Urban Sound Classification with Neural Networks

Master in Computer Science

Manuel Dileo

December 2021

Abstract

This report describes the work behind the software project for the course of Statistical Methods for Machine Learning held by Professor Nicolò Cesa-Bianchi in the accademic year 2020/2021 for the Master in Computer Science. The task is to use TensorFlow 2 to train neural networks for the classification of sound events based on audio files from the UrbanSound8K dataset. I've experimented with different feature selection methods, feed forward neural networks fed with features extracted using the librosa library, convolutional neural networks fed with the spectogram of the audio files, multimodal neural network.

Introduction

The automatic classification of environmental sound is a growing research field with multiple applications to large-scale, content-based multimedia indexing and retrieval [1].
In this report I will discuss the use of TensorFlow 2 [2] to train neural networks for the classification of sound events based on audio files from the UrbanSound8K dataset. The dataset contains 8732 sound excerpts ($\leq$ 4 seconds) of urban sounds labeled with 10 classes.
Neural networks can be fed with raw data or proper features can be extracted. I will illustrate the use of the librosa library [3] to extract several useful features common in the field of sound events classification, or other less tailored features for the considered domain, to train feed forward neural networks (FFNN). PCA [4] and Boruta [5] will be used as dimensionality reduction techniques.
I will describe the use of the spectogram of the audio files as images to fed convolutational neural networks (CNN) where different kernel size will be explored.
I will illustrate multimodal neural networks (MMNN) where both audio features and images can be used to train the model.
I will describe experiments in which I used different feature selection methods, network architectures and some hyperparameter tuning, reporting the obtained average accuracy and standard deviation across the test folds.
This report is structured as follows: Section 1 describes the dataset, the preprocessing techniques used and the audio features extracted; Section 2 describes all the experiments done and briefly some tools and technologies used in this work and why I decided to use them.
The code is available on a github repository[1]and executable in Google Colaboratory.

1 Dataset and preprocessing

1.1 Dataset Description

The *UrbanSound8K* dataset contains 8732 sound excerpts ($\leq$ 4 seconds) of urban sounds labeled with 10 classes: textitair_conditioner, *car_horn*, *children_playing*, *dog_bark*, *drilling*, *enginge_idling*, *gun_shot*, *jackhammer*, *siren*, and *street_music*.
The dataset is provided with 10 predefined folds; each fold contains audio files in wav format. The files metadata are available in the UrbanSound8K.csv file provided with the dataset. Metadata includes the name, fold and classID of the audio files. All audio files have a unique name and ClassIDs are used as labels to predict.
Figure 1 shows the pie plot of the classID distribution and tells us that the dataset is quite balanced.

[1]`https://github.com/manuel-dileo/urban-sound-classification` December 2021

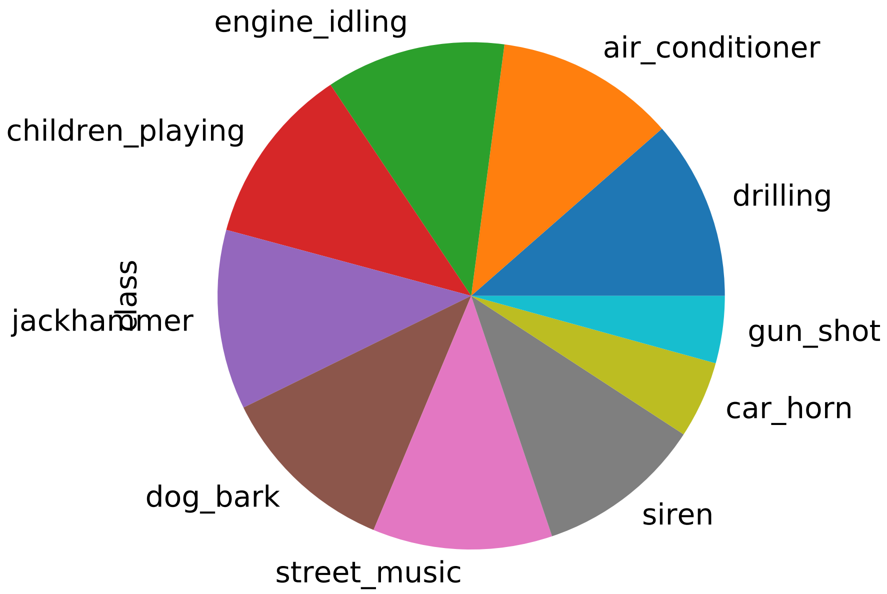

Figure 1: Pie plot of the distribution of the metadata field classID.

