DCASE-MODELS: A PYTHON LIBRARY FOR COMPUTATIONAL ENVIRONMENTAL SOUND ANALYSIS USING DEEP–LEARNING MODELS

Pablo Zinemanas[1], *Ignacio Hounie*[2], *Pablo Cancela*[2], *Frederic Font*[1], *Martín Rocamora*[2], *Xavier Serra*[1]

[1] Music Technology Group, Universitat Pompeu Fabra, Barcelona, Spain,
name.surname@upf.edu
[2] Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay,
{ihounie, pcancela, rocamora}@fing.edu.uy

ABSTRACT

This document presents `DCASE-models`, an open–source Python library for rapid prototyping of environmental sound analysis systems, with an emphasis on deep–learning models. Together with a collection of functions for dataset handling, data preparation, feature extraction, and evaluation, it includes a model interface to standardize the interaction of machine learning methods with the other system components. This also provides an abstraction layer that allows the use of different machine learning backends. The package includes Python scripts, Jupyter Notebooks, and a web application, to illustrate its usefulness. The library seeks to alleviate the process of releasing and maintaining the code of new models, improve research reproducibility, and simplify comparison of methods. We expect it to become a valuable resource for the community.

***Index Terms*—** Python library, deep learning, audio classification, sound event detection, reproducibility

1. INTRODUCTION

Fuelled by an enormous increase in available data, substantially more-powerful computer hardware, and significantly improved algorithms, recent years have witnessed an explosion of machine learning methods across almost all research areas. In particular, deep–learning techniques have become ubiquitous since they have brought new state–of–the–art results in several research fields—such as computer vision, speech recognition, and natural language processing—, and have proved successful in realistic problems [1].

As a result, most modern signal processing methods and models are considerably more complex compared to those of a decade ago, and heavily rely on the data and software used for their implementation [2]. Besides, implementation details can have a profound effect on the reported performance of a method [3–5]. Therefore, it has become increasingly difficult to be able to reproduce the findings or to compare a new method to earlier ones, only based on the description of research systems found in publications [2, 5]. This can slow down progress in the research community, due to the amount of effort devoted to reimplementing methods and to deal with all details of different software packages, and datasets. For the optimal reuse of scientific research, numerous authors and institutions are advocating not only for open–access publications and data but also for the release of software and models that can be of use across a variety of domains [2, 6, 7]. This poses several challenges, such as the development of best practices guidelines to stick to, the need for long–term funds for those initiatives, and the coordination of research efforts–which after all is crucial to the quality of research [8].

The research community around the annual Detection and Classification of Acoustic Scenes and Events (DCASE) Workshop is also part of this phenomenon since most recent works involve deep–learning techniques. For instance, 50 out of 54 papers published at the proceedings of the DCASE 2019 Workshop use models based on deep neural networks in their experiments. Fortunately, the community is increasingly endorsing transparency and best practices for reproducible research. The DCASE Challenge provides an annual benchmark of methods for different tasks [9, 10], and since 2019 it gives specific awards for open–source and reproducible methods. There are numerous readily available datasets [11–18] and various authors publicly release software tools as well as easily usable and well documented implementations of their methods [19–22].

While all such resources are of great value for the progress of the field, there are still several opportunities to improve research reproducibility. For instance, usability aspects have to be considered (documentation, installability, etc.) so that a piece of code is useful rather than simply available. In addition, when trying to compare results of a new method with earlier methods, it is often the case that researchers have to reimplement those baselines or manage to find and modify an existing implementation. This is time–consuming and implies dealing with a variety of different coding conventions and software tools. Together with the intrinsic complexity of deep–learning methods, the process of delving into a given DCASE problem can be a tough one, especially for newcomers.

