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. Since the original dataset needed some time to get the access, I thought to start working with another dataset, and then move to the original when its access would have been granted. Due to this, here there will be presented both a replication of the paper approach together with a monophonic sound event detection. 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 data have been preprocessed 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 my setup has only an AMD GPU. The work is ended by conclusions of section 6.

2 Monophonic SED

Monophonic Sound Event Detection consists of predicting a single class 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 exist also smaller recordings which are in a tiny number.

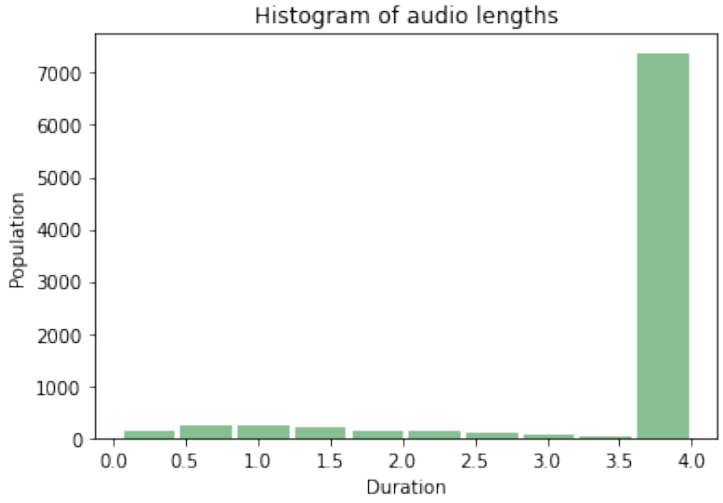


Figure 1: Monophonic audio duration distribution.

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 features w.r.t. the other tracks. All these differences will be equalized during the data preprocessing phase.

2.2 Data pre-processing

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 for audio conversion and standardization.
The reference paper [1] uses Mel Frequency Cepstral Coefficients (MFCC) to transform the audio array in something the network can learn from. 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 hearing does [5]. The idea is to extract the MFCC with the basic settings and change just the number of features 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 data pre-processing is a matrix of 40 features by a varying length depending on the time length of the processed audio. Here a problem arises: a neural network can only process inputs 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].
Processed data are labeled with a categorical encoding using *keras utils* and *sklearn preprocessing* in automatic way.

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.

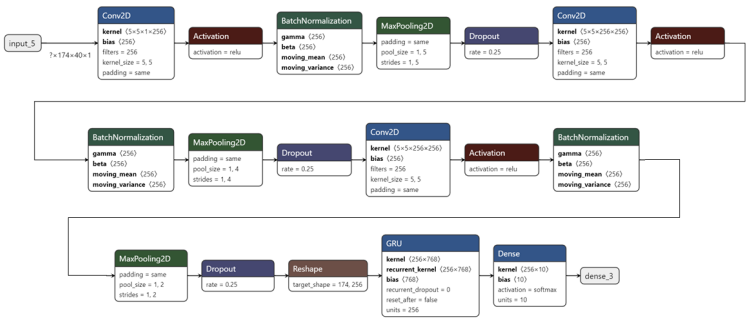


Figure 2: Baseline model for monophonic SED.

The baseline model is composed by 3 convolutional layers 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 dimensions 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 units. 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 act 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]. A representation of baseline model can be found in fig. 2.
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 dimensions plus the batch size, so a global average pooling is used to reduce the dimensions to feed the dense layer. This is a modification of the proposed approach of [1] since the dilated convolutions 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 it 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. Representation of proposed model architecture is visible in fig. 3.

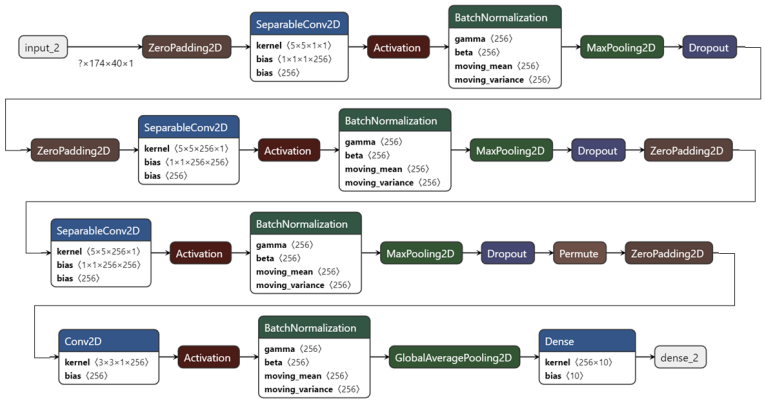


Figure 3: Proposed model for monophonic SED.

