# Urban Sound Classification through Neural Networks

Andrea Frattini

University of Milan

## Contents

## List of Figures

**Abstract**

The aim of this project is to create a good model which enables us to classify audio files provided by the "UrbanSound8k" dataset and make predictions. In particular, for achieving this purpose, different Neural Networks will be implemented over different partitions (or splits) of the dataset, using two distinct features directly extracted from the sound files.

## 1 Introduction to the dataset

The dataset contains 8732 labeled sound excerpts (<=4s) of urban sounds, in WAV format, divided in 10 folds named fold1, fold2, ..., fold10. The UrbanSound8k files are categorized in the following 10 classes and to each of them is also assigned a numerical value that uniquely identify the corresponding class:

- air conditioner = 0
- car horn = 1
- children playing = 2

- dog bark = 3
- drilling = 4
- engine idling = 5
- gun shot = 6
- jackhammer =7
- siren =8
- street music = 9

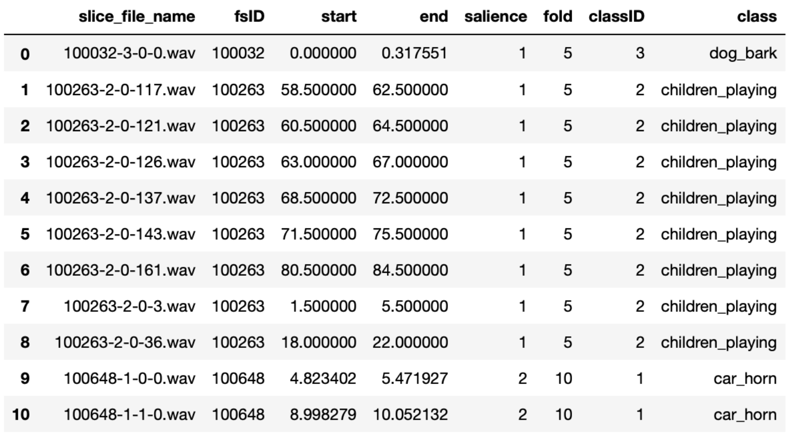
Along with the data, a meta-data CSV file is also provided, containing further informations about each audio file.

---

## 2 Pre-Processing Phase

### 2.1 Inspecting the dataset

Before starting with the analysis itself, it is essential to get an overview of the dataset's structure. Therefore, we can inspect the meta-data CSV file which allows us to find out how and with which characteristics the data has been stored:



| | slice_file_name | fsID | start | end | salience | fold | classID | class |
|---|---|---|---|---|---|---|---|---|
| 0 | 100032-3-0-0.wav | 100032 | 0.000000 | 0.317551 | 1 | 5 | 3 | dog_bark |
| 1 | 100263-2-0-117.wav | 100263 | 58.500000 | 62.500000 | 1 | 5 | 2 | children_playing |
| 2 | 100263-2-0-121.wav | 100263 | 60.500000 | 64.500000 | 1 | 5 | 2 | children_playing |
| 3 | 100263-2-0-126.wav | 100263 | 63.000000 | 67.000000 | 1 | 5 | 2 | children_playing |
| 4 | 100263-2-0-137.wav | 100263 | 68.500000 | 72.500000 | 1 | 5 | 2 | children_playing |
| 5 | 100263-2-0-143.wav | 100263 | 71.500000 | 75.500000 | 1 | 5 | 2 | children_playing |
| 6 | 100263-2-0-161.wav | 100263 | 80.500000 | 84.500000 | 1 | 5 | 2 | children_playing |
| 7 | 100263-2-0-3.wav | 100263 | 1.500000 | 5.500000 | 1 | 5 | 2 | children_playing |
| 8 | 100263-2-0-36.wav | 100263 | 18.000000 | 22.000000 | 1 | 5 | 2 | children_playing |
| 9 | 100648-1-0-0.wav | 100648 | 4.823402 | 5.471927 | 2 | 10 | 1 | car_horn |
| 10 | 100648-1-1-0.wav | 100648 | 8.998279 | 10.052132 | 2 | 10 | 1 | car_horn |

Figure 2.1: Structure of the dataset

As we can see, each row provides us a lot of informations about an audio file even if, for the purposes of our analysis, we are only interested in the following ones: *slice file name*, *fold*, *classID*, *class*.
In particular, the shape of a WAV file (represented by the figure 2.2) could be a good starting point for identifying some important characteristics (e.g. spikes) that could help us the classification of such sounds.