This paper introduces the first release of `DCASE-models`, an open-source Python library, whose main goal is to facilitate various aspects of the typical research pipeline of DCASE related problems with a particular emphasis on deep–learning models. The library has a careful design for easy extension and integration with other software tools. It offers an abstraction to common tasks, such as data preparation, data augmentation, feature extraction, model training, and evaluation. This allows for rapid prototyping of new methods and simplifies the efforts needed to release (and maintain) the code of a new model. Furthermore, the library aims to provide reference implementations of several baselines—including already trained models—to facilitate the comparison of methods. In this way, we strive for a low barrier to entry for students and researchers new to the field. Whenever possible, the library leverages from the original authors' implementations and existing software tools, such as `sed_eval` [21] or VGGish [23].

The package includes thorough documentation that covers the usage of the resources already available and describes the steps needed for extending the library, for instance, with new datasets, features and models. It also contains Python scripts, Jupyter Notebooks and a web interface, to illustrate the usefulness of the library.

2. DESIGN PRINCIPLES AND PRACTICES

The library provides a simple and lightweight set of basic components that are generally part of a computational environmental audio analysis system. Users can exploit the library functions to tackle various tasks—such as acoustic scene classification, sound event detection, and audio tagging—while experimenting with improvements or extensions of its components.

Apart from a collection of functions for dataset handling, data preparation, feature extraction, and evaluation (most of which rely on existing tools), `DCASE-models` includes a model interface to standardize the interaction of machine learning methods with the other system components. This also provides an abstraction layer to make the library independent of the backend used to implement the machine learning model (e.g. Keras,[1] PyTorch,[2] TensorFlow,[3] Scikit-learn[4]). The standardized behavior of the machine learning method implementation allows the comparison of different models in a straightforward manner. The library currently includes Keras implementations of several deep–learning models reported in the literature, which are ready to use with minimal effort. As a result, one can get an application with a few lines of code (see Section 4).

Regarding the usability of the library, the design and implementation are aimed to make it easy to learn and use. It is organized in a flat package layout with classes that define concise interfaces. All functions are thoroughly documented and include example code that demonstrates their usage. Besides, we follow PEP-8 recommendations to make sure code is readable and easy to follow. The documentation of the library is prepared using Sphinx and includes clear instructions on how to extend different components. The latest stable release can be smoothly installed from PyPI and only has requirements of other well-known, portable and tested packages.

Considering the sustainability and maintainability of the library, we strive for adopting modern open–source software development practices, as suggested in [2]. The code is released under the MIT license and all development is conducted on GitHub. This makes the project readily accessible for the community to use, test, contribute, and bring support. It also helps the release of updates by keeping a record of the changes and versions and incorporate other software development services, such as continuous integration testing.

3. LIBRARY ORGANIZATION AND DESCRIPTION

Figure 1 shows a diagram of `DCASE-models` main classes, which also includes some specializations of each base class that are available. Next, a description of the main classes and functionalities is presented, following the order of the typical pipeline: dataset preparation (3.1); data augmentation (3.2); feature extraction (3.3); data loading (3.4); data scaling (3.5); and model handling (3.6).

3.1. Dataset

This is the base class designed to manage a dataset, its paths, and its internal structure. It includes methods to download the data, resample the audio files, and check that both processes succeed.

The library covers several publicly available datasets related to different tasks. At the moment, these are: ESC [11] and UrbanSound8k [12] for audio classification; TAU Urban Acoustic Scenes 2019 and 2020 [14, 18] for acoustic scene classification; URBAN-SED [19], TUT Sound Events 2017 [13] and MAVD-traffic [17] for sound event detection; and FSDKaggle2018 [15] and SONYC-UST [16] for audio tagging. In next releases of the library other relevant datasets will be included.

Each dataset is implemented in the library as a class that inherits from `Dataset`. This design provides a common and simple interface to work with any dataset. For instance, to use the UrbanSound8k dataset, it is enough to initialize its class with the path to the data folder, as follows.

```
>>> dataset = UrbanSound8k(DATASET_PATH)
```

Then, the following methods are used to download the dataset and change its sampling rate (to 22050 Hz).

```
>>> dataset.download()
>>> dataset.change_sampling_rate(22050)
```

Most of the datasets devised for research include a fold split and a corresponding evaluation setup (e.g. 5–fold cross–validation). This fold split is generally carefully selected to avoid biases and data contamination [24]. In order to keep the results comparable to those reported in the literature, `DCASE-models` uses, whenever available, the predefined splits for each dataset. However, the user may define different splits or evaluation setups if needed.

