Acoustic bird detection using deep convolutional neural networks

Neural Networks 2018/2019 - Sapienza University

Leonardo Sarra

January 18, 2019

1 Introduction

Recently, with the rapid decline in global wildlife populations due to environmental pollution, there has been a progressive effort over the years for monitoring vocalizing species as valid indicators of biodiversity.
Monitoring avian population in their habitats is one of such efforts since birds are good ecological indicators of environmental changes and can be used by the researchers to analyze habitat change, migration pattern, pollution, and disease outbreaks.
We call bird audio detection system a system that has as its primary goal to detect if there a bird in the environment given a short audio recording.

It is no surprise that during the DCASE 2018 workshop challenge, the participants were asked to resolve a bird audio detection problem.
My goal was to replicate the results by using multiple approaches that were developed by analyzing the many submissions to the challenge.

2 Dataset

The datasets used to solve the detection task contains 40.000 10-second-long WAV audio files (44.1 kHz mono PCM), that are manually labeled with a 0 or 1 to indicate respectively the absence or the presence of any birds within that 10-second audio clip.
In particular, the datasets used for this task were:

- Field recordings, worldwide ("freefield1010") - a collection of 7,690 excerpts from field recordings around the world, gathered by the FreeSound

project, and then standardized for research.

- Crowdsourced dataset, UK ("warblrb10k") - 8,000 smartphone audio recordings from around the UK, crowdsourced by users of Warblr the bird recognition app. The audio covers a wide distribution of UK locations and environments and includes weather noise, traffic noise, human speech, and even human bird imitations.

- Remote monitoring flight calls, USA ("BirdVox-DCASE-20k") - 20,000 audio clips collected from remote monitoring units placed near Ithaca, NY, USA during the autumn of 2015, by the BirdVox project.

Of these datasets only "warblrb10k" comes with a default evaluation dataset (2000 audio files), for the other dataset I had to extract at random 2000 audio files which were removed by the training set and became part of my evaluation dataset.
All those audio files come pretty in handy when training the neural networks and so improving the performance of our implementation, this is due to the fact that those files were taken in different environments, so enabling us to generalize the detection problem.

3 Data preparation

Before training/testing, all the audio files of the dataset are used to generate a graphical representation (one for each) which will be later used as the input of the neural networks.
The implementation that was developed supports the following graph type:

- Spectrogram: a visual representation of the spectrum of frequencies of a signal as it varies with time.

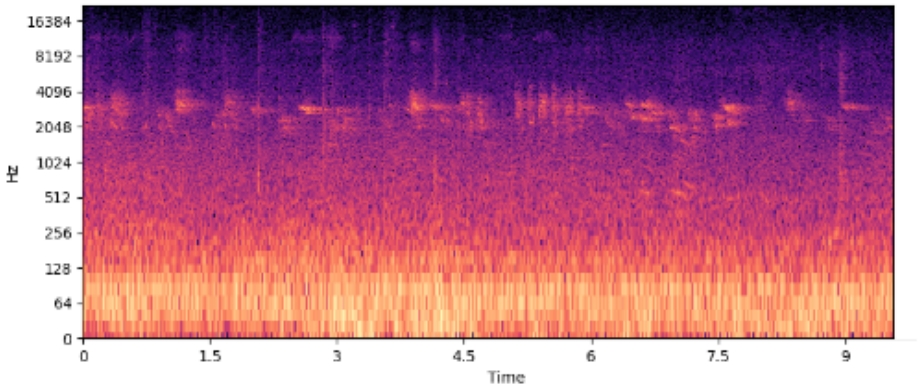
- Mel-spectrogram: A spectrogram which is mel-scaled, this representation will be used to try to capture frequencies in the range 20Hz-20kHz.

- Mel-Filterbanks, representation of the multiple components of the input mel-spectrogram after band-pass filters are applied.

