sneeze”, and ”Thunder”. The classes were selected to represent most of the hyperclasses (classes on the highest level of the class hierarchy), i.e. ”human sounds”, ”animal”, ”music”, ”sounds of things”, and ”natural sounds”. We selected only 8 classes to make the size of the data set easier to manage and the classifying problem easier. We have filtered the samples so that they cannot have more than one label from the group of 8 classes we use. The downloaded data set is fairly balanced, but the last two classes have fewer data than the other classes. Table 8 lists the total number of data in each class that we used in this project. We made a split of 80:20 to training and test data, and further split the training data to training and evaluation data with varying ratios.

	Acoustic Guitar	Bark	Bell	Explosion	Laughter	Siren	Sneeze	Thunder
Training	1007	1270	1065	1250	1229	1322	580	141
Test	227	317	266	312	307	331	145	35
Total	1134	1587	1331	1562	1536	1653	725	176

Table 1: Number of samples in each class.

Examples of the Audio Set can be listened from `https://research.google.com/audioset/index.html`.

5 Experiments

5.1 Experiments with the SVM model

We experimented with the SVM model using different features from the audio samples:

- mel band spectrogram
- mel frequency cepstral coefficients
- zero crossing rate
- delta and delta-delta
- root mean square energy

We tried the mel spectrogram as the input of the SVM, which is a less processed (or abstracted) feature type compared to MFCCs. Spectrograms work well as an input to a CNN network, because CNNs can detect patterns in images. However, SVMs work better with more abstract features. We got a classification accuracy of about 58% (F_1 score 0.58) with similar parameters for the mel spectrogram as defined in section 5.2 for the experiments with CNNs.

Using the MFCCs, we experimented with different parameters. The most important parameters for MFCCs are the window length and the number of mel bands. The optimal frequency resolution we found was about 40 mel bands. Figure 1 depicts the effect the number of mel bands had on the SVM F_1 score. From this graph we can see that increasing the frequency resolution after 40 bands does not improve the results for the SVM.

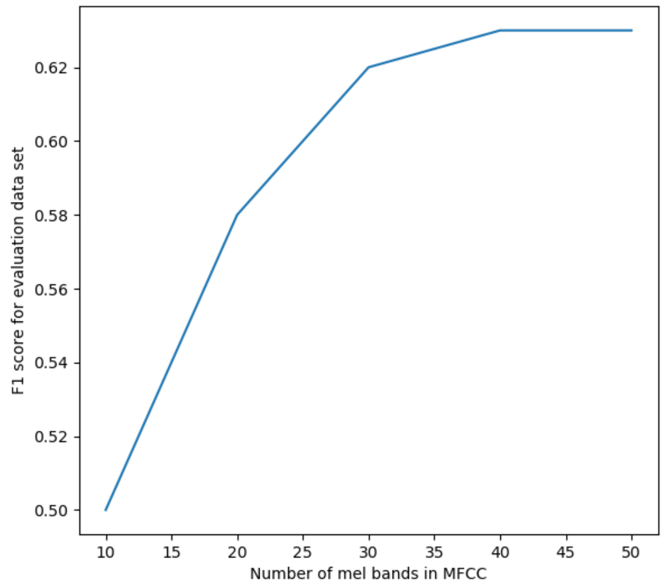


Figure 1: SVM classification accuracy as a function of the number of MFCC mel bands.

Using the MFCCs with a window length of 2048 samples (with a sampling rate of 22050Hz, this means about 90ms in time) with 50% hop length and 40 mel bands, we got the best result for the SVM: about 0.65 F_1 score (64% precision). Interestingly, the window length can be extended up to more than a second (about 30,000 samples) with the F_1 score dropping only to about 0.60. This probably means that the SVM classifies the data mostly based on global spectral characteristics of the audio, because a finer temporal resolution improves the results only a little. Increasing the window length means that the the Fourier transform is calculated over longer periods of audio, creating spectral features that are averaged over longer time periods.