3 Polyphonic SED

Polyphonic SED, instead, is more complicate: the idea is to have recordings with multiple overlapping sounds and to detect correctly the category of each single sound and the instants when the sound starts and ends (the maximum polyphony is 5 in this case). The dataset used is *TUT-SED Synthetic 2016* [2], which is a synthetic dataset, while the section 2 dataset is composed by recordings in real spaces not modified in any way. Details on how the dataset has been

created can be found at `https://webpages.tuni.fi/arg/paper/taslp2017-crnn-sed/tut-sed-synthetic-2016`.

3.1 Data analysis

The dataset is composed by 100 long sound recordings (about some minutes) containing a mixture of different sound events. Each mixture contains multiple labels specified by onset and offset of each single sound event. Dataset contains a total of 36326 events distributed in 16 classes, the distribution can be seen in table 2.

Label	number of elements	Label	number of elements
footsteps	8302	motorcycle	824
horsewalk	6603	thunder	774
bird_singing	6079	glass_smash	693
baby_crying	3342	crowd_cheering	613
dog_barking	3145	alarms_and_sirens	571
gun_shot	1581	mixer	547
crowd_applause	1400	rain	511
cat_meowing	885	bus	456

Table 2: polyphonic dataset label distribution.

From the table, it is easily visible that classes are not distributed equally, moreover they have different duration, for example *rain* is a class with rather long duration while *horsewalk* contains very small duration events.

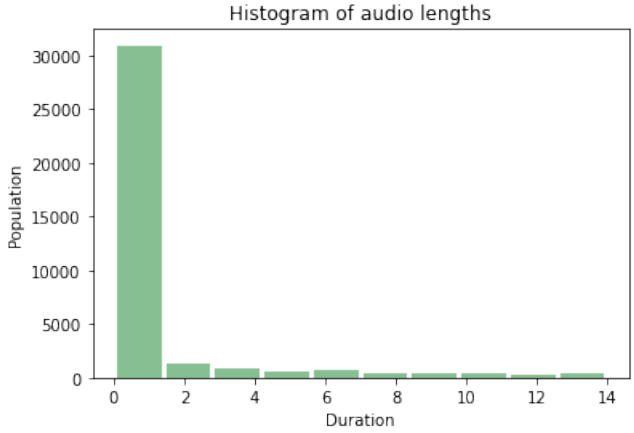


Figure 4: Polyphonic audio duration distribution.

As a rule of thumb of this dataset, one can enounce that the longer the duration the smaller the number of appearances, and vice versa. An histogram of audio lengths distribution can be found in figure 4; this confirms the rule of thumb.
Since the dataset is synthetic, audio properties are unique for all the mixtures: single channel, 32 bit depth and 44.1 KHz sampling rate.

3.2 Data pre-processing

This case will use the same technique of monophonic SED in section 2.2: the audio is loaded and transformed to 16 bit depth and 22 KHz sampling rate, this is done to reduce the dimensionality of processed data and speed up computation at cost of small losses of audio quality.
MFCCs are extracted with different parameters in this case: the number of feature per frame is 40, with 40 mel bands, 20 ms window and 50 % overlap. This means, using formula 1, that it is necessary to use 440 bit per window and 220 for the hop length since it is required a 50 % overlap (this means that the following frame extracted starts at half of the preceding frame).
Extracted data are then subdivided in sub-vectors of features composed by 1024 time frames: this is done to create an equal input for the neural network and enlarge the input data, instead of having less vectors of greater length. Moreover, this allows to standardize the input dimension in an easy way, since at most it is required to pad 1023 frames, while processing each extracted vector without subdivision would have brought bigger padding and much wasted computational power.

