CAPSTONE REPORT

Urban Sound

J. R. Verbiest

10 November 2019

TABLE OF CONTENTS

DEFINITION

PROJECT OVERVIEW

Environmental noise or noise pollution, which is defined as an unwanted or harmful outdoor sound created by human activity [1], is a growing problem in urban area. This noise pollution can affect the quality of life and health. Recent studies [2] have shown that exposure to noise pollution may increase the health risk. Therefore, decreasing the noise in the human environment can contribute to increase both the quality of life and health.

Environmental noise monitoring systems continuously measure the sound levels to quantify the noise level. However, it can be of interest to identify the type of noise source. By combining the noise level and the type of noise in real time we can describe the acoustic environment in a more complete way. Based on the outcome, actions can be taken to reduce the noise levels in urban areas. However, finding the type of noise can be a challenging task there an audio recording contains a mix of different noise sources. Here machine learning can be a helpful tool.

A way to build a real time monitoring system is the use of a low cost, small size, low power, wireless embedded device. Today, there is some attention to perform machine learning direct on these devices, so called machine learning on the edge, where we deploy a pre-trained neural network close to the sensor. However, because a limit memory footprint and compute resources the deployment of a neural network on an embedded device can become a challenge. Because of these limitations trade-off needs to be made between memory footprint, compute power and the accuracy of the neural network.

PROBLEM STATEMENT

The design of a real time smart embedded monitoring system is, given the limit timeframe of this capstone project, out of the scope. The main objective of this capstone is to take a first small step in the design of a real time smart embedded system for environmental sound classification (ESC). In this project we focus mainly on the feature engineering step and the neural network design.

ANALYSIS

DATASET AND EXPLORATION

An overview of suitable datasets for ESC can be found in [4-5]. For the project there are 3 datasets of interest: The Sounds of New York City (Sonyc), more specific the dataset used in the DCASE Challenge 2019 – Task 5 Urban Sound Tagging [6], ESC-50 [7] and the UrbandSound8k dataset [8-9]. For this project the UrbandSound8k is selected.

The UrbanSound8k is a dataset that contains 8732 labelled sound excerpts of urban sounds, in WAV format. These excerpts are less or equal to 4s. There are 10 classes defined: air conditioner, car horn, children plying, dog bark, drilling, engine idling, gun shot, jackhammer, siren and street music. The sampling rate, bit depth, and number of channels can vary from file to file. The files are pre-sorted into ten folds for cross validation and saved in folders named fold1 to fold10. Together with the dataset meta data is provided, as summarized in table 1.

Slice_file_name	The name of the audio file. The name takes the following format: [fsID]-[classID]-[occurrenceID]-[sliceID].wav, where: [fsID] = the Freesound ID of the recording from which this excerpt (slice) is taken [classID] = a numeric identifier of the sound class (see description of classID below for further details) [occurrenceID] = a numeric identifier to distinguish different occurrences of the sound within the original recording [sliceID] = a numeric identifier to distinguish different slices taken from the same occurrence
fsID	The Freesound ID of the recording from which this excerpt (slice) is taken
start	The start time of the slice in the original Freesound recording
end	The end time of slice in the original Freesound recording
salience	A (subjective) salience rating of the sound. 1 = foreground, 2 = background.
Fold	The fold number (1-10) to which this file has been allocated.

classID	A numeric identifier of the sound class: 0 = air_conditioner 1 = car_horn 2 = children_playing 3 = dog_bark 4 = drilling 5 = engine_idling 6 = gun_shot 7 = jackhammer 8 = siren 9 = street_music
class	The class name: air_conditioner, car_horn, children_playing, dog_bark, drilling, engine_idling, gun_shot, jackhammer, siren, street_music.

Table 1. Dataset meta data [9]

	Class	ClassID	fold 1	fold 2	fold 3	fold 4	fold 5	fold 6	fold 7	fold 8	fold 9	fold 10	total
1	air conditioner	0	100	100	100	100	100	100	100	100	100	100	1000
2	drilling	4	100	100	100	100	100	100	100	100	100	100	1000
3	jackhammer	7	120	120	120	120	120	68	76	78	82	96	1000
4	childeren playing	2	100	100	100	100	100	100	100	100	100	100	1000
5	engine idling	5	96	100	107	107	107	107	106	88	89	93	1000
6	street music	9	100	100	100	100	100	100	100	100	100	100	1000
7	dog bark	3	100	100	100	100	100	100	100	100	100	100	1000
8	siren	8	86	91	119	166	71	74	77	80	82	83	929
9	car horn	1	36	42	43	59	98	28	28	30	32	33	429
10	gun shot	6	35	35	36	38	40	46	51	30	31	32	374
		total	873	888	925	990	936	823	838	806	816	837	8732

Table 2: Number of audio files per class and per fold.