The experiments with the delta and delta-delta features did not improve the results. These differential features capture how much the signal changes in time, and how much the change changes. We speculate that, similarly to the window length, these features are not as useful as the spectral features for the SVM with our classes of audio events. The data we use does have temporal structure, but we were not able to capture it effectively with the features that we extracted for the SVM. One reason for this might be that the temporal structure has different time scales in different classes of sound events. For example, to capture how the pitch changes in a siren sound, an appropriate window length might be somewhere around 500ms because the change is very slow. In contrast, a sneeze sometimes lasts less than 500ms in total, and a good time window might be around 5ms to detect the temporal structure of a sneeze. Therefore it is difficult to select the parameters (mainly the window length) that determine the temporal resolution of the features. A related difficulty is that the samples are usually 10 seconds long even if the sound event were a sneeze that lasts for half a second.

We experimented also with the zero crossing rate, which detects the smoothness of the signal (Bäckström et al., 2019). This had no positive effect to the results in our experiments. Similarly, the root mean square energy of each frame did not affect the results, when added to the features with the MFCCs.

After trying to improve the SVM results with more data and more complex features without success, we also tried reducing the feature dimensions and the training data set size to see if this has a negative effect on the results. As noted above, the window length can be increased without the a large decline in classification accuracy. A long window means shorter feature vectors, i.e. simpler features because of a lower temporal resolution. The effect of reducing the mel bands, i.e. spectral resolution, is more significant, as depicted in Figure 1. The training data set size can be reduced to about 140 samples per class with the accuracy dropping only to about 57%. This is about 10 times smaller training set size, so the drop from 64% accuracy to 57% is not very large. One reason for this is that 140 is the size of the smallest class (thunder), so this makes the classes balanced in training set size. However, this is not the only reason since the accuracy of the other classes does not drop much either. The reason we have a large training set size with more than 1000 samples in most classes is because neural networks can achieve a significantly better result by increasing the training set size. According to our experiments, the SVM results do not improve as much as the CNN results by increasing the data set size and making the input features more complex.

For all the experiments with the SVM, we used the radial basis function kernel. The other SVM hyperparameters were the default parameters in the sklearn.svm.SVC Python module.

5.2 Experiments with the CNN model

CNN is an architecture developed to overcome drawbacks of NNs when dealing with spatial structure data. A CNN consists of three basic components:

- Convolution layer. A convolution layer is a matrix that has smaller dimension than the input matrix. It performs a convolution operation with a small part of the input matrix having same dimension. The sum of the products of the corresponding elements is the output of this layer.
- Maxpooling layer. Maxpool passes the maximum value from amongst a small collection of elements of the incoming matrix to the output. Usually it is a square matrix.
- Fully Connected Layer. The final output layer is a normal fully-connected neural network layer, which gives the output.

The audio samples are converted into Mel Spectrogram Image and used for training the CNN. The image size is 256*128, where the image has 128 mel features for each time window. Then depending on the duration of the audio, the generated spectrogram is scaled up or down to 256. For an audio sample of 10 seconds, 400*128 will be the size of the image. This is then scaled to 256*128 The CNN network has around 12M parameters with 5 layers of convolutions intertwined with pooling layers and feeds into a fully connected layer. Considering the size of the dataset and the nature of the data this CNN is not too big and is easy to train. This sort of network is one of the standard CNN architecture that works well in many domains. A dropout layer with ratio 0.5 to avoid over fitting is added. Training was performed on 5200 audio samples and tested on 1950 samples (80:20 split). The architecture of the designed CNN is as follows.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 256, 128, 32)      320
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 254, 126, 64)      18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 127, 63, 64)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 125, 61, 128)      73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 62, 30, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 60, 28, 256)       295168
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 30, 14, 256)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 28, 12, 512)       1180160
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 14, 6, 512)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 12, 4, 1024)       4719616
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 6, 2, 1024)        0
_________________________________________________________________
```

