Implementation of SED with Depthwise Separable and Dilated Convolutions

Neural Networks Sapienza 2020

Riccardo Prinzivalle

March 2021

1 Introduction

This project is a study and implementation of a polyphonic sound event detection extracted from [1]. It is also based on the baseline reference of [1], which is [2]. These two works represent the main source of this project. Here there will be presented both a replication of the paper approach together with a monophonic sound event detection, since to obtain the original dataset took some time, the author thought to start working with another dataset and then move the work to the original dataset when it would have been available. Section 2 and 3 are organized as follows: first an analysis of the dataset is performed to better understand it, then it is explained how the feature have been extracted and finally it is proposed a model to solve the problem. Section 4 regroups the results for both datasets, then it is explained a brief digression on how to train a neural network model on an AMD GPU on section 5 since the author's setup has only an AMD GPU. The work is ended by conclusions of section 6.

2 Monophonic SED

Monophonic Sound Event Detection consist of predicting a single label for an audio recording: the record will likely contain some noise but it generally contains a single and remarkable sound to be identified. In this case, it is used the *UrbanSound8K* dataset [3].

2.1 Data analysis

The dataset is composed by 8732 labelled small sound recordings (less than 4 seconds) from 10 classes: *air_conditioner*, *car_horn*, *children_playing*, *dog_bark*, *drilling*, *engine_idling*, *gun_shot*, *jackhammer*, *siren*, and *street_music*. The classes are balanced except for some, it can be seen in table 1. Only 3 out of 10 classes have less than 1000 elements, so there can be some problems predicting these classes.

Label	number of elements
air_conditioner	1000
car_horn	429
children_playing	1000
dog_bark	1000
drilling	1000
engine_idling	1000
gun_shot	374
jackhammer	1000
siren	929
street_music	1000

Table 1: Monophonic dataset label distribution

Moreover, the recordings have different properties since they come from `www.freesound.org` and are taken as they are. The first difference is in the audio lengths visible in figure 1: the majority of audio have a duration of about 3.5/4 seconds, but there exists also smaller recordings which are in a tiny number.

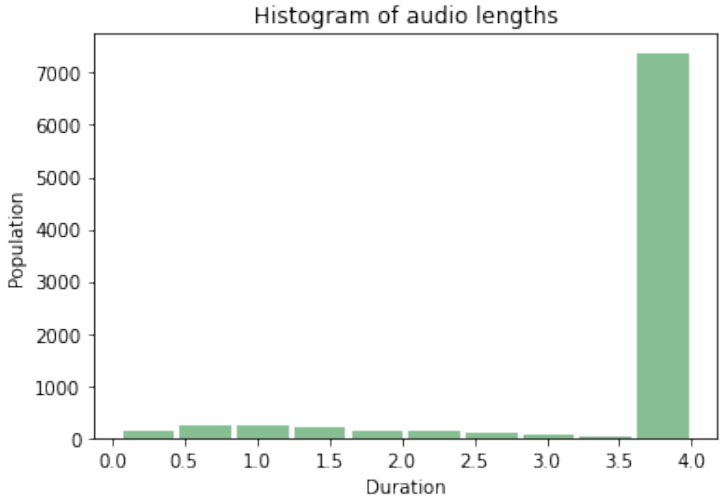

Figure 1: Audio duration distribution

The main differences are in bit depth, from 4 to 32 bit, the majority with 16 bit; and in the sample rate, from 8 KHz to 192 KHz, with the majority with 44.1 KHz. This may be a concern since some audio have a poor quality which can translate in poorer feature w.r.t. the other tracks. All these differences will be equalized during the feature extraction phase.

2.2 Feature Extraction

This phase adopts librosa [4], which is a sound processing library for python. Its use helps to deal with different audio characteristics since by default librosa converts audio to 22 KHz sampling rate and 16 bit depth. Since the majority of audio recordings are at 44.1 KHz, it may seem that down sampling may reduce audio quality, but if we visualize the sound with a spectrogram, it will be clear that most of frequency content is distributed well below the 11 KHz (which is the maximum frequency a 22 KHz sampling rate can process), so in this case it reduces the dimension of the data without losing much information. For what concerns the bit depth, the majority of recordings are already at 16 bit, so it does not change much the data. Audio file are loaded and transformed into array series by *load* function, which is also responsible of audio conversion and standardization.
The reference paper [1] uses Mel Frequency Cepstral Coefficients (MFCC) to extract features from the array sound data. MFCCs are a way of measuring the rate of information change in spectral bands and storing it in coefficients; moreover, the rate of change is modeled in a non linear way since the Mel band is logarithmic and the adoption of this band is able to capture the rate of change in a similar way to what the human hear does [5].
The idea is to extract the MFCC with the basic settings and change just the number of feature extracted per frame to 40 to adapt it to reference paper [1]. The principal basic setting is using a window of 2048 bit for the Fast Fourier Transform inside the MFCC extractor, the other settings are of minor importance in this case. The correspondence between the window and a temporal interval is given by the following formula:

$$window\_interval = \frac{bit\_ftt}{sampling\_rate} \tag{1}$$

This means that if the sampling rate is 22 KHz and the number of bit is 2048, then the window interval is about 93 ms. The execution shows that 93 ms are too much for just 3 recordings which have a smaller duration. The result of feature extraction is a matrix of 40 features by a varying length depending on the length of the processed audio. Here a problem arises: a neural network may only process input of equal length, so the smaller recordings are padded with zeros to reach the dimension of the longer audio, which has a length of 174 frames. The data are ready to be feed as input of the neural network now.
A small technical digression: since python based libraries for sound processing are easier to use but slower, the extracted features are saved in pickle file to be easily loaded without reprocessing every time the dataset [6].