Table 2 shows the distribution of the audio per class and per fold. The class with the lowest number of audio files are car horn and gun shot with respectively 429 and 374 audio files.

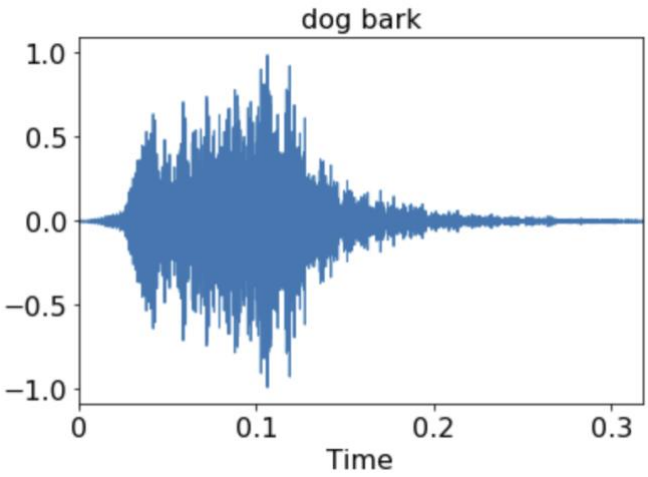


Figure 1: Example of an audio clip.

Figure 1 shows an example of an audio clip (dog bark). The audio file is loaded using LibROSA [10] as a mono channel, normalised with an amplitude value range between -1 and 1 with the default sample rate of 22050 Hz.

BENCHMARK

Salamon et al. [3] compares 5 different algorithms: decision tree (J48), k-NN (k = 5), random forest (500 trees), Support Vector Machine (SVM) (radial basis function kernel), and a baseline majority vote classier (ZeroR). The top performer is the SVM with a 67% classification accuracy for a 4 sec audio slice duration.

The SVM described in [3] will be used as benchmark model. The evaluation metrics is the average test accuracy, which is the number of correct predictions to the total number of predictions made, obtained after 10-fold cross validation.

PLATFORM

The architecture is built in the deep learning framework Keras (1) [14] with TensorFlow as backend. The training is performed on an OS Windows 7 computer, Intel Core i7-6800K CPU, 3.40GHz 128GB RAM and a NVIDEA GeForce GTX 1070 with 8 GB VRAM and 1920 CUDA Cores.

SOFTWARE AND LIBRARIES

The following software and libraries are used:

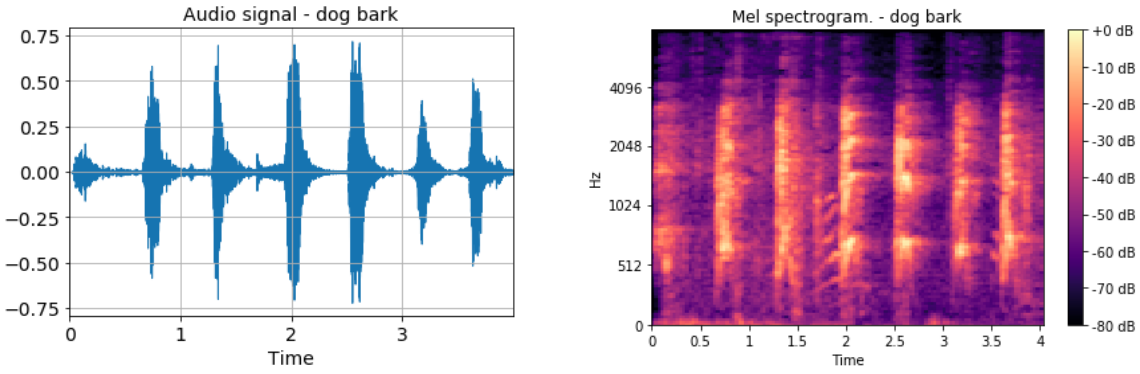
- Python version 3.7.3
- pandas version 0.24.2
- NumPy version 1.16.4
- Matplotlib version 3.1.0
- Scikit-learn version 0.21.2
- LibROSA version 0.6.3
- Keras version 2.2.4

(1) Why Keras. Currently (November 2019) X-Cube-AI supports various deep learning frameworks such as Keras and the quantization of Keras networks. Support for PyTorch, is planned in 2020. This is the reason why Keras is selected for this project.