```
dropout_1 (Dropout)          (None, 6, 2, 1024)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 12288)             0
_________________________________________________________________
dense_1 (Dense)              (None, 500)               6144500
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 4008
=================================================================
Total params: 12,436,124
Trainable params: 12,436,124
Non-trainable params: 0
_________________________________________________________________
```

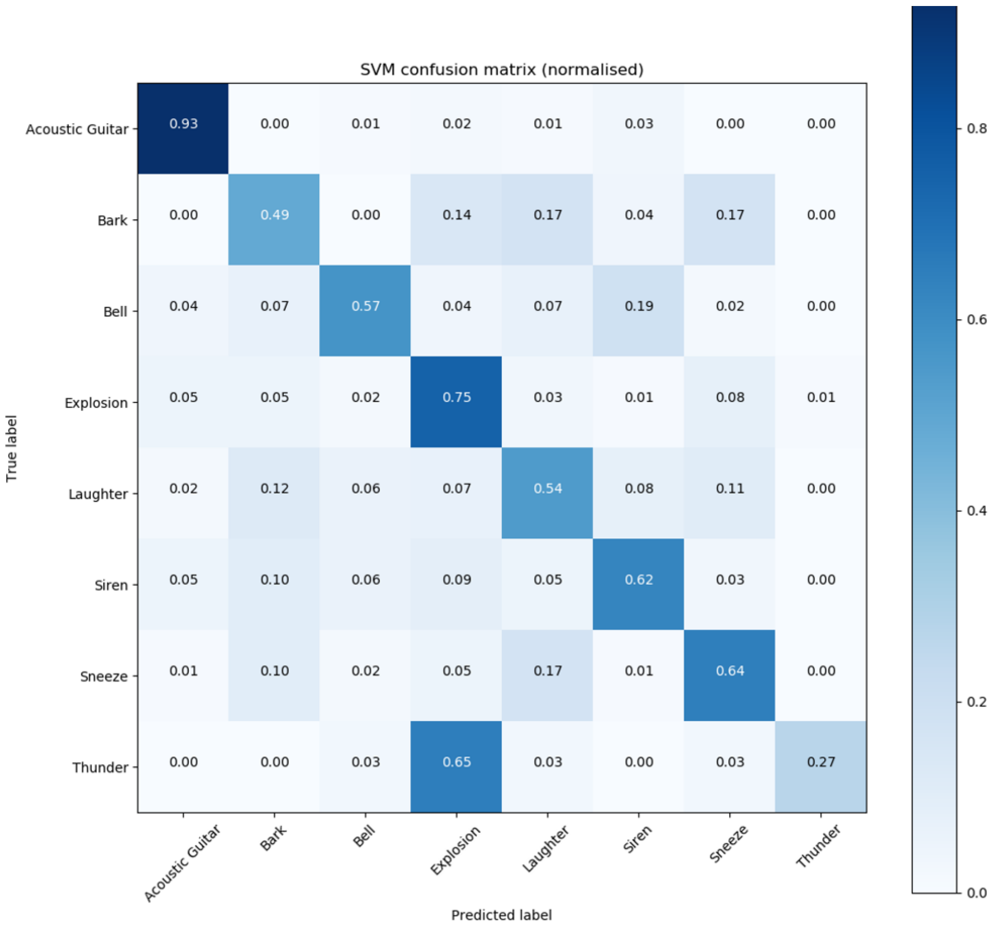
The CNN was trained with multiple hyperparameter configurations to obtain high accuracy. Experiments were also done to try out different configurations for generating the Mel Spectrum.

- *Learning_rate* = 0.0001 Increasing the learning rate led to very unstable results and did not converge well, accuracy wise. Lowering the learning rate took more epochs to train but did not lead to any increase in accuracy.
- *Epochs* = 14 Increasing the epochs reduced the training error but the test error was at its optimal value at usually 13/14th epoch.
- *MaxPool_Stride_Length*. Different at each layer. Tried to vary it to reduce parameters and make it more efficient. But it did not help with the accuracy.
- *Batch_Normalisation*. Tried to use Batch Normalisation before the dropout layer. This led to loss in accuracy. Hence it was removed in the final model.
- *Dropout_Ratio* = 0.5. Adding a dropout ratio helped the model to converge quicker by avoiding over fitting. The ratio was varied between 0.2-0.8. We eventually settled with value 0.5.
- *n_mels* = 128 The number of mel features was varied from 40 up to 512. Increasing it to 512 increases the parameters to a very high value and it becomes harder to train the model. 40 is too low but can be used if we need more efficiency. The accuracy dropped only around 1% when 40 is used.
- *n_fft* = 1024 Varying this did not impact the accuracy much. Varied between 512-2048. Further analysis needs to be done as to why this is the case.

6 Results

6.1 SVM

The SVM achieved a maximum accuracy of 64% on the test set. Figure 2 visualises the results in a confusion matrix. This result was achieved using only MFCCs as the input, with 40 mel bands and windows of about 90ms. The most significant confusions of classes are expected. Most test data in the thunder class are classified in the explosion class because thunder had fewer training