3.2. AugmentedDataset

The previously defined `dataset` instance can be expanded using data augmentation techniques. The augmentations implemented so far are pitch–shifting, time–stretching, and white noise addition. The first two are carried out by means of `pysox` [25].

An augmented version of a given dataset can be obtained by initializing an instance of the `AugmentedDataset` class with the `dataset` as a parameter, as well as a dictionary containing the name and parameters of each transformation.

```
>>> aug_dataset = AugmentedDataset(dataset,
                                   augmentations)
```

After initialization, the following method will perform the actual augmentation and create new audio files for every dataset element according to the type and parameters of each augmentation.

```
>>> aug_dataset.process()
```

The augmented dataset is indeed an instance of `Dataset`, so it can be used as any other dataset in the following steps of the pipeline.

3.3. FeatureExtractor

This is the base class to define different types of feature representations. It has methods to load an audio file, extract features, and save them. It can also check if the features were already extracted.

Four types of feature representations have been implemented as specializations of the base class, namely Spectrogram, MelSpectrogram, VGGish [23], and Openl3 [26]. The first two are classic time–frequency representations which are implemented using `librosa` functions. The last two are pre–trained neural–network–based models that extract embeddings from the audio signal. Openl3 is a recently proposed neural network trained on audio–visual information. VGGish is a deep convolutional neural network trained on the Audioset dataset that is also used as a feature extractor.

A `FeatureExtractor` is initialized with some parameters. For instance, to define a `Spectrogram` feature extractor the parameters are: length and hop in seconds of the feature representation

[1]`https://keras.io`
[2]`https://pytorch.org`
[3]`https://www.tensorflow.org`
[4]`https://scikit-learn.org`

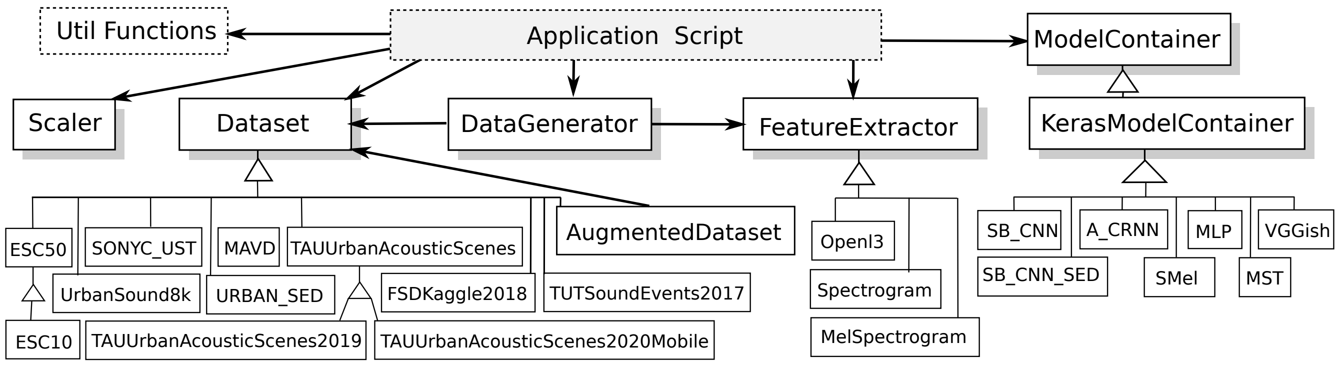

Figure 1: Class diagram of `DCASE-models` showing all base classes and some of the implemented specializations.

analysis windows (model's input); window length and hop size (in samples) for the short-time Fourier Transform (STFT) calculation; and the sampling rate. If the audio files are not sampled at this frequency, they are converted before calculating the features.

```
>>> features = Spectrogram(
        sequence_time=1.0, sequence_hop_time=0.5,
        audio_win=1024, audio_hop=512, sr=22050)
```

After initialization, the following method computes the features for each audio file in the dataset.

```
>>> features.extract(dataset)
```

Once the features are extracted and saved to disk, they can be loaded using `DataGenerator` as explained in the following.