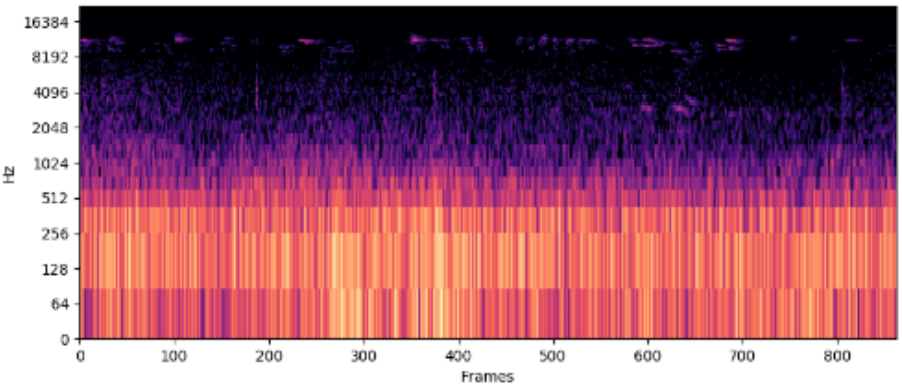
- Log-scaled mel-spectrogram energy features, a representation of the energy features computed starting from a mel-spectrogram.

- Mel-frequency cepstrum (MFCC), a mel-scaled representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum.

The graphs, after the removal of high/low frequencies, the normalization and conversion (to dB units) are saved as 1000x400 images.
Each of those graphs is very specific and can be used to detect birds with very different results which will later be exposed.



Spectrogram



Filterbank

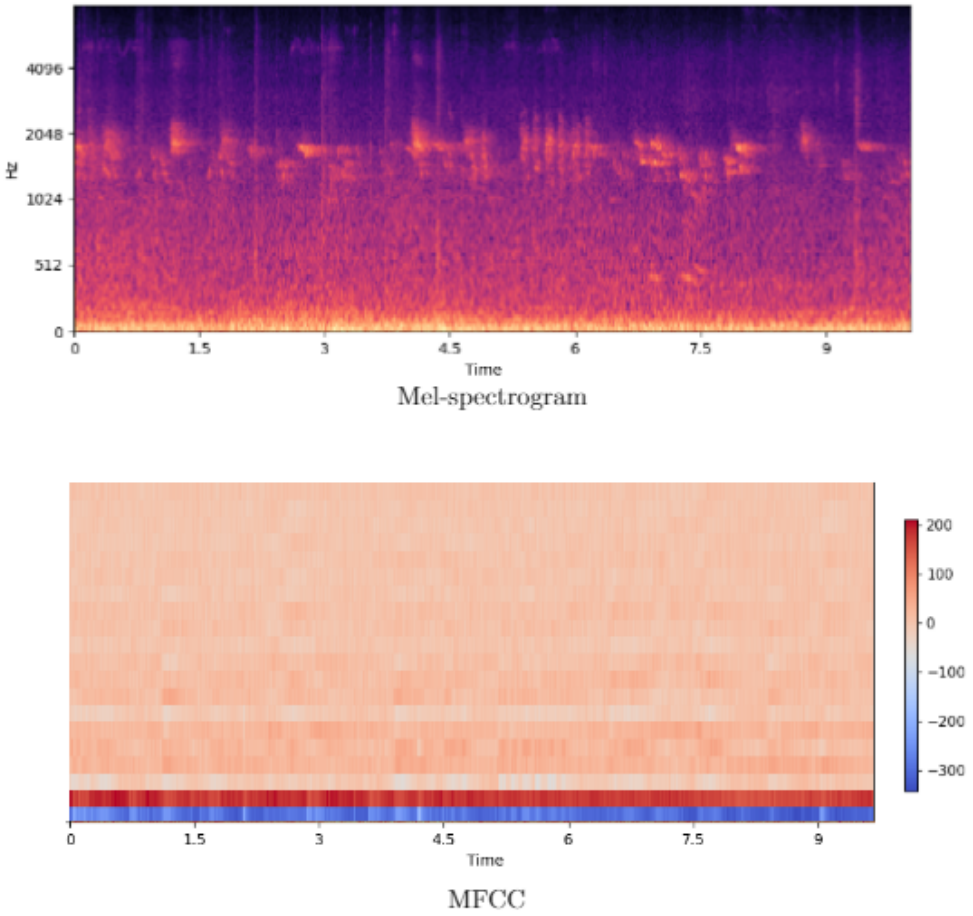


Figure 1: Examples of the graphs that were used

4 Data augmentation

While the dataset is pretty big there is still a chance to incur in overfitting. To increase model performance and improve generalization to new recording conditions and habitats, various data augmentation techniques are applied in both time and frequency domain in order to artificially create new training data from existing one.
The augmentation supported are the following:

- Additive noise, a fixed amount of noise is added to the audio file.
- Random noise, a random amount of noise is added to the audio file.
- Pitch-shift the waveform by a fixed amount of half-steps.