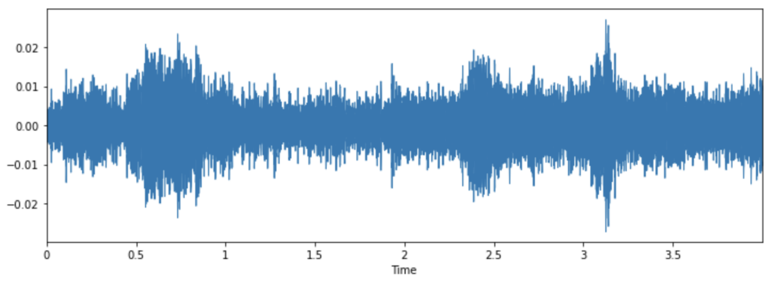


Figure 2.2: Wave plot of audio file 100263-2-0-117.wav

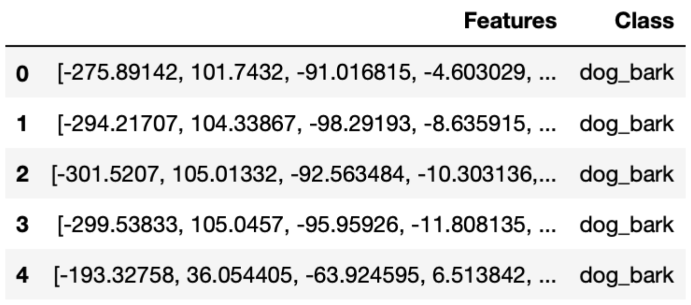
The problem with audio files is that, visually, they all look like this one even if, sometimes, they have some specific spikes (like the one around 3.1 seconds) or other spikes not so clear (like the one between 0.5 and 0.8). However, these spikes can be caused by different actions: focusing on the spike at 3.1 sec, it could be caused either by a gun shot in a crowded street or a basketball bouncing on the court in a park full of playing kids.
So, if we would train the model through rows, then it wouldn't be able to detect the different causes of similar spikes. Therefore, it would probably be better to extract some more specific features from the audio files and work with them.

## 2.2 Extracting different features

First, we need to identify some specific features that could help us in the classification of sound files and, then, we also need to extract them from our data. The most common features used in audio file classification tasks are the *MFCCs* and the *Chromagram*.

### 2.2.1 Mel-Frequency Cepstral Coefficients (MFCC)

MFCCs are commonly used in environmental sound analysis: the features are extracted on a per-frame basis using a window size of 23.2 ms and 50% frame overlap. Then it's computed a 40 Mel bands between 0 and 22050 Hz and are kept the first 25 MFCC coefficients. The per-frame values for each coefficient are summarized across time using the following summary statistics: minimum, maximum, median, mean, variance, skewness, kurtosis and the mean and variance of the first and second derivatives, resulting in a feature vector of dimension 225 per slice.
After the extraction of the *MFCCs*, the data are aggregated in a data frame and the result is the following:



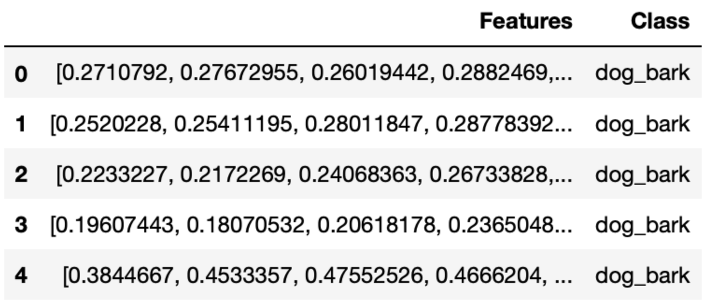
| | Features | Class |
|---|---|---|
| 0 | [-275.89142, 101.7432, -91.016815, -4.603029, ... | dog_bark |
| 1 | [-294.21707, 104.33867, -98.29193, -8.635915, ... | dog_bark |
| 2 | [-301.5207, 105.01332, -92.563484, -10.303136,... | dog_bark |
| 3 | [-299.53833, 105.0457, -95.95926, -11.808135, ... | dog_bark |
| 4 | [-193.32758, 36.054405, -63.924595, 6.513842, ... | dog_bark |

Figure 2.3: Example of the dataframe containing the MFCCs and their class

Basically, in the "Feature" column we have one array per row containing in a list the values of the aforementioned statistics of an audio file, while in the column "Class" is stored the label (e.i. the class) of the corresponding file.