3.4. DataGenerator

This class uses instances of `Dataset` and `FeatureExtractor` to prepare the data for model training, validation and testing. An instance of this class is created for each of these processes.

```
>>> data_gen_train = DataGenerator(
    dataset, features, train=True, folds=['train'])

>>> data_gen_val = DataGenerator(
    dataset, features, train=False, folds=['val'])
```

At this point of the pipeline, the features and the annotations for training the model can be obtained as follows.

```
>>> X_train, Y_train = data_gen_train.get_data()
```

Additionally, instances of `DataGenerator` can be used to load data in batches. This feature is especially useful for training models on systems with memory limitations.

3.5. Scaler

Before feeding data to a model, it is common to normalize the data or scale it to a fixed minimum and maximum value. To do this, the library contains a `Scaler` class, based on `scikit-learn` preprocessing functions, that includes `fit` and `transform` methods.

```
>>> scaler = Scaler("standard")
>>> scaler.fit(X_train)
>>> X_train = scaler.transform(X_train)
```

In addition, the scaler can be fitted in batches by means of passing the `DataGenerator` instance instead of the data itself.

```
>>> scaler.fit(data_gen_train)
```

It is also possible to scale the data as it is being loaded from the disk, for instance, when training the model. To do so, the `Scaler` can be passed to the `DataGenerator` after its initialization.

```
>>> data_gen_val.set_scaler(scaler)
```

3.6. ModelContainer

This class defines an interface to standardize the behavior of machine learning models. It stores the architecture and the parameters of the model. It provides methods to train and evaluate the model, and to save and load its architecture and weights. It also allows the inspection of the output of its intermediate stages (i.e. layers).

The library also provides a container class to define Keras models, namely `KerasModelContainer`, that inherits from `ModeContainer`, and implements its functionality using this specific machine learning backend. Even though the library currently supports only Keras, it is easy to specialize the `ModelContainer` class to integrate other machine learning tools, such as PyTorch.

Each model has its own class that inherits from a specific `ModelContainer`, such as `KerasModelContainer`. The models currently implemented using Keras are: Multi–layer Perceptron (MLP), SB-CNN [20], SB-CNN-SED [19], A-CRNN [27], MST [28], SMel [29] and VGGish [23]. Other models are in preparation for inclusion in next releases of the library.

A model's container has to be initialized with some parameters.

```
>>> model_cont = SB_CNN(**model_params)
```

These parameters vary across models, among which the most important are: input shape, number of classes, and evaluation metrics. Specific parameters may include the number of hidden layers or the number of convolutional layers, among others.

The `ModelContainer` class has a method to train the model.

```
>>> model_cont.train((X_train, Y_train),
                     **train_params)
```

Training parameters can include, for example, number of epochs, learning rate and batch size. To train the model in batches, the `DataGenerator` object can be passed to the `train` method instead of the pre–loaded data.

```
>>> model_cont.train(data_gen_train, **train_params)
```

Performing model evaluation is also simple. For instance, the following code uses the test set for evaluating the model.

```
>>> data_gen_test = DataGenerator(
    dataset, features, train=False, folds=['test'])
>>> X_test, Y_test = data_gen_test.get_data()
>>> results = model_cont.evaluate((X_test, Y_test))
```