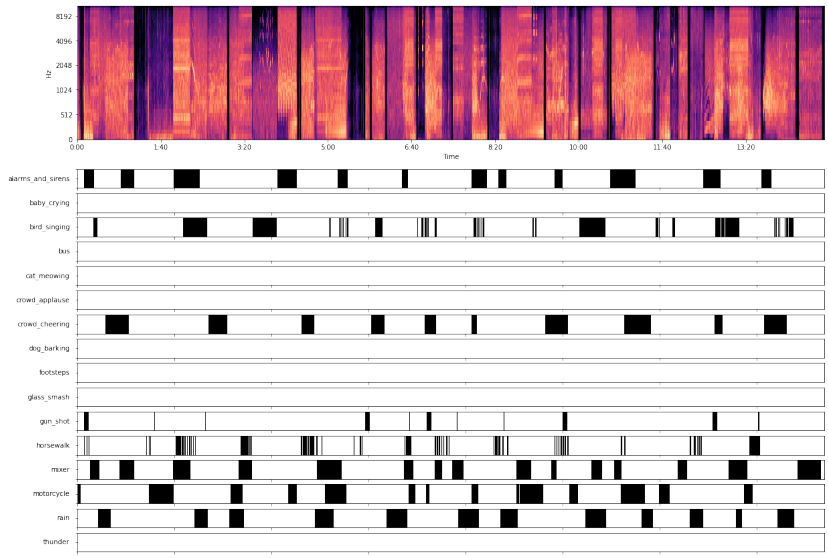


Figure 5: Representation of pre-processed data with Mel spectrogram on the upper part and labeled data in the lower part.

Fig. 5 contains an example of visualization of processed data and associated labels: the upper part of the image contains a visualization of what the audio file looks like in Mel Spectral band, from which the MFCCs are extracted to fed the network. The lower part is labeling associated to the audio file: the black parts are where the category on the left is present in the audio file, a complete white plot means that the category is not present at all in the file. Labeling procedure will be explained in following section, 3.3.

3.3 Data labeling

Since here we are dealing with polyphonic SED, the data labeling is more complex of what it has been seen in section 2.2. Data are labeled manually with a one hot encode: each sub-vector is instead a matrix of 1024 time frames by 40 features, and each time frame must be labeled with 16 possible classes; a zero if the class is not present and 1 if the class is present in the chosen time frame. This method will output a 1024 time frame by 16 classes matrix which will be used as ground truth during the training. This way of labeling will eventually bring a label matrix whose composition is mainly based on zeros, which will made a hard problem but the models proposed are able to bring satisfactory results. This means that it is not good to rely only on accuracy since the dataset is highly unbalanced, so authors of [1] propose to use f1-score. This fact can be easily recognized in lower part of figure 5.

3.4 Model formulation

This section will explain multiple model architecture introduced for poly SED: a baseline, a dilated baseline, a dessed baseline and a dessed-dilated approach (*dessed stands for depth-wise separable convolution based model*). Each model has input of 1024 time frame by 40 features by 1 channel and outputs a 1024 time frame by 16 classes matrix.

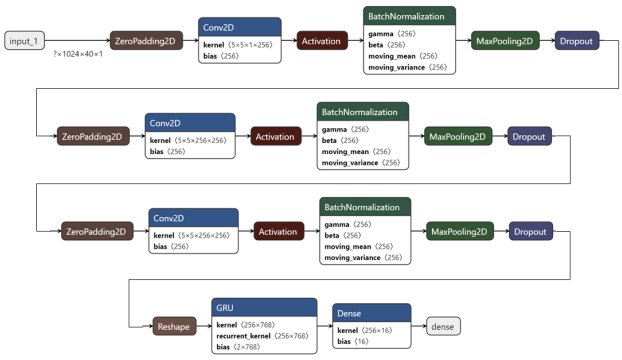


Figure 6: Baseline model for polyphonic SED.