- Time stretch of the input audio to a specific length, padding is added when the targeted length is less than 10-sec.

For our training environment, only a small percentage of the audio data got augmented (20% of the training dataset).

5 The neural networks

Neural networks are a simple system made of multiple interconnected processing elements which process information by their dynamic state response to external inputs.
Neural networks are typically organized in layers, made up of a number of interconnected nodes containing an activation function.
The input is given using the input layer, which communicates to one or more hidden layers where the actual processing is done via a system of weighted connections. The hidden layers then link to an output layer where the answer is given as output.
Neural networks are very useful for finding complex or numerous patterns that are hard for a human to extract and teach the machine to recognize.
In particular one of the problems in which the neural networks shine is the images analysis.

For my task, I opted to consider the problem as an image detection one which can be solved using the previously mentioned graphical representations (resized to 224x224, 3 channels RGB) that I can create starting from the audio files.
The neural network that I crafted to accomplish this goal is named LeoNet and is VGG-style network which is compact, with few layers and trainable parameters.
LeoNet, more specifically, is a CNN, a feed-forward neural network trained by back propagation, which has 4 stacked convolutional layers and has a kernel size of 3x3. It has dropout, max-pooling, dense layers, ReLU and ELU activations, and SGD with momentum as its loss function.
It attaches ReLU activations at the first convolutional layer and at the dense layer while the other convolutional layers use ELU, the output layer, instead, uses a softmax activation.

I also crafted another CNN, this one named LeoNetV2, which is very similar to LeoNet. LeoNetV2 is way denser in the last layer, has only a ReLU

activation at the start (after that ELU is used, except for the output layer which, like LeoNet, uses a softmax activation) and no batch normalization layers (except in the layer with ReLU).

Table 1: LeoNet architecture

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 224, 224, 32)	896
batch_normalization_1 (Batch)	(None, 224, 224, 32)	128
activation_1 (RELU)	(None, 224, 224, 32)	0
conv2d_2 (Conv2D)	(None, 224, 224, 32)	9248
batch_normalization_2 (Batch)	(None, 224, 224, 32)	128
elu_1 (ELU)	(None, 224, 224, 32)	0
max_pooling2d_1 (MaxPooling2)	(None, 112, 112, 32)	0
dropout_1 (Dropout)	(None, 112, 112, 32)	0
conv2d_3 (Conv2D)	(None, 112, 112, 32)	9248
batch_normalization_3 (Batch)	(None, 112, 112, 32)	128
elu_2 (ELU)	(None, 112, 112, 32)	0
max_pooling2d_2 (MaxPooling2)	(None, 56, 56, 32)	0
dropout_2 (Dropout)	(None, 56, 56, 32)	0
conv2d_4 (Conv2D)	(None, 56, 56, 32)	9248
batch_normalization_4 (Batch)	(None, 56, 56, 32)	128
elu_3 (ELU)	(None, 56, 56, 32)	0
max_pooling2d_3 (MaxPooling2)	(None, 28, 28, 32)	0
dropout_3 (Dropout)	(None, 28, 28, 32)	0
flatten_1 (Flatten)	(None, 25088)	0
dense_1 (Dense)	(None, 128)	3211392
activation_2 (RELU)	(None, 128)	0
dropout_4 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 2)	258
Output (softmax)	(None, 2)	0