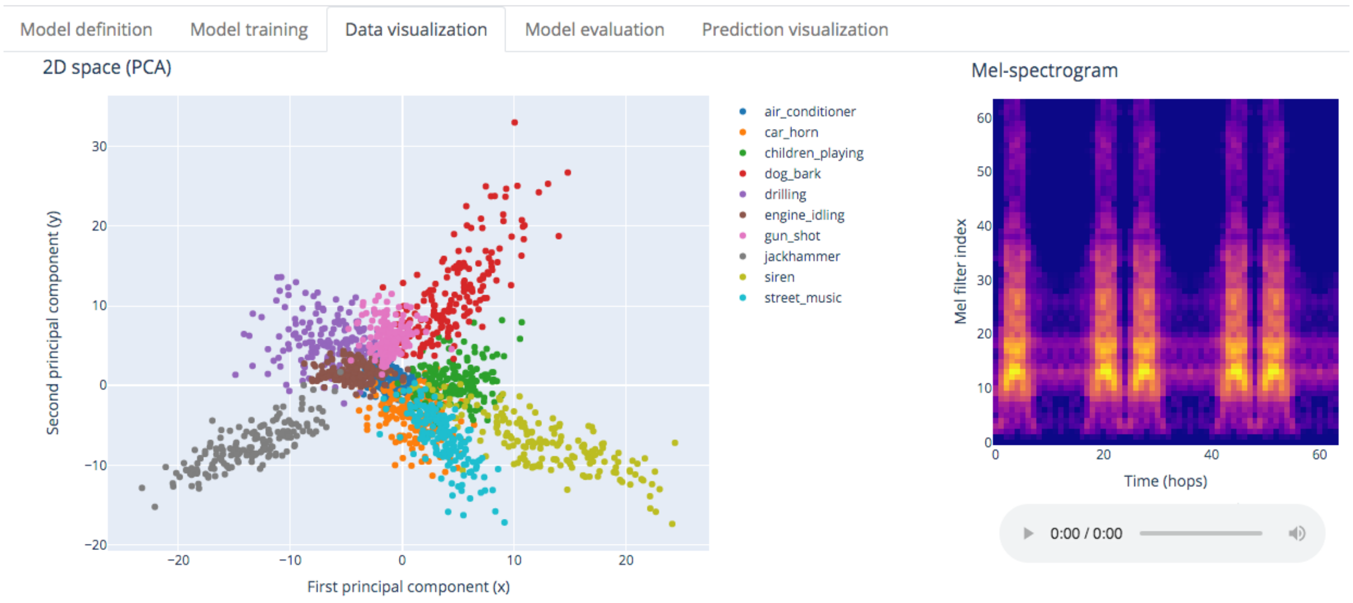

Figure 2: Screenshot of the web application for sound classification developed with `DCASE-models` as backend. In the *Data visualization* tab, the user can explore the training set by visualizing a 2D projection (principal component analysis, PCA) space of some model's intermediate output. It is also possible to inspect wrongly classified instances, visualize the feature representation, and listen to the audio files.

The results' format depends on which metrics are used. By default, the evaluation is performed using metrics available from the `sed_eval` library [21]. Therefore, the results are presented accordingly. Nevertheless, `DCASE-models` enables the use of others evaluating frameworks such as `psds_eval` [22], or the use of user–defined metrics in a straightforward way.

When building deep–learning models it is common practice to use fine–tuning and transfer learning techniques. In this way, one can reuse a network that was previously trained on another dataset or for another task, and adapt it to the problem at hand. This type of approach can also be carried out with the `ModelContainer`.

4. APPLICATION EXAMPLES

The Python package of the library includes a set of examples, organized into three different categories, which illustrate the usefulness of `DCASE-models` for carrying out research experiments or developing applications. These examples can also be used as templates to be adapted for implementing specific DCASE methods.

Firstly, some Python scripts are provided, that perform each step in the typical development pipeline of a DCASE task, i.e downloading a dataset, data augmentation, feature extraction, model training, fine–tuning, and model evaluation. See the documentation of the library for a tutorial that follows all these examples.

Secondly, several Jupyter Notebooks are also included whose aim is to replicate some of the experiments reported in the literature using `DCASE-models`, in particular those in [17, 19, 20, 27, 29]. We plan to add some other implementations of recent papers.