1.2 Feature extraction and preprocessing

Using librosa library, I read each audio file to obtain the corresponding numpy [6] array with sampling rate equal to 44100Hz. Then, for each urban sound I've extracted the features that seems to me the most interesting for my task, that are the following:

- The *Mel frequency cepstral coefficients* (*MFCCs*) of the signal, that are a small set of features which concisely describe the overall shape of a spectral envelope. They models the characteristics of the human voice. I used number of MFCCs equal to 20.

- A *chroma vector*, that is typically a 12-element feature vector indicating how much energy of each pitch class, $\{C, C, D, D, E, \ldots, B\}$, is present in the signal. In short, it provides a robust way to describe a similarity measure between audio pieces.

- The *Zero-Crossing Rate* (*zcr*), a very simple way for measuring the smoothness of a signal. A voice signal oscillates slowly, for example, a 100 Hz signal will cross zero 100 per second, whereas an unvoiced fricative can have 3000 zero crossings per second. I computed the zero crossing rate using an hop length equal to 256 and a frame length equal to 512.

- Other less tailored features for the considered domain, that are *Root Mean Square* (*rms*) of the audio signal and its statistics (min, max, average, standard deviation). I computed the rms using an hop length equal to 256 and a frame length equal to 512, padding the signal by $[\frac{frame\_length}{2}]$ on either side.

The description of this features are taken from KDNuggets[2]. Both MFCCs, chroma vector and zcr were aggregated through time using the average.
Then, I've applied a StandardScaler[3] on each feature and combined them with the file name, fold and *classID* to obtain the final dataset. Dataset was splitted in train and test set w.r.t folds according to requirements.

I also computed the spectogram of each audio file, saved them as RGB images and then scaled to 64×64 pixels as input for CNNs.
A spectrogram is a visual representation of the spectrum of frequencies of a signal as it varies with time. It is a graph with two geometric dimensions: one axis represents time, and the other axis represents frequency; a third dimension indicating the amplitude of a particular frequency at a particular time is represented by the intensity or color of each point in the image.
Spectograms were obtained using a colormap "inferno", 2048 of data points used in each block for the *Fast Fourier Transform (FFT)*, sampling frequency equal to two, center frequency of the data equal to zero, 128 as number of points of overlap between blocks, and dB as scale. Other parameters can be found in matplotlib documentation[4] but were setted to default values.

2 Experiments

2.1 Feed Forward Neural Networks

A *feedforward neural network (FFNN)* is an artificial neural network wherein connections between the nodes do not form a cycle. In particular, in this report we refer to FFNNs by exploring a specific FFNN family that is the *Multi-layer Perceptron (MLP)*. In a MLP, neurons in a specific layer can have only connections with the neurons in the next layer. In our setting, each neuron in an hidden layer has a connections with each neuron in the next layer (Dense layer).
I experimented with different FFNN architectures by conducting a model search from no hidden layers upto three hidden layers with 32, 24 and 16 units in the first, second and third hidden layers. The number of input neurons was equal to the number of features, that is 37, while the number of output neurons was equal to the number of class, that is 10. The number of hidden neurons were