The baseline model is similar to the one used in section 2.3, but the GRU is configured to give back also the time frames and not only the features. Other different parameters are the activation function of dense layer, a *sigmoid*, the loss function, *binary cross-entropy*, and the metric, *binary accuracy*. A representation can be found in fig. 6.
The dilated baseline substitutes the recurrent neural network with a dilated convolution: the block comprehends a dilated convolution, batch normalization, max pooling and dropout; it is followed by a 1x1 convolution with (1,3) stride since the original paper [1] uses a dilated convolution directly with stride (1,3) but the framework used in this project (Keras) does not allow to specify both dilation and stride different from 1, so I though that a possible solution to overcome this could have been to use the 1x1 convolution with (1,3) stride.
Different values of kernel and dilation has been proposed in [1], I have chosen a 3x3 kernel with 10 as time dilation rate since it seemed a good compromise. Before executing the dilated convolution, input is padded on temporal dimension only of a value equal to the dilation rate to obtain an untouched temporal dimension as output of convolutional layer. All other unspecified parameters are unchanged and equal to baseline model. The adoption of dilated convolutions allow to model long temporal context, reduce parameters number and eliminate the dissolving gradient for longer temporal sequences. Fig. 7 contains a representation of this model.

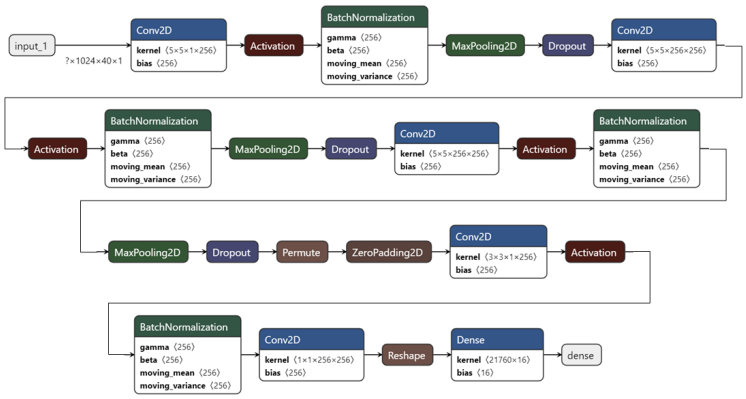


Figure 7: Dilated baseline model for polyphonic SED.

The dessed baseline substitutes the convolutional layers with depth-wise separable convolutions; all other operations and parameters are equal to the baseline. This substitution allows to drastically reduce the amount of used parameters in the first layers. It is possible to see a representation of the model in fig. 8.

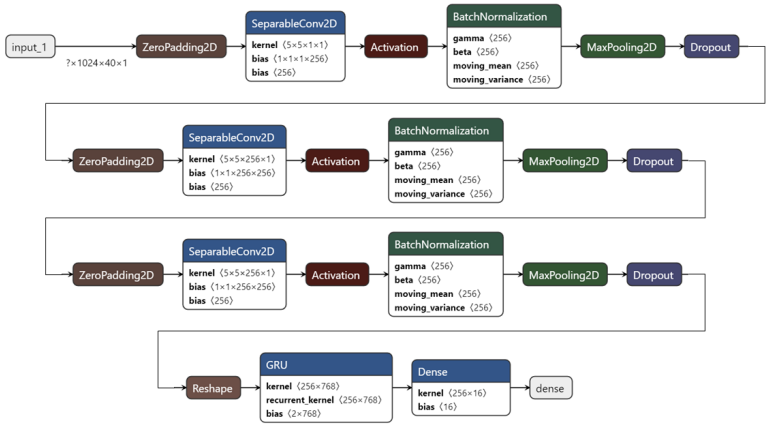


Figure 8: Dessed baseline model for polyphonic SED.

The combination of dessed and dilated model is what the reference paper [1] proposes as new approach to polyphonic SED, which theoretically should decrement the total number of parameters of about 7 times and give better prediction. During the training phase I had to propose a modification of the dilated part since the 1x1 convolution block does not work well in cooperation with depthwise separable convolution so I adopted a max pooling operation with kernel and stride of dimension (1,3) and this gave satisfying results. A representation can be found in fig. 9.

