Urban sound classification

Francesco Tomaselli

July 23, 2021

Contents

1 Introduction

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

The goal of this project is to build a neural network to classify audio files from the *UrbanSound8k* dataset.

This dataset contains audio divided in ten classes, each one representing a different type of city sound, for instance, we can find *car horns*, *dogs barking*, *sirens*, etc. A deeper discussion about the dataset is present in the Subsection 2.1 on the following page.

The presented methodology is composed of three main parts. The first step is to extract relevant features from audio files using the *Librosa* library. This is discussed on Section 2 on the next page.
The next step consists in composing and refining a neural network to classify the data obtained from the previous step. This part is made possible by the *Keras* library and it is discussed in Section 3 on page 5.
Lastly, results from the classification, namely accuracy and standard deviation among test sets, are presented in Section 4 on page 7.

The project is developed in *Python* and the code is structured in a *src* package. Each one of the sub-packages contains code to deal with the different parts of the project. For instance, the *data* folder contains the classes to extract features and to manage a dataset.

In addition to the package there is a *notebook* folder containing the different steps of the project and the various experiments made.

2 Feature extraction

This Sections presents the original dataset structure and the steps followed to create training and test sets from it.

Note that the models mentioned in this section are three layers networks with a reasonable number of neurons, trained for 100 epochs with default parameters for the Stochastic Gradient Descent optimizer. The accuracy on the training is computed with a cross validation approach. Details about models and validation techniques used in the project can be found at Section 3 on page 5.

2.1 Dataset structure

The dataset contains ten folds of audio samples, each one about four seconds long. The samples are divided in ten classes among ten folds. The following table shows the classes and their relative frequency in the dataset.

Class name	Number of samples
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

The training set consists of the first four folds plus the sixth, the other folds create five different test sets. The table shows the numerosity of the different sets.

Dataset	Number of samples
Training set	4499
Test set 5	936
Test set 7	838
Test set 8	806
Test set 9	816
Test set 10	837

2.2 First dataset

Choosing the features to extract was challenging due to the inexperience with working on audio files.

The *Librosa* library provides many feature to choose from, to keep it simple, for the first try with this dataset the extracted features are these three ones:

1. *Mel-frequency cepstral coefficients*
2. *Chromagram*
3. *Root-mean-square*

Each feature consists of an array of arrays containing measurements. A series of functions were applied to each sub-array and results were concatenated in a final feature vector. The functions applied are *minimum*, *maximum*, *mean* and *median*.

This approach resulted in 132 components feature vectors.

Feature scaling After testing some neural networks on the first dataset the results were not promising. One of the reasons is the big difference in ranges among feature vector components.

To mitigate this effect a *StandardScaler* from *sklearn* was applied. This lead to an improvement on the results using the same model as before.

2.3 Extended dataset

To improve results on the training set new features are added, namely:

1. *Zero-crossing rate*
2. *Roll-off frequency*
3. *Spectral flux onset strength*

After applying the same four functions to these three new arrays, a total of 12 new features are added to the dataset. Scaling yield to promising results on the first dataset, so the same approach is applied to the extended dataset.

After testing a network on the new dataset results improved once again.

PCA Adding new features can lead to better results in the end but they all need to be useful to the model, so the extended dataset was subject of some experiments with feature selection, in particular *PCA* algorithm from *sklearn* was applied.

After some experiments with the number of features to select, 120 out of 144 features were selected. This led to a small improvement on the training set, so this final reduced dataset was selected to perform hyperparameter tuning.

3 Model definition

This Section presents how the networks used on the training set are structured and how the Hyperparameter tuning phase was performed.

3.1 Neural network structure

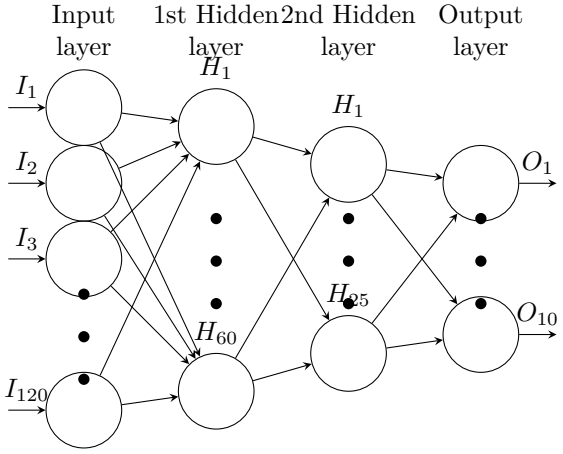
The starting point for the neural network structure was a reasonable network in terms of hidden neurons to prevent over-fitting, indeed a high number of units in the hidden layers would end up in learning too much from the dataset.

The rule of thumb followed to decide hidden neurons quantity is the following:

$$\#hidden\ neurons = \frac{2}{3}\#input\ neurons + \#output\ neurons$$

The next step was to decide the hidden layer number. To respect the number of hidden neurons, two hidden layers were considered. A more complex structure in terms of layers number would mean having a real small number of neurons per layer.

To give reference, this is the model used on the PCA training set, mentioned at the end of Subsection 2.3 on the previous page, with 120 input features.



The activation function for the input and hidden layers is a *Relu* and the output one uses a *softmax*. The loss used is the *Sparse Categorical Crossentropy loss* as it is suited for this kind of problems.

As discussed in the previous Section, models with this logic for construction were tested on the four training sets to choose the one to tune the final network on.

3.2 Hyperparameter tuning

Choosing the training set with PCA applied, led to the best results with cross validation, although the model was reasonable, it can not be the final one, as many parameters were left on their default value, for instance, *learning rate* and *momentum* of the optimizer were not touched.

The main goal now is to experiments with different model parameters to find the best one.

Grid vs Random search Two of the most commonly used strategies in Hyperparameter optimization are *Grid* and *Random Search*.

In both cases we define ranges of parameters to test different combinations, for instance, fixed the number of neurons, one could try to find the best combination of learning rate and momentum that optimize accuracy on the training set.

While similar, the two methodologies differs in the amount of exploration they do. The Grid search try all the possible combinations of parameters, while the Random approach fixes a number of iterations and picks an arbitrary combination each time.

Obviously the first one is more computationally expensive than the second, if we fix a small amount of possible iterations, but in theory it finds a better result than going the random route. Nonetheless the Grid Search can led to over-fitting, and in practice Random Search in preferred.

To get the best out of the two techniques, a first Random Search is performed on a larger parameter space, then, a Grid Search is used on much smaller ranges to optimize the previously found model.

Random search Starting from the model introduced in the previous Subsection, parameters are tweaked to find a better one. The optimizer used is the *Stochastic Gradient Descent*. The considered ranges for parameters for this run are:

1. *Neurons*: first and last layer stay the same, while the two hidden layers are tested with a number of neurons respectively equals to:
$$60 + 2i \text{ and } 25 + 2j, \text{ with } i, j \in \{-2, -1, 0, 1, 2\}$$
2. *Learning rate*: $0.001, 0.01, 0.1, 0.5$
3. *Momentum*: $0.0, 0.01, 0.1, 1$
4. *Epochs*: $80, 100, 120$
5. *Batch sizes*: $32, 64, 128$

An early stopper is used on the fit, and cross validation with 5 folds is performed on each combination. The total possible models are 3600, but the search was performed with 200 iterations in total.

Grid search The best model from the random search was then fine-tuned with a Grid Search, where ranges are in the proximity of the parameters found by the previous search.

4 Results

4.1 First dataset

4.2 Extended dataset

5 Final remarks