Finally, a web interface for sound classification is also included as a proof of concept of the potential of `DCASE-models` to build high–level applications for computational environmental audio analysis. It gives access to most of the library's functionalities through a graphical user interface. Besides, it provides visualization tools to explore the dataset and to inspect the errors made by the model. It is also possible to listen to the audio files of the dataset and to test the model on an audio file provided by the user. Figure 2 shows a screenshot of the web application. The HTML front–end is developed with the `dash` library.[5]

5. CONCLUSION

In this paper, the first release of `DCASE-models` is presented.[6] This open–source Python library provides a number of classes useful for rapid prototyping solutions for DCASE related problems, with a particular emphasis on deep–learning models. The library has a flat and light design that allows easy extension and integration with other existing tools. The design also provides an abstraction layer that seeks to mitigate the impact of changes in the backends used for implementing the machine learning models. We put considerable effort into the usability aspects of the library to encourage its adoption by the DCASE community. In particular, we hope it turns out helpful for those students and researchers new to the field. In this sense, we look forward to other researchers' feedback and contributions. Additionally, we believe that the library could simplify the process of releasing and maintaining the code of new models. This, in turn, could improve research reproducibility and simplify methods comparison. To this respect, the package includes a set of application examples, some of which replicate a number of experiments reported in the literature. Besides, a web interface for sound classification is provided as a proof of concept of the usefulness of the library for developing high–level applications for computational analysis of acoustic scenes and sound events. We foresee an interesting use case for the library as a tool to facilitate understanding and explainability of DCASE machine learning models.

[5] `https://plotly.com/dash/`
[6] `https://github.com/MTG/DCASE-models`

6. REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 56–67, 2015.

[2] B. McFee, J. W. Kim, M. Cartwright, J. Salamon, R. M. Bittner, and J. P. Bello, "Open-source practices for music signal processing research: Recommendations for transparent, sustainable, and reproducible audio research," *IEEE Signal Processing Magazine*, vol. 36, no. 1, pp. 128–137, Jan. 2019.

[3] P. Vandewalle, J. Kovacevic, and M. Vetterli, "Reproducible research in signal processing," *IEEE Signal Processing Magazine*, vol. 26, no. 3, pp. 37–47, 2009.

[4] C. Raffel, B. McFee, E. J. Humphrey, J. Salamon, O. Nieto, D. Liang, and D. P. W. Ellis, "mir_eval: A transparent implementation of common MIR metrics," in *Proc. of the 15th ISMIR*, Oct. 2014, pp. 367–372.

[5] J. Six, F. Bressan, and M. Leman, "A case for reproduciblity in MIR: replication of 'a highly robust audio fingerprinting system'," *Transactions of the International Society for Music Information Retrieval*, vol. 1, no. 1, pp. 56–67, 2018.

[6] M. Colom, B. Kerautret, N. Limare, P. Monasse, and J. Morel, "IPOL: A new journal for fully reproducible research; analysis of four years development," in *Proc. of the 7th International Conference on New Technologies, Mobility and Security (NTMS)*, Jul. 2015, pp. 1–5.

[7] E. Bjornson, "Reproducible research: Best practices and potential misuse [perspectives]," *IEEE Signal Processing Magazine*, vol. 36, no. 3, pp. 106–123, 2019.

[8] S. Leonelli and R. A. Ankeny, "Repertoires: How to Transform a Project into a Research Community," *BioScience*, vol. 65, no. 7, pp. 701–708, 05 2015.

[9] A. Mesaros, T. Heittola, E. Benetos, P. Foster, M. Lagrange, T. Virtanen, and M. D. Plumbley, "Detection and classification of acoustic scenes and events: Outcome of the DCASE 2016 challenge," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 26, no. 2, pp. 379–393, Feb. 2018.

[10] A. Mesaros, A. Diment, B. Elizalde, T. Heittola, E. Vincent, B. Raj, and T. Virtanen, "Sound event detection in the DCASE 2017 Challenge," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 27, no. 6, pp. 992–1006, Jun. 2019.

[11] K. J. Piczak, "ESC: Dataset for Environmental Sound Classification," in *Proc. of the 23rd ACM international conference on Multimedia*, Oct. 2015, pp. 1015–1018.

[12] J. Salamon, C. Jacoby, and J. P. Bello, "A dataset and taxonomy for urban sound research," in *Proc. of the 22st ACM International Conference on Multimedia*, Nov. 2014.