### 2.2.2 Chromagram

Chroma-based features, which are also referred to as "pitch class profiles", are a powerful tool for analyzing sound files whose pitches can be meaningfully categorized and whose tuning approximates to the equal-tempered scale. One main property of chroma features is that they capture harmonic and melodic characteristics of sounds (generally music), while being robust to changes in timbre and instrumentation.
After the extraction, the data are, once again, aggregated in a data frame and the result looks like in figure 2.4:

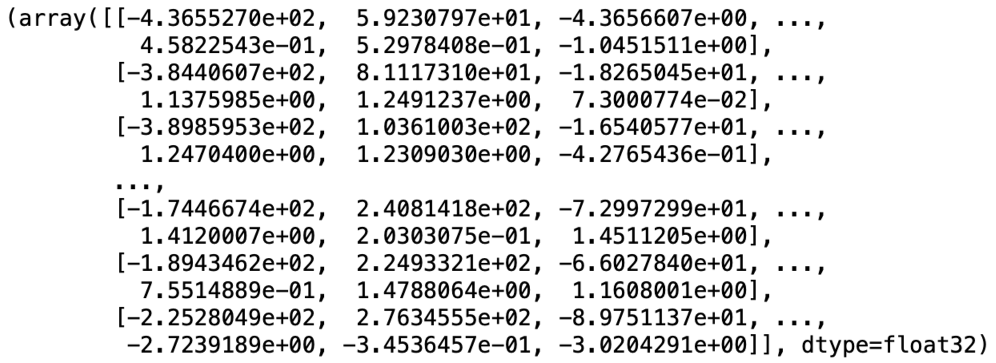


| | Features | Class |
|---|---|---|
| **0** | [0.2710792, 0.27672955, 0.26019442, 0.2882469,... | dog_bark |
| **1** | [0.2520228, 0.25411195, 0.28011847, 0.28778392... | dog_bark |
| **2** | [0.2233227, 0.2172269, 0.24068363, 0.26733828,... | dog_bark |
| **3** | [0.19607443, 0.18070532, 0.20618178, 0.2365048... | dog_bark |
| **4** | [0.3844667, 0.4533357, 0.47552526, 0.4666204, ... | dog_bark |

Figure 2.4: Example of the dataframe containing the chroma features and the class

### 2.2.3 Preparing the data

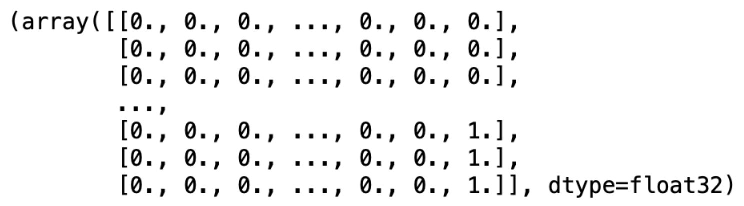
After having extracted the two different features it's crucial to prepare the data for being analyzed. First, we need to separate the values of the features ($X$) from the labels ($Y$). Since the latter are categorical, we must transform them into binary variables through the *One-Hot Econding* technique, creating as many dummy variables as there are categories and, for each row, setting to 1 the dummy corresponding the category to which the file belongs and 0 the others. In this way, the $X$ variables will look like, for example, in fig. 2.5



```
(array([[-4.3655270e+02,  5.9230797e+01, -4.3656607e+00, ...,
          4.5822543e-01,  5.2978408e-01, -1.0451511e+00],
        [-3.8440607e+02,  8.1117310e+01, -1.8265045e+01, ...,
          1.1375985e+00,  1.2491237e+00,  7.3000774e-02],
        [-3.8985953e+02,  1.0361003e+02, -1.6540577e+01, ...,
          1.2470400e+00,  1.2309030e+00, -4.2765436e-01],
        ...,
        [-1.7446674e+02,  2.4081418e+02, -7.2997299e+01, ...,
          1.4120007e+00,  2.0303075e-01,  1.4511205e+00],
        [-1.8943462e+02,  2.2493321e+02, -6.6027840e+01, ...,
          7.5514889e-01,  1.4788064e+00,  1.1608001e+00],
        [-2.2528049e+02,  2.7634555e+02, -8.9751137e+01, ...,
         -2.7239189e+00, -3.4536457e-01, -3.0204291e+00]], dtype=float32)
```