[2]`https://www.kdnuggets.com/2020/02/audio-data-analysis-deep-learning-python-part-1.html` December 2021
[3]`https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html`,December 2021
[4]`https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.specgram.html` December 2021

chosen to not exceed twice the number of input neurons plus the number of output neurons as rule of thumb.
I used the accuracy as evaluation metric and the sparse categorical cross-entropy as loss function. The hidden layers has a ReLu activation function to reduce the vanishing gradient problem. The output layer has a softmax activation function. Given a probability vector, for instance the vector of the probability for a certain point for each class to belong to a certain class, the softmax activation function has the effect of emphasizing the distance between the highest probability value and the others, and this is a good behaviour when we have to deal with multilabel classification [7].
I used the early stopping callback[5] that stops the training when the accuracy has stopped improving with a level of patience, that is the number of epochs with no improvement after which training will be stopped, equal to three.
For the zero hidden-layer model, I used stochastic gradient descent (sgd) as optimizer while for the deep FNNNs I used adam [8], that is a stochastic gradient descent method based on adaptive estimation of first-order and second-order moments that introduces a sort of memory of how fast it was going in the previous steps to accelerate or decelerate the current descent. Adam tipically performs better than sgd in deep learning context[6].
I've also used a learning decay approach for the deep learning models and dropout layers, that simply disable a certain percentage of neurons in a certain layer during the training phase (in our case the 25% of neurons), as regularization technique. Each experiment was done using 100 as number of epoch and 128 as batch size. The learning rate decays as shown in Figure 2.

Table 1 shows the average and standard deviation accuracy on the test folds of the various FFNN architectures. The best results are achieved using two hidden layers and Figure 3 shows the bar plot of the accuracy on the different test folds. The results are in line with what is generally recommended by best practices for this type of task.
Note that I don't report the accuracy on training set that is general higher w.r.t the one obtained on test set. Also the early stopping callback stops the training after about 40 epochs compared to the expected 100. This two aspects make me suspect of overfitting. I think this behaviour happens mainly because of noisy labels assigned by human annotators. For this reason I decided to use few number of hidden layers and neurons and some regularization techniques.

2.1.1 Feature selection

Feature selection is a fundamental step in many machine learning pipelines. You dispose of a bunch of features and you want to select only the relevant ones and to discard the others. The aim is simplifying the problem by removing unuseful features which would introduce unnecessary noise.
I used Boruta algorithm to try to discard irrelevant features for my classification

[5]`https://keras.io/api/callbacks/early_stopping/`, December 2021
[6]`https://cloud.google.com/blog/products/ai-machine-learning/learn-tensorflow-and-deep-learning-without-a-phd`, December 2021

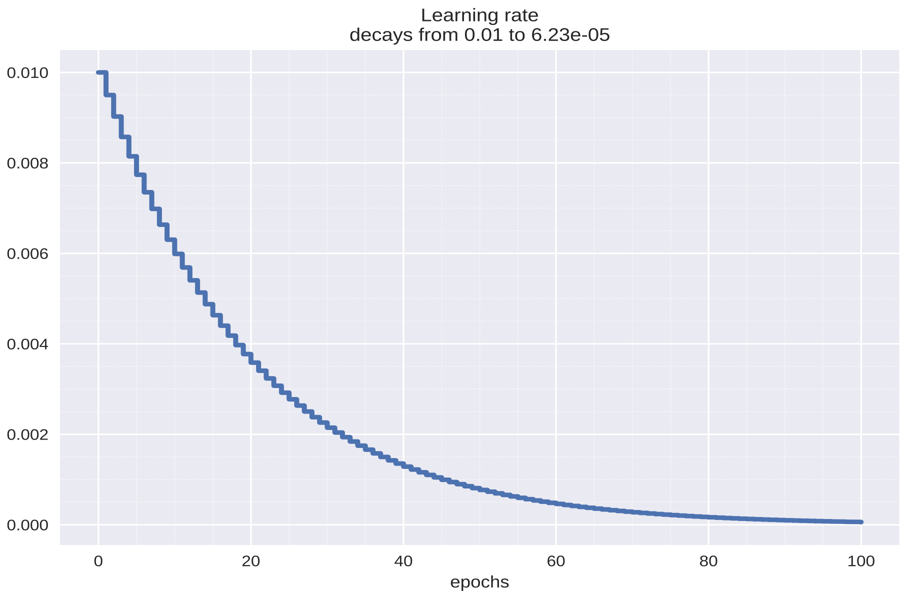

Figure 2: Learning rate decays from 0.01 to $6.23e - 0.5$ during 100 epochs.