Total params: 3,240,802
Trainable params: 3,240,546
Non-trainable params: 256

Table 2: LeoNetV2 architecture

Layer (type)	Output Shape	Param #
Input (Conv2D)	(None, 222, 222, 32)	896
batch_normalization_1 (Batch)	(None, 222, 222, 32)	128
activation_1 (RELU)	(None, 222, 222, 32)	0
conv2d_1 (Conv2D)	(None, 220, 220, 32)	9248
activation_2 (ELU)	(None, 220, 220, 32)	0
max_pooling2d_1 (MaxPooling2)	(None, 110, 110, 32)	0
dropout_1 (Dropout)	(None, 110, 110, 32)	0
conv2d_2 (Conv2D)	(None, 108, 108, 32)	9248
activation_3 (ELU)	(None, 108, 108, 32)	0
max_pooling2d_1 (MaxPooling2)	(None, 54, 54, 32)	0
dropout_2 (Dropout)	(None, 54, 54, 32)	0
conv2d_3 (Conv2D)	(None, 52, 52, 32)	9248
activation_4 (ELU)	(None, 52, 52, 32)	0
max_pooling2d_3 (MaxPooling2)	(None, 26, 26, 32)	0
dropout_3 (Dropout)	(None, 26, 26, 32)	0
flatten_1 (Flatten)	(None, 21632)	0
dense_1 (Dense)	(None, 256)	5538048
activation_5 (ELU)	(None, 256)	0
dropout_4 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 2)	514
Output (softmax)	(None, 2)	0

Total params: 5,567,330
Trainable params: 5,567,266
Non-trainable params: 64

6 Training setup

For the training setup I start by augmenting the existing training dataset: 10% of the files are subjected to a random noise and pitch shift (shift by a major third) augmentation, another 10% is augmented by adding a fixed noise to the data array and by time shifting the 10 sec file into 8.5 sec file (padding is added to reach 10 sec duration).
After the augmentation phase I produce every graph type for every file of the training dataset.
At this point I have all the images to train LeoNet/LeoNetV2 from the ground up.
As recommended by the challenge organizers, two development sets are used for training and the remaining one for testing the performance.

Fold	Training dataset	Testing dataset
1	BirdVox20k, ff1010bird	warblrb10k
2	warblrb10k, ff1010bird	BirdVox20k
3	BirdVox20k warblrb10k	ff1010bird

Note that during the learning phase the training dataset is split in an 85%-15% training-validation configuration, where the first one is used for training the neural networks while the second one comes in handy when evaluating the improvements of the metrics during the training.

The neural networks are trained for 30 epochs, wherein each epoch the whole training dataset is passed in multiple batches of 20 entries.
The learning process produces two models for each training run, one model is the one produced after the last epoch while the other is a model computed using the weights that provided the best accuracy on the validation set during the training.
On top of that the learning process supports early stopping to prevent overfitting, this means that the learning will be interrupted if the validation loss metric becomes stale for a fixed number of epochs.

7 Testing setup

After the training phase we have all the trained models which cover both Leonet and LeonetV2 on every graph type on all the stratified 3-way cross-validation configurations (for a total of 30 trained models), the only things that are missing are the graphs of the testing dataset which are produced like in the training phase, except for the augmentation which isn't applied because there is no need to artificially create new instances.
At this point, we can test the implementation and so evaluating the performance of one neural network against a specific type of graphs.
The implementation developed supports a multiple-learners approach, this means that multiple models can be used to analyze a particular audio file, for example I can use a LeoNetV2 model trained on mel-spectrograms, together with a LeoNetV2 model trained on spectrograms and a LeoNet model trained on filterbanks, and take as the prediction for a specific file the result with the highest confidence level.
This approach is supposed to improve the performance of our implementation because different graphs may highlight difference features (with a different confidence level) that help us detect birds.

8 Results

The results were computed by testing every possible model on every graph type on all the stratified 3-way cross-validation configurations. As suggested by the DCASE organizer I used the AUC score to evaluate the final results. The AUC score provides an aggregate measure of performance across all possible classification thresholds.
One way of interpreting AUC is as the probability that the model ranks a random positive example more highly than a random negative example. It is important not to exclusively use only the accuracy metric because, in the case of imbalanced datasets, it can provide a false sense of correctness.

Before I started evaluating the performance I had some idea about which of the graphs would prove more useful in classifying birds.
I thought that spectrograms were a bad idea because of their "transparency" property, which comes from the fact that a particular observed frequency in a spectrogram cannot be assumed to belong to a single sound as the magnitude of that frequency could have been produced by any number of accumulated

sounds or even by the complex interactions between sound waves such as phase cancellation. This makes it difficult to separate simultaneous sounds in spectrogram representations.
I also had low expectations for the MFCC graph, which may be useful for speech recognition on small frames but considering that my implementation wasn't built in a way to analyze different frames from the same file I expected this approach to be ineffective.
Using MFCC(s) to generate a dictionary of MFCC-based words which can later be used to detect birds (and eventually their family) would have been by far a better alternative.
The only method for which I had some expectations were the one that used mel-spectrograms because Mel frequency spacing approximates the mapping of frequencies to patches of nerves in the cochlea, and thus the relative importance of different sounds to humans.
Thus, binning a spectrum into approximately mel frequency spacing widths lets you use spectral information in about the same way as human hearing, other than that I wasn't sure if not analyzing frame by frame would be a problem, but from the experience of the others participants of the DCASE challenge I knew that this should not be the case.