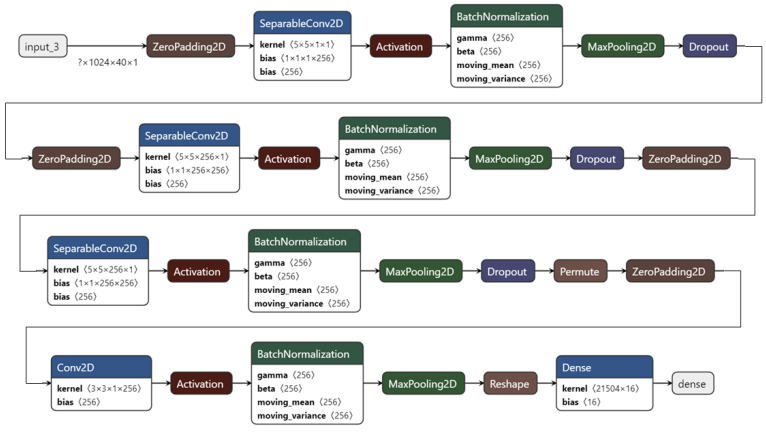


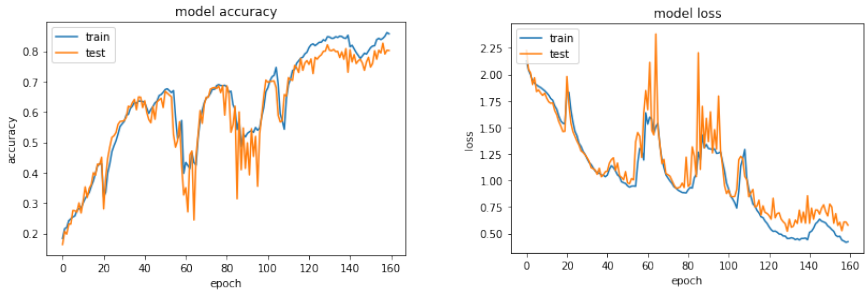
Figure 9: Proposed model for polyphonic SED.

4 Experimental Results

This section contains the practical results achieved training the models and testing them. Experiments have been performed both on Google Colab [9] both on my local machine with Ryzen 7 3700x, 16 GB RAM and RX 5700. Initially I tested training on CPU but it was too much slow, about 6 times vs a GPU, so I tested both on Colab and on my local GPU. The frameworks used to work with AMD GPU are explained in section 5.
Both dataset has been divided as follows: 60% for training, 20% for validation and the remaining 20% as test. Model has been trained looking at validation loss: the model with lowest value is saved and the training is stopped when the monitored value has not decreased for longer than 30 epochs. Train is performed with batch size of 16 where possible or 8 if memory is not enough. All the code is implemented exploiting Keras framework [8].

4.1 Monophonic results

Baseline model performs well overall: it has been trained for 159 epochs, a total of 5 hours and a half. Training graphs showing accuracy and loss both on train and validation set are in fig. 10. Graphs show a very strange behavior since learning does not create, in this case, smooth functions, but curves are instead with many hop and gaps in the values between different epochs.



(a) Accuracy.

(b) Loss.

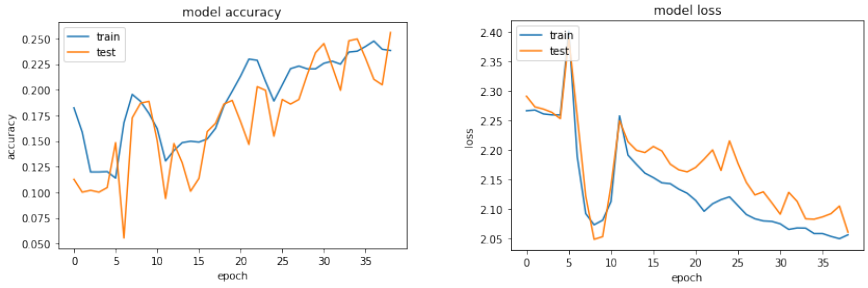
Figure 10: Baseline training graphs

This model achieved an overall accuracy of 0.824; differently from what it has been said in section 2.1, the classes with lowest number of elements are not the ones with lowest f1-score, probably that is caused by some difficulties which depend on sound properties primarily and not on class population. Table 3 contains values of f1-score for each class for values tested on baseline model. The value of loss function, instead, goes down till 0.521, and it's the lowest value recorded during training; on test set it is 0.524.
For what concerns the proposed model, it does not adapt well to the monophonic SED and performs very poorly. Figure 11 contains accuracy and loss graphs