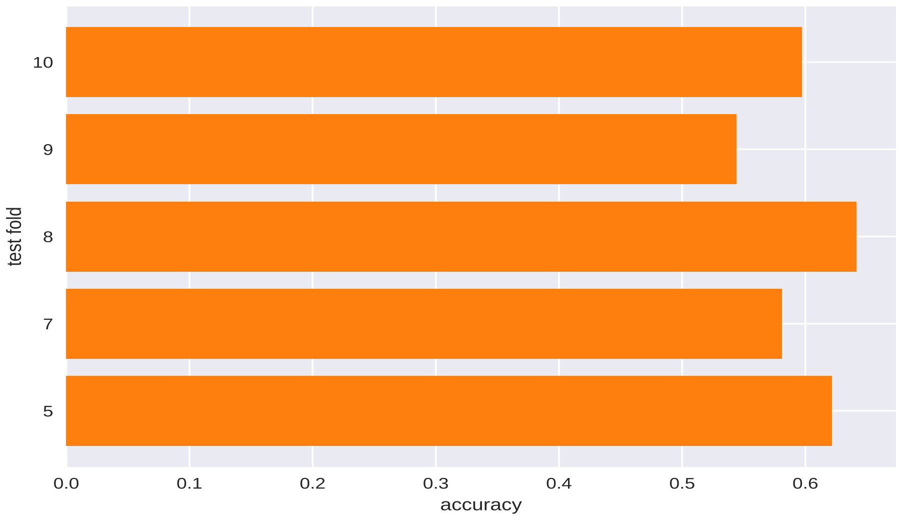

Figure 3: Bar plot of the accuracy of the two hidden-layer FFNN on the different test folds.

Network Architecture	Avg Accuracy	Std Accuracy
Zero Hidden-Layer	0.56	0.04
One Hidden-Layer	0.59	0.04
Two Hidden-Layer	0.60	0.03
Three Hidden-Layer	0.59	0.04

Table 1: Average and Standard Deviation accuracies on the different test folds for the different FFNN architectures.

task. Enter in the details of Boruta is out of the scope of this report but is an algorithm that starting from sklearn's SelectFromModel[7] introduces two ideas, shadow features and iterations on trial, resulting a very robust tool for feature selection.
I used Boruta with a *max_depth* five Random Forest Classifier, a *p_value* equal to 0.05 and a number of iterations equal to 100. After 9 iterations, Boruta decided to stop and to reject nothing so I did the same.
Then, I used PCA as feature extraction technique. First, I had performed a dimensionality reduction obtaining 20 components from all the original features and I trained the two hidden-layer FFNN with them.
The results were very poor and I thought the reason was the fact that I mixed different type of features to obtain the new resulting representation. So I decided to obtain a 20 components dimensionality reduction using PCA to pass from 20 to 10 component for MFCCs, from 12 to 5 for chroma vector and then consider jointly the resulting representations plus the other features (zcr, rmse statistics). Accuracy had improved by more than 10 percentage points but it was still unsatisfactory so I decided to not report it.

2.2 Convolutional Neural Networks

Convolutional neural networks (*CNNs*) are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers. They are specialized for processing data that has a know, grid-like topology like time-series or image data [7]. In my case an input is a 64×64 pixel image representing a spectrogram in RGB color model.
All the experimented CNN architectures had one input layer with *input_shape* equal to $(64, 64, 3)$, a rescaling layer in which they scale the pixels in $[0, 1]$, a certain number of Conv2D layer, each of them followed by a MaxPooling2D layer, a Flatten layer, a Dense layer and then an output layer with softmax as activation function. I adopted the "same convolution" with *kernel_size* equal to 3 and all the convolutional and dense layers had ReLu as activation function. The other settings are the same as FFNNs.
I conducted a model search from one convolutional layer upto three with 16, 32