METHODOLOGY

DATA PRE-PROCESSING

FEATURE ENGINEERING



(a) (b)

Figure 2: Example of a 4 sec audio clip (dog bark). (a) time signal, mono, sample rate 16kHz. (b) Mel spectrogram in decibel (dB), number of Mel bands = 128.

The audio clips are resampled to a sample rate of 16kHz [2], and mono channel is used. Given the audio clips are less or equal to 4 sec and the Convolutional Neutral Network (CNN) request a fixed length segment the audio clips, zero padding is applied if the signal is smaller than 4 sec. The 4 sec audio signals are translated to the Mel spectrogram in decibel (dB), with a number of Mel bands equal to 128, and final standardization is applied (mean = 0 and standard deviation = 1). The standardized Mel spectrogram is used as input feature for the CNN during training, validation and testing. In the capstone project no data augmentation is performed.

[2] The trained model will be deployed on an embedded device containing a microphone with a sampling frequency of 16kHz.

CONVOLUTION NEURAL NETWORK

PROCEDURE

A CNN is built, trained and optimized using fold 4 to 10 (training set = 66%), both fold 2 and 3 are used for the validation set (= 23%) and fold 1 is used for the test set (= 11%). The training set is used to train the CNN, the validation set is used to check how well the model is performing after each training epoch (i.e. check for over- or underfitting). The validation loss is used to compare the different trained models. The model with the lowest validation loss is selected and cross validation will be applied to obtain the final test accuracy.

In this project we keep the number of layers in the CNN as low as possible, and the target average test accuracy is set to ≥ 70%.

BASELINE CNN

As start point a baseline CNN is designed using two Conv2D layers followed by a Flatten layer and three Dense layers. In the training process the Adam optimizer is used ($\beta_1 = 0.9$, $\beta_2 = 0.999$, amsgrad=False), the loss function is the categorical cross entropy, the batch size is set to 125, and the number of training epochs is set to 50, no early stopping was done during training. The learning rate for the baseline CNN is set to 0.001.

```
baseline_model = models.Sequential()
baseline_model.add(Conv2D(32, (3,3), input_shape = (num_rows,num_columns,num_channels)))
baseline_model.add(Activation('relu'))
baseline_model.add(Conv2D(32, (3,3)))
baseline_model.add(Activation('relu'))
baseline_model.add(Flatten())
baseline_model.add(Dense(125))
baseline_model.add(Activation('relu'))
baseline_model.add(Dense(125))
baseline_model.add(Activation('relu'))
baseline_model.add(Dense(10,activation ='softmax'))
```

Figure 3: Keras implementation baseline CNN.

The number of output filters is 32 for both the Conv2D layers, the kernel size height = 3 and width = 3. The input and hidden Dense layer have both 125 nodes, the output Dense layer has 10 nodes, which equals the number of classes. The relu activation function is used for the two Conv2D and the two Dense layers and the softmax activation for de output Dense layer. The Keras code is shown in figure 3, the model summary is shown in figure 4.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 126, 124, 32)	320
activation_1 (Activation)	(None, 126, 124, 32)	0
conv2d_2 (Conv2D)	(None, 124, 122, 32)	9248
activation_2 (Activation)	(None, 124, 122, 32)	0
flatten_1 (Flatten)	(None, 484096)	0
dense_1 (Dense)	(None, 125)	60512125
activation_3 (Activation)	(None, 125)	0
dense_2 (Dense)	(None, 125)	15750
activation_4 (Activation)	(None, 125)	0
dense_3 (Dense)	(None, 10)	1260

Figure 4: Model summary baseline CNN

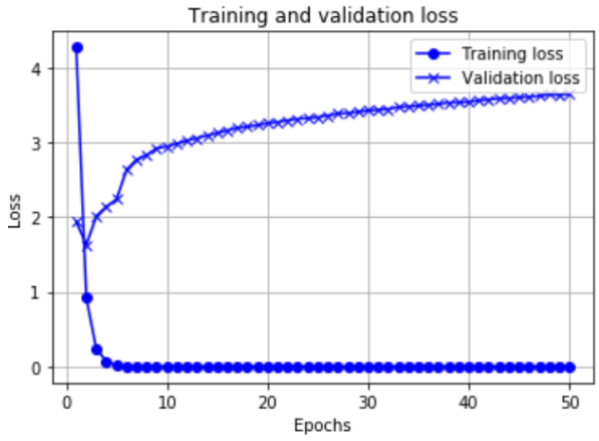


Figure 5: baseline CNN training and validation loss

Figure 5 shows the baseline CNN training and validation loss. We clearly observe overfitting. Already after 7 epochs a train accuracy of 100% is reach, after the 50 epoch the validation accuracy is 53.9%, the final test accuracy is 59.9%.