Label	f1-score	Label	f1-score
air_conditioner	0.921	engine_idling	0.905
car_horn	0.857	gun_shot	0.813
children_playing	0.706	jackhammer	0.866
dog_bark	0.759	siren	0.880
drilling	0.788	street_music	0.780

Table 3: Monophonic baseline f1-score per class.

recorded during training: also in this case the curve are not so smooth, but the overall performance is not comparable with baseline. Accuracy on test set is 0.263 and loss is 2.04, which are bad results compared with baseline. Moreover, this model has trained for a lower amount of epoch (39) due to the loss value not decreasing after 30 epochs; looking at the graphs, maybe changing the value to 40 epochs may have continued the training with better results but I suppose that overall results would not have been comparable with baseline model.



(a) Accuracy

(b) Loss

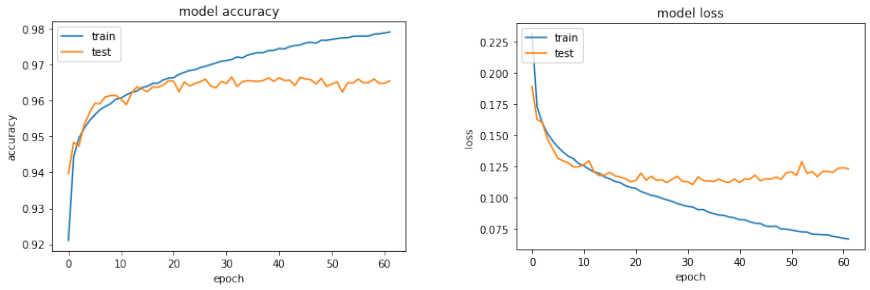
Figure 11: Proposed model training graphs.

It must be said that the proposed model was developed for polyphonic SED, so maybe it's the fault of adaptation proposed and a better model can be proposed to overcome the bad result of this model. I suppose the part causing this poor results is the connection between the dilated convolution and the dense layer: two ideas arises, or the dilated convolution is not the right choice when it is needed just a label for each recording and not a multilabel, or the global average pooling is not the right way to downscale from dilated convolution to dense layer.

4.2 Polyphonic results

Baseline model performs good: it trains for a small number of epochs, just 62, and learns very fast, getting a test accuracy of 0.962 and test loss of 0.135. From fig. 12, it can be seen that this model presents a visible overfit after about 20 epochs, the best model is taken from epoch 32 so it is the best combination

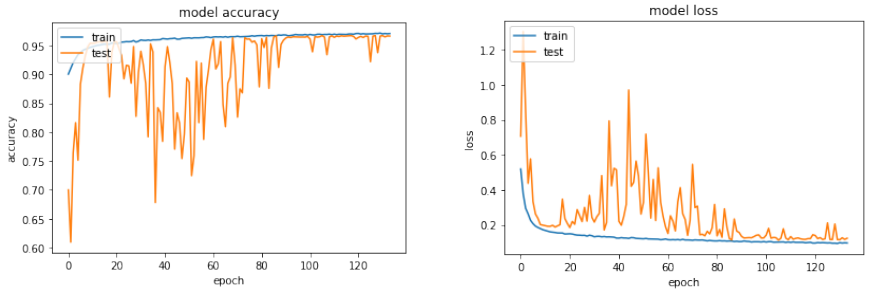
between generalization and overfitting of this case.



(a) Accuracy. (b) Loss.

Figure 12: Baseline training graphs.