[13] A. Mesaros, T. Heittola, A. Diment, B. Elizalde, A. Shah, E. Vincent, B. Raj, and T. Virtanen, "DCASE 2017 challenge setup: Tasks, datasets and baseline system," in *Proc. DCASE Workshop*, Nov. 2017, pp. 85–92.

[14] A. Mesaros, T. Heittola, and T. Virtanen, "A multi-device dataset for urban acoustic scene classification," in *Proc. DCASE Workshop*, Nov. 2018, pp. 9–13.

[15] E. Fonseca, M. Plakal, F. Font, D. P. W. Ellis, X. Favory, J. Pons, and X. Serra, "General-purpose tagging of freesound audio with audioset labels: Task description, dataset, and baseline," in *Proc. DCASE Workshop*, Nov. 2018.

[16] M. Cartwright, A. E. M. Mendez, J. Cramer, V. Lostanlen, G. Dove, H.-H. Wu, J. Salamon, O. Nov, and J. Bello, "SONYC urban sound tagging (SONYC-UST): A multilabel dataset from an urban acoustic sensor network," in *Proc. DCASE Workshop*, Oct. 2019, pp. 35–39.

[17] P. Zinemanas, P. Cancela, and M. Rocamora, "MAVD: a dataset for sound event detection in urban environments," in *Proc. DCASE Workshop*, Oct. 2019.

[18] T. Heittola, A. Mesaros, and T. Virtanen, "Acoustic scene classification in DCASE 2020 challenge: generalization across devices and low complexity solutions," in *Proc. DCASE Workshop*, 2020, submitted.

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

[20] J. Salamon and J. P. Bello, "Deep convolutional neural networks and data augmentation for environmental sound classification," *IEEE Signal Processing Letters*, vol. 24, pp. 279–283, Mar. 2017.

[21] A. Mesaros, T. Heittola, and T. Virtanen, "Metrics for polyphonic sound event detection," *Applied Sciences*, vol. 6, no. 162, 2016.

[22] C. Bilen, G. Ferroni, F. Tuveri, J. Azcarreta, and S. Krstulovic, "A framework for the robust evaluation of sound event detection," in *Proc. IEEE ICASSP*, May. 2020.

[23] S. Hershey, S. Chaudhuri, D. P. W. Ellis, J. F. Gemmeke, A. Jansen, C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold, M. Slaney, R. Weiss, and K. Wilson, "CNN architectures for large-scale audio classification," in *Proc. IEEE ICASSP*, Mar. 2017. [Online]. Available: https://arxiv.org/abs/1609.09430

[24] I. H. Witten and E. Frank, "Credibility: Evaluating what's been learned," in *Data Mining. Practical Machine Learning Tools and Techniques*, 2nd ed. Elsevier, 2005, ch. 5, pp. 143–185.

[25] R. M. Bittner, E. J. Humphrey, and J. P. Bello, "pysox: Leveraging the audio signal processing power of sox in python," in *Proc. of the 17th ISMIR*, Aug. 2016.

[26] J. Cramer, H.-H. Wu, J. Salamon, and J. P. Bello., "Look, listen and learn more: Design choices for deep audio embeddings," in *Proc. IEEE ICASSP*, May 2019, pp. 3852–3856.

[27] S. Adavanne, P. Pertilä, and T. Virtanen, "Sound event detection using spatial features and convolutional recurrent neural network," in *Proc. IEEE ICASSP*, Mar. 2017, pp. 771–775.

[28] T. M. S. Tax, J. L. D. Antich, H. Purwins, and L. Maaløe, "Utilizing domain knowledge in end-to-end audio processing," in *Proc. of the 31st Conference on Neural Information Processing Systems (NIPS)*, Dec. 2017.

[29] P. Zinemanas, P. Cancela, and M. Rocamora, "End–to–end convolutional neural networks for sound event detection in urban environments," in *Proc. of the 24th Conference of Open Innovations Association FRUCT*, Apr. 2019, pp. 533–539.