To reduce overfitting and to increase the accuracy refinements to the CNN are applied.

REFINEMENTS

To reduce overfitting and to improve the accuracy following refinements are applied to the baseline network:

- Add MaxPooling2D layer (reduce overfitting).
- Add an extra Conv2D-Conv2D-BatchNormalization-MaxPooling2D layers (improve accuracy).
- Increase the number of nodes in the Dense Layer (improve accuracy).
- Add Dropout layer in the Dense layer (reduce overfitting).
- Add L2 kernel regularization in the Dense layer (reduce overfitting).
- Decreased to learning rate to 0.0001.

FINAL CNN

The Keras implementation is shown in figure 5 and the model is summarized in figure 6.

```
model = models.Sequential()
model.add(Conv2D(32, (3,3), input_shape = (num_rows,num_columns,num_channels)))
model.add(Conv2D(32, (3,3), padding='same'))
model.add(MaxPooling2D((2,2), strides=2))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(64, (3,3), padding='same'))
model.add(Conv2D(64, (3,3), padding='same'))
model.add(MaxPooling2D((2,2), strides=2))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(128, (3,3), padding='same'))
model.add(Conv2D(128, (3,3), padding='same'))
model.add(MaxPooling2D((2,2), strides=2))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(256, (3,3), padding='same'))
model.add(Conv2D(256, (3,3), padding='same'))
model.add(MaxPooling2D((2,2), strides=2))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Flatten())
model.add(Dense(256,  kernel_regularizer=l2(0.1)))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(256, kernel_regularizer=l2(0.1)))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(10,activation ='softmax'))
```

Figure 6: Keras implementation final CNN.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 126, 124, 32)	320
conv2d_2 (Conv2D)	(None, 126, 124, 32)	9248
max_pooling2d_1 (MaxPooling2	(None, 63, 62, 32)	0
batch_normalization_1 (Batch	(None, 63, 62, 32)	128
activation_1 (Activation)	(None, 63, 62, 32)	0
conv2d_3 (Conv2D)	(None, 63, 62, 64)	18496
conv2d_4 (Conv2D)	(None, 63, 62, 64)	36928
max_pooling2d_2 (MaxPooling2	(None, 31, 31, 64)	0
batch_normalization_2 (Batch	(None, 31, 31, 64)	256
activation_2 (Activation)	(None, 31, 31, 64)	0
conv2d_5 (Conv2D)	(None, 31, 31, 128)	73856
conv2d_6 (Conv2D)	(None, 31, 31, 128)	147584
max_pooling2d_3 (MaxPooling2	(None, 15, 15, 128)	0
batch_normalization_3 (Batch	(None, 15, 15, 128)	512
activation_3 (Activation)	(None, 15, 15, 128)	0
conv2d_7 (Conv2D)	(None, 15, 15, 256)	295168
conv2d_8 (Conv2D)	(None, 15, 15, 256)	590080
max_pooling2d_4 (MaxPooling2	(None, 7, 7, 256)	0
batch_normalization_4 (Batch	(None, 7, 7, 256)	1024
activation_4 (Activation)	(None, 7, 7, 256)	0
flatten_1 (Flatten)	(None, 12544)	0
dense_1 (Dense)	(None, 256)	3211520
activation_5 (Activation)	(None, 256)	0
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 256)	65792
activation_6 (Activation)	(None, 256)	0
dropout_2 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 10)	2570

Figure 7: Final CNN (model summary)