data and we did not experiment with balancing the data. Classes in the hyperclass of *human sounds*, namely laughter and sneeze, are confused often. This might be due to the fact that the audio in these classes often include speech, even more than due to the sounds of sneeze and laughter being similar. The human sounds are also confused with bark. Again, this is probably due to audio in these classes including also speech and other human sounds. Some laughter might also sound like dog barks. As expected, the sound events of bells and sirens are also confused frequently because they are similar with each other. Acoustic guitar sounds seem to be the easiest to recognise, possibly because they have recognisable spectral characteristics as well as because the acoustic guitar audio has usually 10 seconds of guitar playing whereas a sneeze, for example, often lasts only about a second in a 10 second sample.



	Acoustic Guitar	Bark	Bell	Explosion	Laughter	Siren	Sneeze	Thunder
Acoustic Guitar	0.93	0.00	0.01	0.02	0.01	0.03	0.00	0.00
Bark	0.00	0.49	0.00	0.14	0.17	0.04	0.17	0.00
Bell	0.04	0.07	0.57	0.04	0.07	0.19	0.02	0.00
Explosion	0.05	0.05	0.02	0.75	0.03	0.01	0.08	0.01
Laughter	0.02	0.12	0.06	0.07	0.54	0.08	0.11	0.00
Siren	0.05	0.10	0.06	0.09	0.05	0.62	0.03	0.00
Sneeze	0.01	0.10	0.02	0.05	0.17	0.01	0.64	0.00
Thunder	0.00	0.00	0.03	0.65	0.03	0.00	0.03	0.27

Figure 2: Confusion matrix for the SVM model

6.2 CNN

CNN has achieved an accuracy of 83.53% on the test set. Below is the confusion matrix for the test set for the CNN. Most of the classifications have produced good results, the main source of error are between only a couple of classes. Thunder samples are classified as explosion. This is understandable because the nature of the audio is very similar and also Thunder had very few audio samples in the dataset and this accuracy may be improved by adding more thunder samples.

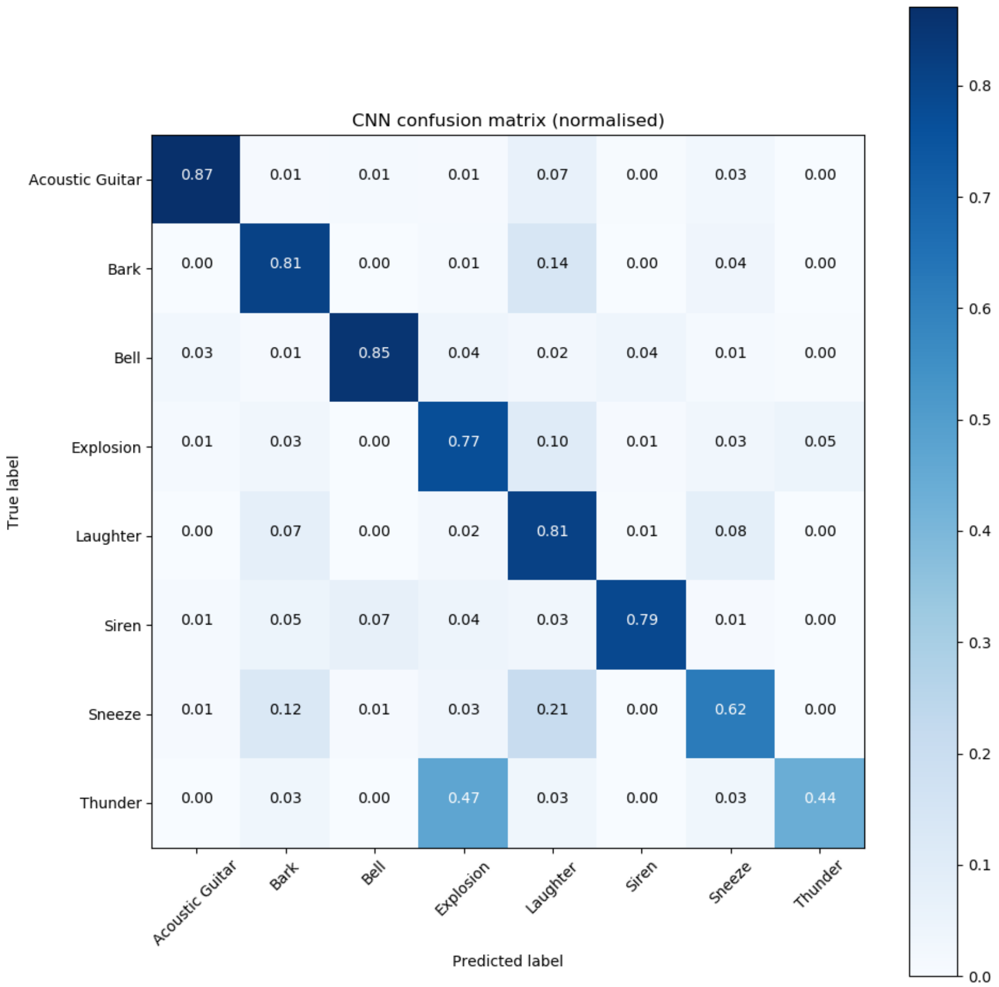


Figure 3: Confusion matrix for the CNN

7 Conclusions

Using the mel spectrograms as input and same amount of training data, the CNN achieves significantly better results (84%) than the SVM (58%). The CNN is able to learn to extract features from the mel spectrograms because of multiple hidden layers that recognise features of different levels of abstraction. In contrast, the SVM requires that the input features are more compressed and abstracted already before inputted to the SVM. Using MFCCs instead of mel spectrograms improves the results with the SVM a little (to 64%), but the CNN still performs much better. The better success of CNNs in this task might be because they can better recognise temporal structure in the audio, which is spatial structure when audio is converted to a spectrogram. In our experiments, the SVM does not benefit much from a fine temporal resolution.