[7]`https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectFromModel.html#sklearn.feature_selection.SelectFromModel`, December 2021

and 64 units in the first, second and third hidden layers considering also 32, 64 or 128 neurons in the Dense hidden layer.
Table 2 shows some of the results obtained with CNNs. The results are worst than the ones obtained with FFNNs and I think the reason is the fact that also spectrograms are extracted features and not the original source of information. Anyway, the best CNNs results are achieved with 3 convolutional layers and 128 hidden neurons in the dense layer, but with a standard deviation greater wrt the 2 convolutional layer architecture, which I accept considering it negligible. Figure 4 shows the bar plot of the accuracy over the different test folds of that model.
I've also tested different size for the kernel in the convolutional layers, that is

Network Architecture	Avg Accuracy	Std Accuracy
One Conv2D Layer, 128 dense neurons	0.44	0.05
Two Conv2D Layer, 128 dense neurons	0.48	0.02
Three Conv2D Layer, 128 dense neurons	0.49	0.04

Table 2: Average and Standard Deviation accuracies on the different test folds for the best CNN architectures.

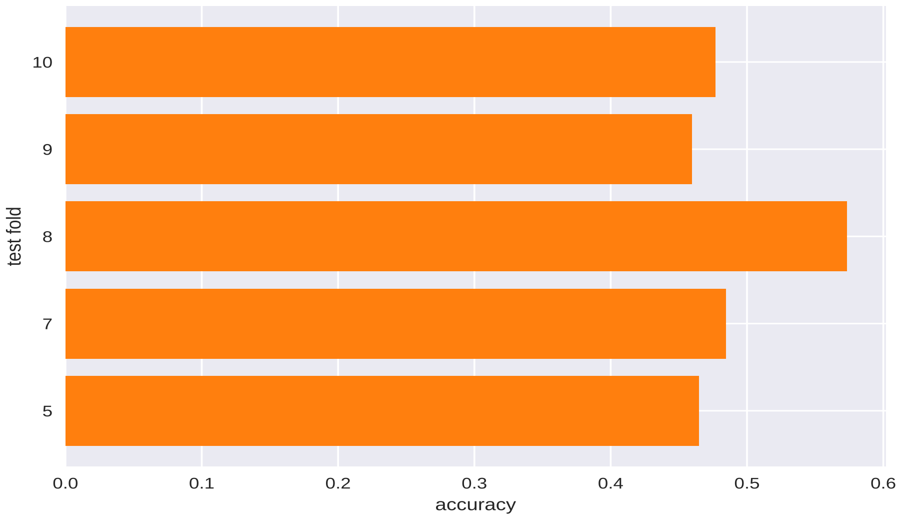

Figure 4: Bar plot of the accuracy of the best CNN (3 Conv2D layer + Dense layer with 128 neurons) on the different test folds.

the height and width of the 2D convolution window, varying from 1 to 5. Figure 5 shows the average accuracy on the test folds in function of the kernel size and it confirms that the best value is 3.