The results for LeoNet and LeoNetV2 were as follow:

Table 3: Results for LeoNetV2 architecture

Fold/Graph type	Melspectrogram	Mel-energy	Spectrogram	Filterbank	MFCC
1	83,6%	80,7%	80,0%	70,9%	58,6%
2	59,0%	66,0%	65,1%	68,0%	60,0%
3	83,3%	82,0%	79,4%	72,5%	68,0%
Average AUC	75,3%	76,2%	74,8%	70,4%	62,2%

Table 4: Results for LeoNet architecture

Fold/Graph type	Melspectrogram	Mel-energy	Spectrogram	Filterbank	MFCC
1	79,4%	81,2%	73,5%	75,5%	50,0%
2	66,0%	63,1%	60,4%	65,0%	50,0%
3	80,8%	78,4%	75,8%	73,0%	49,2%
Average AUC	75,4%	74,2%	69,9%	71,1%	49,7%

As expected results for MFCC were disappointing for the reason already mentioned. Spectrogram, instead, worked surprisingly well but not as good as when the mel-spectrogram and mel-energy features representations are used which are by far the ones that provide the best AUC score.
The model which used filterbank also provided discrete results sometimes even performing better than the other representations. From the results, we can see that overall LeoNetV2 performs better than its less dense counterpart.
The most problematic fold is definitely the one which is tested against the BirdVox20k dataset (fold n.2), note that this drop in performance also afflicted other participants of the DCASE challenge and had to be expected because that particular dataset contains community made audio files which comes from a very specific environments that cannot be generalized using the data provided by the other datasets.
Obviously, if I train a model using the BirdVox20k training set I have no problem in detecting birds on the BirdVox20k testing set.

The augmentations played an important role in improving the performance, providing a boost of performance of 4-5%. The augmentations that turned out more effective where the random noise and the frequency shift augmentation.
As said before, the implementation that was developed supports multiple models when making predictions on the testing dataset, in particular, the model which provide the higher confidence for a particular entry of the dataset will be used to evaluate it, so I decided to use the most promising models to check if I could boost the performance a bit more by using them together.
I opted to use the LeoNet melspectrogram model with the LeoNetV2 mel-energy features model and later on to use the LeoNetV2 mel-energy model with the LeoNetV2 spectrogram model, the results were as follow:

Fold/trained models	LeoNet mel & LeoNetV2 mel-en
1	80,4%
2	66,4%
3	83,0%
Average AUC	76,6%

Table 5: Results for multiple-learners approach 1/2

Fold/trained models	LeoNetV2 mel-en & LeoNetV2 spectrogram
1	81,0%
2	66,9%
3	82,7%
Average AUC	76,8%

Table 6: Results for multiple-learners approach 2/2

The results are promising and the multiple-learners approach ended up providing results that are better than both of the ones achieved by the single models, nevertheless, the performance improvements are limited.
Note that even when mel-spectrogram and mel-energy are usually the best two representations the best result is achieved by considering mel-energy and spectrograms, this is probably due to the fact that the two representations were able to focus on different features whose combination provides better results on our task.

While those results are pretty good considering that each model was trained for around 30 epochs, there is still something room for improvements.
First of all, we could start by introducing a frame by frame analysis for each audio file in order to improve the performance when using MFCC to generate a dictionary of words.
Other improvements may include the support for more augmentations, like one that shuffles the frames of an audio file.
Incrementing the number of epochs may also provide a boost in performance but right now using the LeoNetV2/LeoNet neural networks architecture provides the following result.