7.1 Future Work

We had a class imbalance in our data set, which made the results for the class ”thunder” worse than for the other classes. Addressing the class imbalance would be one of the things to try if we continued the experiments further. We would also like to expand the dataset by adding more classes and try to adapt our models to a larger dataset. This way we could analyse the scalability of the models to larger problems. Another experiment we could do is to use MFCCs as features to a CNN or another Deep Neural Network to analyse the performance when using MFCCs.

8 Division of labor

Task	**Responsibility**
Introduction	Anssi
Literature Study - HMM & SVM	Anssi
Literature Study - DNN	Anand
Dataset Preparation	Anand
Feature Extraction	Anssi
Support Vector Machines	Anssi
Convolutional Neural Networks	Anand

Table 2: Division of Labour

9 Acknowledgements

We would like to thank our tutor Anja Virkkunen for her help and feedback throughout the project work.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Adavanne, S., Parascandolo, G., Pertilä, P., Heittola, T., and Virtanen, T. (2016). Sound event detection in multichannel audio using spatial and harmonic features. *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2016 Workshop (DCASE2016)*, pages 6–10. `http://urn.fi/URN:ISBN:978-952-15-3807-0`.

Bäckström, T., Räsänen, O., and Das, S. (2019). Introduction to speech processing. Accessed 16th December 2019 from: `https://wiki.aalto.fi/display/ITSP/Introduction+to+Speech+Processing`.

Chollet, F. et al. (2015). Keras. `https://keras.io`.

Dang, A., Vu, T. H., and Wang, J.-C. (Dec 2017). A survey of deep learning for polyphonic sound event detection. pages 75–78. IEEE. `https://ieeexplore.ieee.org/document/8336092`.

Foggia, P., Petkov, N., Saggese, A., Strisciuglio, N., and Vento, M. (2015). Reliable detection of audio events in highly noisy environments. *Pattern Recognition Letters*, 65:22–28.

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