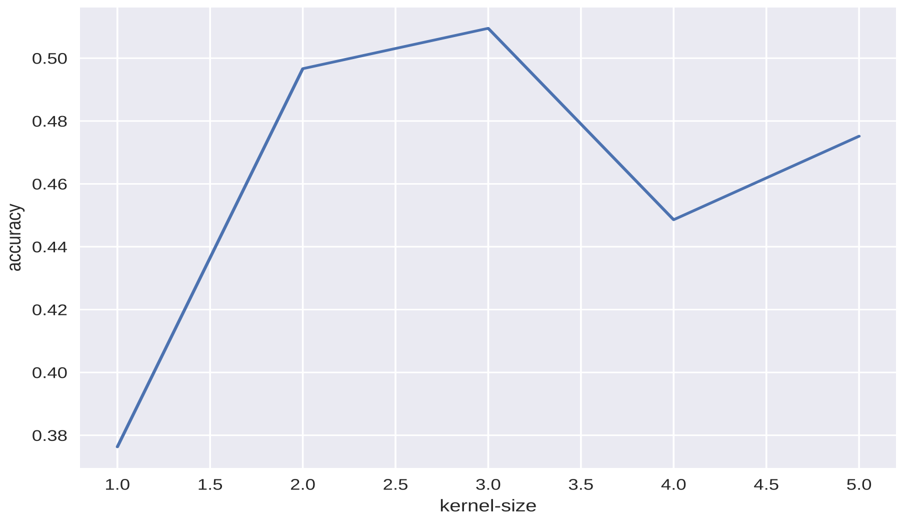

Figure 5: Average accuracy of the best CNN architecture in function of the kernel size in the convolutional layers.

2.3 Multimodal Neural Networks

A multimodal neural network is a neural network composed by different sub-models that is able to capture multimodal data, i.e. different type of input, trying to combine them to improve the results for a certain task. Information in the real world usually comes as different modalities that are characterized by different statistical properties and the scope of multimodal learning is to express the joint representations of the different modalities.
Figure 6 shows my proposed MMNN architecture for this task. There is the best CNN I found that is fed with spectograms and the best FFNN that is trained with audio features, then a layer that concatenates the last hidden layer output of the two models and finally a last dense hidden layer with 128 neurons followed by the output layer. All the not specified settings are the same of the other experiments.
Figure 7 shows the accuracy of the MMNN on the different test folds. The MMNN has an average accuracy equal to 0.59 with a standard deviation equal to 0.03. The results are very close to the ones obtained with the best FFNN. Figure 8 shows the accuracy along 40 epochs of the best FNNN, CNN and MMNN on the validation set. Validation set is obtained considering together data in all the test folds. The graph confirms how multimodal learning can be a competitive and very interesting tool to increase performance on this task.

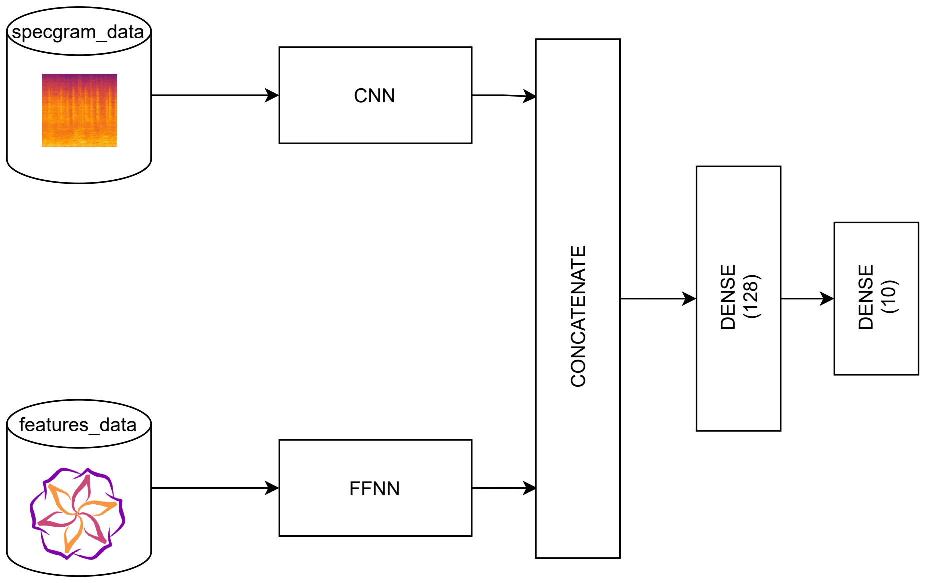

Figure 6: The proposed multimodal neural network architecture for urban sound classification.