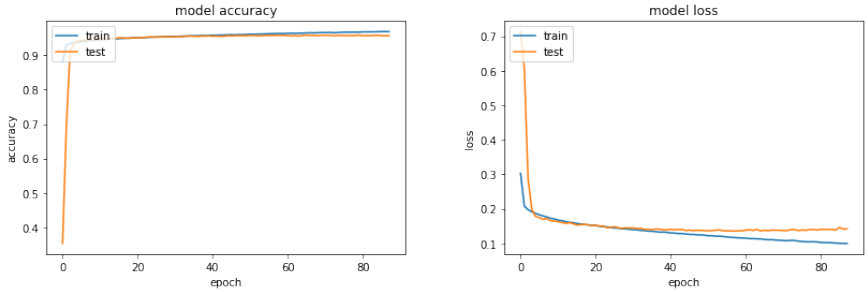
F1-score has been computed manually on a micro averaging measure; in this case it is 0.715, higher w.r.t. what achieved in [1]. Training required half an hour in this case.



(a) Accuracy. (b) Loss.

Figure 13: Dilated model training graphs.

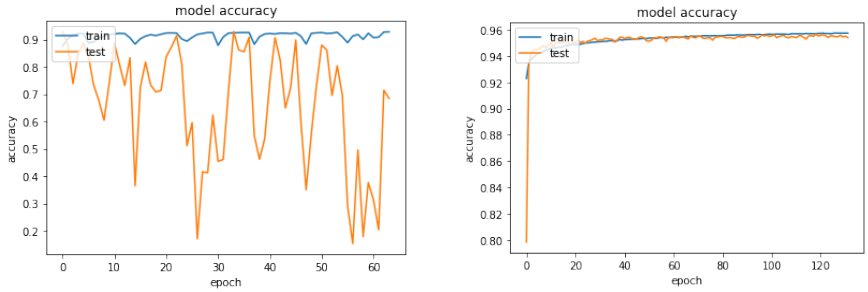
The adoption of dilated convolution allows the model to be trained for a higher number of epochs, 136, achieving better accuracy and loss w.r.t. the baseline model, 0.966 and 0.115 respectively. The training curves are a bit more noisier than baseline ones, as reported in fig 13.
Overall f1-score is 0.7299 and it is the best value achieved in the polyphonic results. The model has been trained for 7 hours and a half.
The introduction of depth-wise separable convolutions allows to reduce drastically the number of parameters needed by the model and test accuracy does not change so much, 0.953, and 0.152 for test loss. The model has been trained for 88 epochs for a total of an hour. The model gives rise to a tiny overfit with growing epochs, but paltry compared to what happened with baseline model. Training history graphs can be found in fig. 14.



(a) Accuracy. (b) Loss.

Figure 14: Dessed model training graphs.

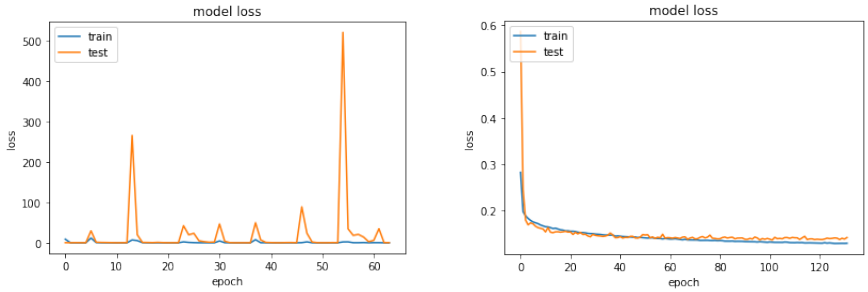
Overall the model loses about 0.1 point of f1-score, 0.633, which is enough to say that the baseline is better considering the score but the number of parameters is lower in this new approach.



(a) 1x1 convolution. (b) Max pooling.

Figure 15: Proposed model accuracy comparison.

For what concerns the proposed model, here there is a comparison on both the methods tested to connect the dilated convolution with the dense layer. The 1x1 convolution with (1,3) stride performs poorly: after 64 epochs the accuracy arrives to 0.928 and loss to 0.24. This may seem a good performance, but if we look at f1-score, it shows the bad score: 0.125, which is very low compared to the other proposed approaches. Fig. 15 shows differences in accuracy of the two new model approaches.
The other alternative used, is to adopt a max pooling layer with (1,3) dimension both for kernel and stride instead of 1x1 convolution; this different approach has the hoped benefits: 0.952 accuracy and 0.148 loss. The f1-score is 0.628 which is lower compared to the other proposed approaches but a lot higher w.r.t. the 1x1 convolution case. Fig. 16 shows the history of loss functions in both new model approaches.