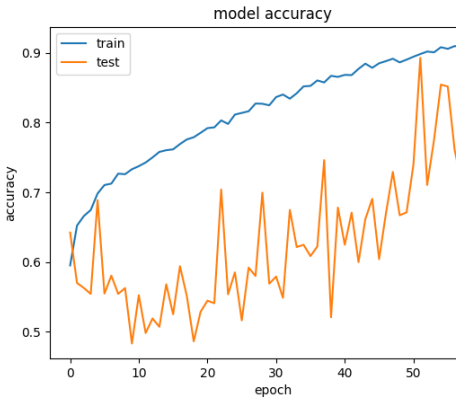


Figure 2: Accuracy history for LeoNetV2 in fold 1

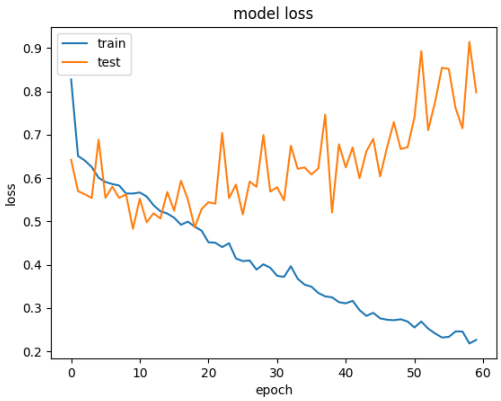


Figure 3: Loss history for LeoNetV2 in fold 1

As shown the architecture incurs in overfitting after some epochs, this is the reason why I decided to limit introduce early stopping support in the implementation and to limit the maximum epochs to 30. In order to prevent overfitting, a solution may be to tinker with the architecture and its learning rate.

There is also room for improvement for the multiple-learners approach, for example, instead of taking the model with the higher confidence, we could decide which model to use by analyzing the properties of the audio such that I will choose the most significant graph to classify each file.

9 How to use

The project is available in my GitHub repo[1] and requires Tensorflow, Python 3.6.7, Keras, librosa (for audio processing) and other dependencies (primarily for plotting), that are written in "requirements.txt", to be installed.
It's is important to place the testing and training set in the right location.
The audio dataset has to be placed inside
data/audio/{type of dataset}/{name of dataset} folder located inside the project folder.
Inside this folder, there must be a labels.csv file containing at each row an entry "{file_name},{dataset_name},{has_bird}", those entries have to cover every single file of the dataset they are referring to.

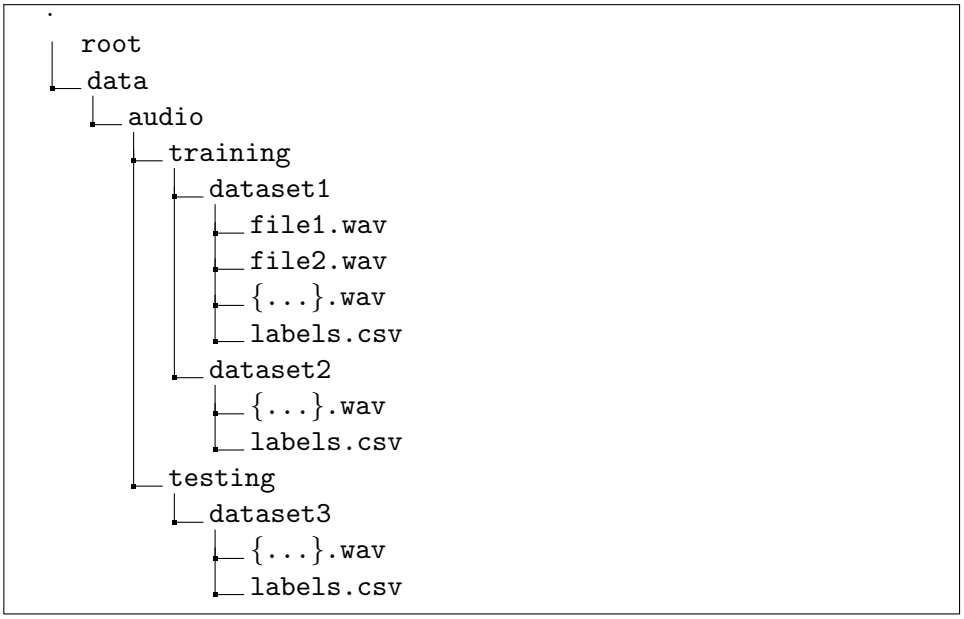


Figure 4: Example of a correct organization of the datasets

Below is an explanation of how the scripts are supposed to be used in order to correctly perform the parsing, learning and evaluation phase.

