Urban sound classification

Francesco Tomaselli

August 8, 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 [1].

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 made at 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, 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 discussed in Section 3 on page 6.
Lastly, results from the classification, namely accuracy and standard deviation among test sets, are presented in Section 4 on page 10.

Project structure The project folder is structured as follows:

- *src*: this contains the source code for the project. Each sub-folder contains code for a specific part of the processing. In particular, the *data* folder holds classes to extract features and to manage the dataset, the *model* folder contains the class to create the Neural Network, then *utils* stores utility functions to measure performances;
- *data*: here data is stored, there is a *processed* and *raw* sub-folders, where the first stores computed datasets and the second original data;
- *models*: trained models are saved here;
- *notebooks*: the folder contains the Jupyter notebooks used in the project, where code from src is practically used.

The code is written in *Python* [2].

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 *multilayer perceptron* a reasonable number of neurons, trained for 100 epochs with default parameters for the *Stochastic Gradient Descent optimizer.* [3] [4]

The accuracy on the training is computed with a *cross-validation* approach. [5] Details about models and validation techniques used in the project can be found at Section 3 on page 6.

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

All the operations on the datasets are performed with *Pandas* library. [6]

2.2 First dataset

Choosing the features to extract was challenging due to the inexperience working on audio files, thus some *research* on what features to choose was necessary [7].

To extract features from audio files *Librosa* was used. [8] The 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* from the *Numpy* library [9].

This approach resulted in 132 components feature vectors.

Parallelizing feature extraction Extracting the three features listed above is really intensive but the task is easily parallelizable, in fact, each file is independent from one another.

For this purpose *Dask* was used to speed up the computation, and extract features from audio files in a multi-threading fashion. [10]

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 *scikit learn* was applied [11]. The result is a dataset where each feature has more or less a distribution centered in zero with unit variance. This lead to an improvement on the results using the same model as before.

2.3 Extended dataset

Results using the three features named in the previous Subsection are promising but not enough, thus, to improve results on the training set, new audio characteristics are extracted, namely:

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

As before, *minimum*, *maximum*, *mean* and *median* are applyied to each feature vector and results are concatenated, resulting in 12 new features for each audio file.

Scaling yield to promising results on the first dataset, so the same approach is applied to the extended one.

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

Feature selection 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 scikit learn was applied [12].

The main idea is to select a reduced number of features from the total, without loosing information. This approach often leads to better results, as useless features are discarded.

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

3 Model definition

This Section presents how the networks used on the training set are structured, the second Subsection gives an overview of the performances on the different training sets, lastly, the Hyperparameter tuning phase is described.

3.1 Neural network structure

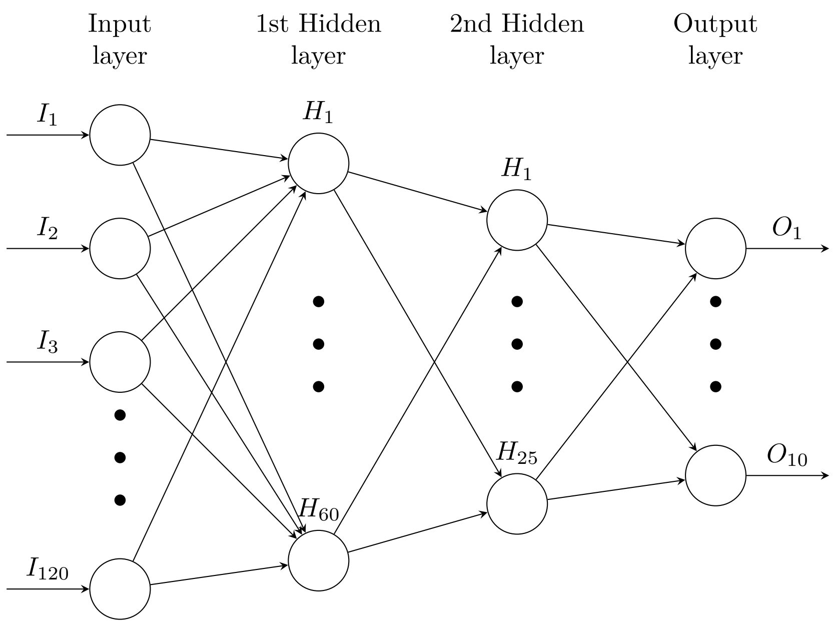
The starting point for the neural network structure was a reasonable network in terms of hidden neurons to prevent over-fitting, indeed a high number of units in the hidden layers would end up in learning too much from the dataset.

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 page 4, with 120 input features.



The activation function for the input and hidden layers is a *Relu* and the output one uses a *Softmax* [13] [14]. The loss used is the *Sparse Categorical Crossentropy loss* as it is suited for this kind of problems [15].

To build the actual model *Tensorflow* and *Keras* library are used. [16] [17]

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 Initial training set results

Section 2 on page 3 talked about the four different training sets obtained from the dataset and, without going into details, stated that there was continuous improvement. This subsection presents the results obtained on the training set.