Goetze, S., Schroder, J., Gerlach, S., Hollosi, D., Appell, J.-E., and Wallhoff, F. (2012). Acoustic monitoring and localization for social care. *Journal of Computing Science and Engineering*, 6(1):40–50.

Graves, A., Mohamed, A.-r., and Hinton, G. (May 2013). Speech recognition with deep recurrent neural networks. pages 6645–6649. IEEE. `https://ieeexplore.ieee.org/document/6638947`.

Guo, G. and Li, S. Z. (2003). Content-based audio classification and retrieval by support vector machines. *IEEE transactions on Neural Networks*, 14(1):209–215.

Heittola, T. (2019). Sound event detection. `http://www.cs.tut.fi/~heittolt/research-sound-event-detection`. Accessed 5-November-2019.

Ma, L., Milner, B., and Smith, D. (2006). Acoustic environment classification. *ACM Transactions on Speech and Language Processing (TSLP)*, 3(2):1–22.

McFee, B., Lostanlen, V., McVicar, M., Metsai, A., Balke, S., Thomé, C., Raffel, C., Lee, D., Zalkow, F., Lee, K., Nieto, O., Mason, J., Ellis, D., Yamamoto, R., Battenberg, E., Morozov, V., Bittner, R., Choi, K., Moore, J., Wei, Z., Seyfarth, S., nullmightybofo, Friesch, P., Stöter, F.-R., Hereñú, D., Thassilo, Kim, T., Vollrath, M., Weiss, A., and Weiss, A. (2019). librosa/librosa: 0.7.1. `https://doi.org/10.5281/zenodo.3478579`.

Parascandolo, G., Huttunen, H., and Virtanen, T. (Mar 2016). Recurrent neural networks for polyphonic sound event detection in real life recordings. pages 6440–6444.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python . *Journal of Machine Learning Research*, 12:2825–2830.

Qian, Y. and Woodland, P. C. (Dec 2016). Very deep convolutional neural networks for robust speech recognition. pages 481–488. IEEE. `https://ieeexplore.ieee.org/document/7846307`.

Schroder, J., Moritz, N., Anemuller, J., Goetze, S., Kollmeier, B., Schroder, J., Moritzn, Goetze, S., and Kollmeier, B. (2017). Classifier architectures for acoustic scenes and events. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 25(6):1304–1314. `http://dl.acm.org/citation.cfm?id=3149061`.

Temko, A., Malkin, R., Zieger, C., Macho, D., Nadeu, C., and Omologo, M. (2006). Clear evaluation of acoustic event detection and classification systems. In *International Evaluation Workshop on Classification of Events, Activities and Relationships*, pages 311–322. Springer.

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

Wold, E., Blum, T., Keislar, D., and Wheaton, J. (1996). Content-based classification, search, and retrieval of audio. *IEEE MultiMedia*, 3(3):27–36. `https://doi.org/10.1109/93.556537`.