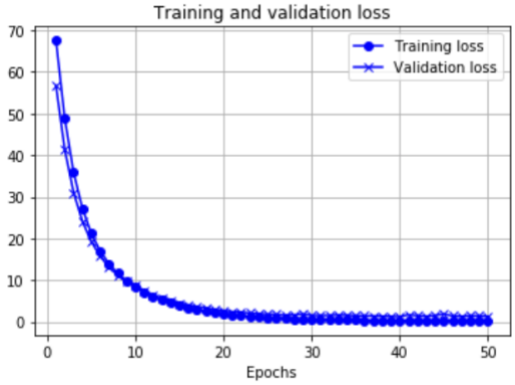


Figure 8: final CNN training and validation loss

Figure 8 shows the training and validation loss. The validation accuracy after 50 epochs is 70.1% the final test accuracy is 74.1%. In the training process the Adam optimizer is used ($\beta_1 = 0.9$, $\beta_2 = 0.999$, amsgrad=False), the loss function is the categorical cross entropy, batch size is set to 125, and the number of training epochs is set to 50, no early stopping is done during training. The learning rate for the final CNN is set to 0.0001.

Accuracy	**Baseline CNN**	**Final CNN**
Validation	53.9%	70.1%
Test	59.9%	74.1%

Table 3: Comparison Validation accuracy after 50 epochs and the final test accuracy, baseline versus final CNN.

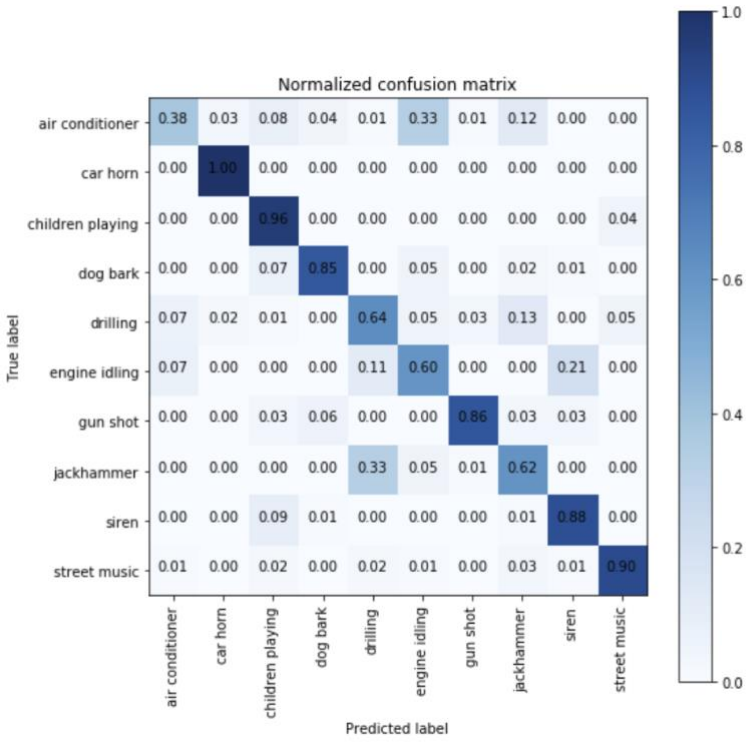


Figure 8: Normalized confusion matrix final CNN.

Table 3 shows the validation and test set accuracy of the final versus the baseline CNN. We observe an improvement in both the validation and test accuracy.

Figure 8 shows the normalized confusion matrix. Four classes have a correct prediction lower than 70%: air conditioner (38%), engine idling (60%), jackhammer (62%) and drilling (64%). We observe that for class jackhammer 33% is misclassified as drilling and for drilling 13% is misclassified as jackhammer. Also, for air conditioner 12% is misclassified as jackhammer and 33% misclassified as engine drilling. The hypothesis is that the vibrations produced by the air conditioner, drilling, engine idling and jackhammer translate into similar audio characteristics which are difficult to distinguished given the current features.

CROSS VALIDATION

10-Fold cross validation is performed on the optimized CNN. In the training process the Adam optimizer is used ($\beta_1 = 0.9$, $\beta_2 = 0.999$, amsgrad=False), the loss function is the categorical cross entropy, the batch size is set to 125, and the number of training epochs is set to 20, no early stopping was done during training. The learning rate for the cross validation is set to 0.0001.

Table 4 shows the average test accuracy of the CNN compared to the baseline SVM. The obtained average accuracy is 74.4%, 7.4% larger compared to the baseline SVM, with a standard deviation of 5.9%. No standard deviation for the baseline SVM was given in [3].

Baseline SVM [3]	CNN
67%	74.4% ± 5.9%

Table 4: average test accuracy.