Note that all the random seeds used by Tensorflow were fixed to make results reproducible.

Class unbalance ...

Cross validation To estimate performance on the training set cross-validation with five folds was used. Basically the dataset is divided into five parts and a model is repeatly trained on four and tested on one. The mean accuracy on the test folds gives a hint about the model performance.

Results For each training presented training set, a model was defined with the structure presented at the beginning of this Section, and these are the results:

Training set	Mean accuracy	St. deviation
132 features unscaled	0.0669	0.0488
132 features scaled	0.5201	0.0668
144 features scaled	0.5586	0.0842
120 features reduced with PCA	0.5608	0.0998

There is a great improvement after scaling the training set, after that small refinements were made. As accuracy is the best on the last dataset, this was the one selected to perform the Hyperparameter tuning.

3.3 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 left untouched.

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

Grid and Random search comparison Two of the most commonly used strategies in Hyperparameter optimization are *Grid* and *Random Search* [18].

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 layers 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*: $0.0, 0.01, 0.1, 1$.

An early stopper is used on the fit with 100 *epochs*, *batch size* is fixed at 32, and cross validation with 5 folds is performed on each combination. The total possible models are 400, but the search was performed with 100 iterations in total.

The best model found was the following:

- *Neurons*: 120 for input, 62 for the first hidden layer, 27 for the second, and 10 for output;
- *Momentum*: 0.1;
- *Learning rate*: 0.1.

Comparing the initial model with the one found now, a small improvement can be seen both in accuracy and standard deviation:

Model	Mean accuracy	St. deviation
Initial model	0.5608	0.0998
Random search result	0.5857	0.0715

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.

Parameters this time was:

1. *Neurons*: first and last layer stay the same, while the two hidden layers are tested with a number of neurons respectively equals to:

$$62 + i \text{ and } 27 + j, \text{ with } i, j \in \{-1, 0, 1\}$$

2. *Learning rate*: $0.08, 0.1, 0.12$;
3. *Momentum*: $0.08, 0.1, 0.12$.

The total possible models was 81, the best found is the following:

- *Neurons*: 120 for input, 62 for the first hidden layer, 28 for the second, and 10 for output;
- *Momentum*: 0.12;
- *Learning rate*: 0.08.

As before, a comparison shows that Grid Search fine tuned the model to have better accuracy, unfortunately standard deviation increased by a small quantity:

Model	Mean accuracy	St. deviation
Initial model	0.5608	0.0998
Random search result	0.5857	0.0715
Grid search result	0.5877	0.0763

This last model is the one selected to evaluate test set performances.

4 Results and final remarks

This last Section shows the results obtained on the different test sets and possible improvements.

4.1 Test set results

The final model selected is the last one found on previous Section, found by the Grid Search. As stated on the first Section, the five test sets are made out of the folds number five, seven, eight, nine and ten.

The following are the results on those folds:

Test set	Accuracy
Fold 5	0.6731
Fold 7	0.6301
Fold 8	0.7184
Fold 9	0.6630
Fold 10	0.6906

Finally, the mean accuracy and standard deviation for the test sets are:

Mean accuracy	Standard deviation
0.6750	0.0293

4.2 Future works

The results on the test sets are promising, but there is definitely room for improvement.

Indeed, a more refined training set creation can be made, by exploiting more features from the Librosa library, testing different type of scalers and experimenting with different feature selection techniques.

Finally, the models used in the project are simple, more complex Neural Networks with Convolutional layers could learn more from the training set and improve performances on test.

References

[1] Justin Salamon, Christopher Jacoby, and Juan Bello. A dataset and taxonomy for urban sound research. 11 2014.

[2] Python. https://www.python.org.

[3] Wikipedia. Multilayer perceptron. https://en.wikipedia.org/wiki/Multilayer_perceptron.

[4] Wikipedia. Stochastic gradient descent. https://en.wikipedia.org/wiki/Stochastic_gradient_descent.

[5] Scikit Learn. Kfold. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html.

[6] Pandas. https://pandas.pydata.org.

[7] Joel Jogy. How i understood: What features to consider while training audio files? https://towardsdatascience.com/how-i-understood-what-features-to-consider-while-training-audio-files-eedfb6e9002b.

[8] Librosa. https://librosa.org.

[9] Numpy. https://numpy.org.

[10] Dask. https://dask.org.

[11] Scikit learn. Standard scaler. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.

[12] Scikit Learn. Pca. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html.

[13] Wikipedia. Rectifier. https://en.wikipedia.org/wiki/Rectifier_(neural_networks).

[14] Wikipedia. Softmax function. https://en.wikipedia.org/wiki/Softmax_function.

[15] Kiprono Elijah Koech. Cross-entropy loss function. https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e.

[16] Tensorflow. https://www.tensorflow.org.

[17] Keras. https://keras.io.

[18] Kishan Maladkar. Why is random search better than grid search for machine learning. https://analyticsindiamag.com/why-is-random-search-better-than-grid-search-for-machine-learning.