2.3 Model formulation

This work presents two different model architectures for monophonic SED, both derived from [1]: a baseline architecture, composed by convolutional and recurrent layers, and a proposed model, which is constituted by depth-wise separable

convolutions followed by dilated convolutions.
Each model accepts an input of the following form: the first dimension is time, so the 174 frames, then the feature dimension (40) and the channel dimension, used by the convolution, which is just one since no convolution has been applied yet.
The baseline model is composed by 3 convolutional layer and 1 recurrent layer. Each convolution is followed by batch normalization, max pooling and dropout. Each convolution uses 256 channel, 5x5 kernel, unitary stride and ReLU activation function. Padding is added to preserve the input dimension (*same* padding). Max pooling is performed with different kernel dimension in the layers: (1,5), (1,4) and (1,2) respectively, with same padding to maintain dimension: this strange arrangement of pooling is to obtain a unitary feature dimension at the end of the third convolutional layer and preserve unchanged the time dimension. Dropout is added to reduce overfitting and it has a rate of 0.25. Since feature dimension is reduced to a unit, the tensor is reshaped to a matrix composed by the number of frames as rows and the number of channel as columns. This tensor is the input of the recurrent layer, which is a Gated Recurrent Unit of 256 unit. The output of the GRU is the input of a dense layer with 10 outputs and softmax activation function. The role of the 3 components is the following: the convolutions acts as feature extractor from the MFCCs, the GRU are used to identify temporal pattern and the dense layer is used as classifier. The model is trained using adam as optimizer, with categorical cross-entropy as cost function and categorical accuracy as metric. Adam optimizer is used with standard parameters [7].
ADD MODEL IMAGE
The proposed approach substitutes the convolutional layers with depth-wise separable convolutions and the recurrent layer with dilated convolution. Parameters are equal to the previous model, the only difference is that dilated convolution output has 3 dimension plus the batch size, so a global average pooling is used to reduce the dimension to feed the dense layer. This is a modification of the proposed approach of [1] since the dilated convolution are designed to output a tensor which can be reshaped to a 2D output to maintain temporal frames, instead in this case it is necessary to just assign a single category to all the frames together since only one sound is contained in each audio input, so I decided to use global average pooling to connect dilated convolution and the dense layer. Initially I tried to apply a convolution with unitary kernel and (1,3) stride to reduce dimension, followed by a max pooling to connect to dense layer but it produced worse result and I adopted the global average pooling solution; in both cases of proposed approach performs very poorly compared to baseline model, as it will be seen in section 4. This is due to the fact that the proposed model was developed for polyphonic SED and it is adapted to this case, probably a better formulation is possible for this specific case.
ADD MODEL IMAGE

3 Polyphonic SED

3.1 Data analysis

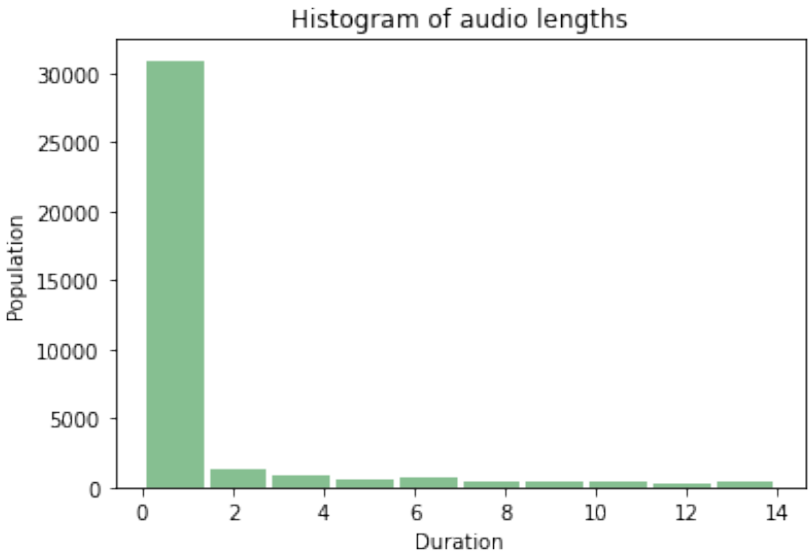

Figure 2: Decision trees assembly only confusion matrix

3.2 Feature Extraction

3.3 Model formulation

4 Experimental Results

4.1 Monophonic results

4.2 Polyphonic results

5 How to train NN on AMD GPU

6 Conclusions

References

[1] Konstantinos Drossos, Stylianos I. Mimilakis, Shayan Gharib, Yanxiong Li, and Tuomas Virtanen. Sound event detection with depthwise separable and dilated convolutions, 2020.

[2] Emre Cakir, Giambattista Parascandolo, Toni Heittola, Heikki Huttunen, and Tuomas Virtanen. Convolutional recurrent neural networks for polyphonic sound event detection. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(6):1291–1303, Jun 2017.

[3] J. Salamon, C. Jacoby, and J. P. Bello. A dataset and taxonomy for urban sound research. In *22nd ACM International Conference on Multimedia (ACM-MM'14)*, pages 1041–1044, Orlando, FL, USA, Nov. 2014.

[4] Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, volume 8, 2015.

[5] The dummy's guide to mfcc. `https://medium.com/prathena/the-dummys-guide-to-mfcc-aceab2450fd`. Accessed: 2021-02-25.

[6] Pickle python library. `https://docs.python.org/3.8/library/pickle.html`.

[7] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.