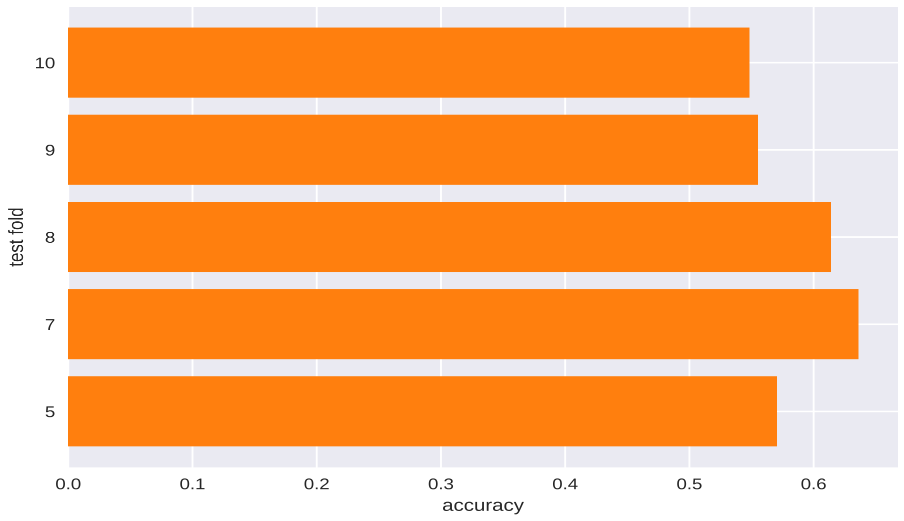

Figure 7: Bar plot of the accuracy of the MMNN on the different test folds.

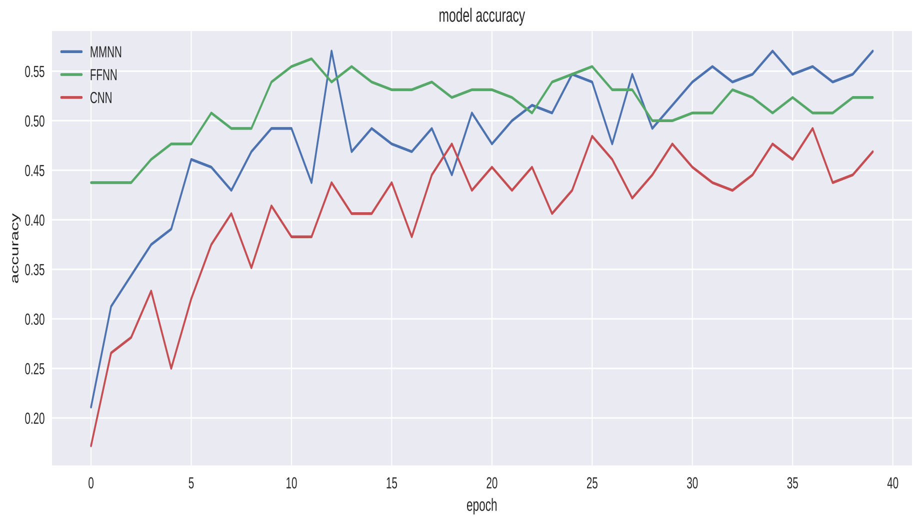

Figure 8: Accuracy in function of 40 epochs of the best CNN (red line), the best FFNN (green line), the MMNN (blue line) on the validation set.

Conclusion

I used TensorFlow 2 to train neural networks for the classification of sound events based on audio files from the UrbanSound8K dataset. The dataset contains 8732 sound excerpts ($\leq$4 seconds) of urban sounds labeled with ten different classes.
For each audio file I've extracted the Mel frequency cepstral coefficients, the chroma vector, the zero-crossing rate, the RMSE and its statistics, to construct a dataset with which I fed FFNNs, and its spectogram, saved as 64×64 pixels RGB image, to fed CNNs.
Feature selection and feature extraction techniques such as Boruta and PCA proved ineffective and worsened the performance of the models. FFNNs performs better than CNNs while MMNN can be a competitive and very interesting kind of neural network to increase performance on this task.

References

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

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore,

Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] 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.

[4] Karl Pearson F.R.S. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[5] Miron B. Kursa and Witold R. Rudnicki. Feature selection with the Boruta package. *Journal of Statistical Software*, 36(11):1–13, 2010.

[6] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

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