(a) 1x1 convolution.

(b) Max pooling.

Figure 16: Proposed model loss comparison.

Even if the f1-score is much different, both new models have a similar training time around 3 hours and a half.

Model	Accuracy	Loss	Training time	f1-score	Number of parameters	Dimension
Baseline	0.962	0.1356	0:32	0.71566	3.7M	43 KB
Dilated	0.966	0.115	7:30	0.7299	3.7M	43 KB
Dessed	0.9532	0.1524	0:59	0.6332	0.5M	6.5 KB
Proposed Model 1x1 Convolution	0.9287	0.241	3:14	0.1245	0.5M	6 KB
Proposed Model avg pooling	0.9523	0.148	3:26	0.6281	0.5M	6 KB

Table 4: Polyphonic model results.

Table 4 sums up the polyphonic results: the best performing model is the dilated one, where only the recurrent layer is substituted, but the advancement w.r.t. the baseline, 0.015 of f1-score, does not justify the 15 times more time needed for training. The adoption of depth-wise separable convolution allows to drastically reduce the number of parameters and the dimension of the model but the f1-score looses about 0.1 w.r.t. the baseline and it is needed 6 times more for training. Basically, the new approaches allow to obtain a smaller model with slightly decreased performances that are well suited for deployment on machines which have limited resources (86% dimension reduction with 14% performance loss).
The obtained results are slightly different to [1]: the baseline and dilated perform about 0.1 higher of f1-score in this case, while dessed and proposed model are in line with reference paper but have a smaller amount of parameter (0.5M vs 0.7M) probably due to the different framework used in original paper (pytorch).

5 How to train NN on AMD GPU

Since my machine has a gaming AMD GPU, I decided to try finding a way to train models on it. There exists different ways to do it:

- ROCm [10] is an open platform GPU compiler developed by AMD, which is capable of running deep learning framework on top of it but my GPU is still not supported.
- PlaidML [11] is a tensor compiler running as back-end of common frameworks. It works well with Keras but has the drawback that Keras does not support back-end different from TensorFlow starting from version 2.4 and it was refactored from version 2.2.5 so PlaidML allows to work with Keras 2.2.4, which is not the leading edge version but all the things needed by this project work well.
- DirectML [12] is a back-end developed by Microsoft which supports TensorFlow version 1.x on not CUDA GPUs, latest supported version is 1.15, which is the last one till the big refactor which brought version 2.0 of TF.

So both alternatives use old versions but the PlaidML one seems more promising since Keras code written for version 2.2.4 is completely usable by newer version: the code has been tested both on local machine with PlaidML as back-end both on Colab with newer Keras version and TensorFlow as back-end and worked without problems.
Later this year, both frameworks are working on bigger updates: DirectML is working on bringing support for TensorFlow 2.x, while PlaidML is working on integrating MLIR, a new compiler, which has been integrated also by TensorFlow last years so in the end also PlaidML will support TensorFlow 2.x exploiting MLIR as abstraction layer with TF.

6 Conclusions

Different approaches for Sound Event Detection have been proposed in this work: for monophonic SED, the baseline works good while the new approach has some problems and needs some improvements to be competitive; for polyphonic SED, four different approaches have been proposed with very good results, maybe the best approach is the baseline if one has small amount of time to train the model but has enough space for storing the model, while both dessed and dessed-dilated have a small decrement in performance but huge decrement in model dimension.
Even if the code has been developed for an old version of Keras, it is completely compatible with newer version, as shown by the fact that the model has been trained also on Colab with newest libraries versions.
Overall it has been a very good learning experience, and I am satisfied by the work produced. All the code related to this implementation can be found on my GitHub: `https://github.com/Prinzivalle/Sound-Event-Detection`.

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.

[8] Keras. `https://keras.io/`.

[9] Google colab. `https://colab.research.google.com/`.

[10] Radeon open compute. `https://github.com/RadeonOpenCompute/ROCm`.

[11] Plaidml. `https://github.com/plaidml/plaidml`.

[12] Directml. `https://github.com/microsoft/DirectML`.