[1] https://github.com/LithiumSR/bird_detection_nn

9.1 Generating the graphs

After the dataset has been correctly placed we can start producing the graphs for the training and testing dataset audio files using "GraphGenerator.py", we can also choose the desired augmentations (recommended for the training datasets).

```
usage: GraphGenerator.py [graph_type] [folder_type] [folders] [additive_noise]
                         [random_noise] [time_stretch_rate] [skip_probability]

Customization options for the graph generator

positional arguments:
  graph_type          Set type of graph
                      (i.e: mfcc/melspectrogram/melspectrogram-energy/spectrogram).
  folder_type         Set folder type (e.g: training or testing).
  folders             Folders that will be parsed (e.g: "folder1,folder2").
  additive_noise      Additive noise (ignored if 0).
  random_noise        Random noise (ignored if 0).
  time_stretch_rate   Time stretch (ignored if time_stretch_rate= audio time).
  skip_probability    Probability of skipping an augmentation regarding a file.
```

Note that multiple executions are required because each run can cover only a graph type and one folder type (training/testing).

9.2 Training the neural networks

At the end of the execution of the graph generator the data folder will contains also the images generated in the 1000x400 format and we are in the position of training our neural network models using the "NetworkLearner.py" script.
At the end the AUC score achieved by the provided configuration will be printed on screen.

```
usage: NetworkLearner.py [graph_type] [neural_network] [folders] [batch_size]
                         [validation_percentage] [epochs] [output]
                         [early_stopping]

Customization options for the network learner

positional arguments:
  graph_type            Set type of graph (i.e: mfcc/melspectrogram/
                        melspectrogram-energy/spectrogram).
  neural_network        Set the network you want to train (i.e: LeoNet/LeoNetV2).
  folders               Set of folders of the training set from which
                        the graphs will be taken.
  batch_size            Batch size of the files used to train the model.
  validation_percentage Percentage of file used for validation.
  epochs                Number of epochs.
  output                Output filename.
  early_stopping        Use early stopping when training the model.
                        (disable by default if validation percentage is 0).
```

After the execution of this script, we will end up with two trained models "best_{output name}.h5" and "{output name}.h5" where the first is the checkpoint model which provided the best validation accuracy during the training while the second one is the model obtained after the last epoch of training.
The script will also create two new files "acc_{output name}.png" and "loss_{output name}.png" which are images representing the evolution of the loss and the accuracy metric during the training.
As for the graph generation phase, more runs are required, each for every fold of the folders and for every type of graphs and neural network.

9.3 Evaluating networks performance

Now that the models are trained, the only thing that is missing in our workflow is the evaluation of the models' performance. To do this the "NetworkEvaluation.py" script is used.

```
usage: NetworkEvaluation.py [models] [folders] [batch_size]

Customization options for the network evaluation script

positional arguments:
  models      Set models that will be used to make the predictions
              (e.g: "type_graph1,model1.h5 type_graph2,model2.h5").
  folders     Set of folders that will be used as the source of the graphs
              (e.g: "folders1, folders2).
  batch_size  Batch size of the files used to evaluate the model.
```

The script can support multiple models as input trained on different graphs using different network architectures, in this case, the one that will be used as output is the one which offers the higher confidence for a particular file.

References

[1] Automatic acoustic detection of birds through deep learning: the first Bird Audio Detection challenge, D. Stowell, Y. Stylianou, M. Wood, H. Pamua, and H. Glotin
https://arxiv.org/pdf/1807.05812

[2] Acoustic bird detection with deep convolutional neural networks, Lasseck Mario
http://dcase.community/documents/challenge2018/technical_reports/DCASE2018_Lasseck_76.pdf

[3] A Bag of MFCC-based words for bird identification, Julien Ricard and Herve Glotin
http://ceur-ws.org/Vol-1609/16090544.pdf

[4] Studies on Bird Vocalization Detection and Classification of Species, SeppoFagerlund
https://pdfs.semanticscholar.org/87d6/39403cf946281f55e425f77298fc1aed163a.pdf