Table 5 shows the correct prediction for the different folds and different classes. We observe that still 3 classes have a correct prediction lower than 70%: air conditioner, engine idling and jackhammer. We observe also large variation for these 3 classes between the different folds.

	correct prediction										
Class	**fold 1**	**fold 2**	**fold 3**	**fold 4**	**fold 5**	**fold 6**	**fold 7**	**fold 8**	**fold 9**	**fold 10**	**total**
air conditioner	50	22	25	60	73	62	92	31	39	66	52
car horn	100	81	100	59	70	96	93	90	97	94	88
children playing	94	81	91	40	90	92	83	82	80	97	83
dog bark	83	97	86	85	77	87	83	84	86	75	84
drilling	42	82	61	82	75	89	85	94	70	81	76
engine idling	65	37	40	60	62	47	62	58	94	80	61
gun shot	94	97	94	74	97	100	86	100	97	97	94
jackhammer	33	36	70	53	95	71	13	72	93	99	64
serin	87	67	88	87	90	58	90	44	99	59	77
street music	88	91	69	92	98	76	90	85	97	89	88
average test accuracy	**69.0**	**65.5**	**69.3**	**70.0**	**81.7**	**75.3**	**77.3**	**71.3**	**82.6**	**82.2**	**74.4**

Table 5: Correct prediction for each test fold.

CONCLUSION

In this project, first steps are taken in the design of a neural network, with a focus on feature engineering and the neural network design. Given an audio file the Mel spectrogram is extracted, and different neural networks are trained. To reduce overfitting, different strategies where explored such as MaxPooling2D, dropout and L2 kernel regularization, to improve the accuracy extra Conv2D-Conv2D-BatchNormalization-MaxPooling2D layers where added with different kernel sizes. Final, 10-fold cross validation is performed, the average test accuracy is calculated and the correct prediction for each class and fold is summarized. The final CNN results in an average test of 74.4%, 7.4% larger compared to the baseline SVM. During cross validation we observed that 3 classes have a correct prediction lower than 70%: air conditioner, engine idling and jackhammer. The hypothesis is that the vibrations produced by the air conditioner, engine idling and jackhammer translate into similar audio characteristics which are difficult to distinguished given the current features.

REFERENCES

[1] Directive 2002/49/EC of the European Parliament and of the Council of 25 June 2002 Relating to the Assessment and Management of Environmental Noise, Jun. 2002.

[2] L. Poon, The Sound of Heavy Traffic Might Take a Toll on Mental Health, CityLab. [Online]. Available: https://www.citylab.com/equity/2015/11/city-noise-mental-health-traffic-study/417276/

[3] J. Salamon, C. Jacoby and J. P. Bello, A dataset and Taxonomy for Urban Sound Research, 22nd ACM International Conference on Multimedia, Orlando USA, Nov. 2014. [Online]. Available: http://www.justinsalamon.com/uploads/4/3/9/4/4394963/salamon_urbansound_acmmm14.pdf

[4] Datasets Environmental sounds. [Online]. Available: http://www.cs.tut.fi/~heittolt/datasets

[5] Detection and Classification of Acoustic Scenes and Events. [Online]. Available: http://dcase.community

[6] DCASE2019 – Task 5: Urban Sound Tagging. [Online]. Available: http://dcase.community/challenge2019/task-urban-sound-tagging

[7] ECS-50: Dataset for Environmental Sound Classification. [Online]. Available: https://github.com/karoldvl/ESC-50

[8] Urban sound. [Online]. Available: https://urbansounddataset.weebly.com

[9] Urbansound8k dataset. [Online]. Available: https://urbansounddataset.weebly.com/urbansound8k.html

[10] libROSA. [Online]. Available: https://librosa.github.io/librosa/

[11] Jivitesh Sharma, Ole-Christoffer Granmo, Morten Goodwin, Environmental Sound Classification using Multiple Feature Channels and Deep Convolution Neural Networks. Version 2. (arXiv: 1908.11219 Submitted on 28 Aug 2019, last revised 25 Sept 2019, version 4). [Online]. Available: https://arxiv.org/abs/1908.11219

[12] Karol J. Piczak, Environmental sound classification with convolutional neural networks, IEEE International Workshop on Machine Learning for Signal Processing, Sept. 17-20, 2015, Boston, USA

[13] Urban Sound Classification, Part 2. Applying Convolutional Neural Network. [Online]. Available: http://aqibsaeed.github.io/2016-09-24-urban-sound-classification-part-2/

[14] Keras. [Online]. Available: https://keras.io