Figure 2.5: Example of the structure of the X data for MFCCs

While the dependent variables ($Y$) look like:



```
(array([[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 1.],
        [0., 0., 0., ..., 0., 0., 1.],
        [0., 0., 0., ..., 0., 0., 1.]], dtype=float32)
```

Figure 2.6: Example of the structure of the Y data for MFCCs

This process has been performed twice, once for each feature.
Now, we are ready to start with the analysis.

### 2.3 Different training and test set sizes

For a complete and deep analysis, we not only will implement different models over different features, but we will also train and test them using two different sizes of the homonymous sets:

- First, we will train our models over the folds number 1, 2, 3, 4 and 6 and test them over folds number 5, 7, 8, 9 and 10.
- Secondly, we will train our models over a training set composed by the 80% of the entire dataset and we will test them over the remaining 20% of the dataset.

In this way, we will be able to detect and understand if and how our models will respond in a different way with respect to different training set sizes.

# 3 Analysis

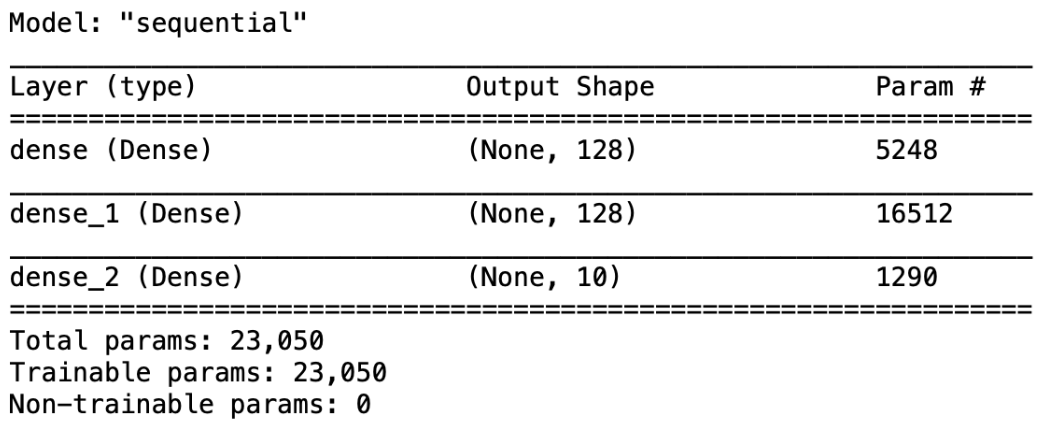
### 3.1 Structure of the Neural Networks

Understand which could be the best neural network for classifying and predicting such audio files it's not an easy task since the common procedure in these cases is to implement NNs with different architectures and then empirically test them over the data. Thus, for sake of completeness, in this paper we will implement 5 different models: they all use the *ReLu* activation function, which is the most used for hidden layers since it's simple to implement and effective at overcoming the limitations of other popular activation

functions, such as for example Sigmoid and Tanh. In particular, the *ReLu* crushes the input values into the interval $[0; max(x)]$ since it is calculated as $max(0, x)$ and then if the input value ($x$) is negative then a zero is returned, otherwise, the value of $x$ is returned. Moreover, for the output layers of all the models is used the *softmax* activation function, due to the issue of results' interpretability in multi-classification problems. By definition, the *softmax* activation will output one value for each node in the output layer. The output values will represent (or can be interpreted as) probabilities and the values sum to 1. The error between the expected and predicted multinomial probability distribution is often calculated using cross-entropy, and this error is then used to update the model: this is called the cross-entropy loss function.
All these neural networks are shown in detail in the following paragraphs.

### 3.1.1 Model 1

We start with the simplest NN, which is composed by only one hidden layer of 128 neurons.

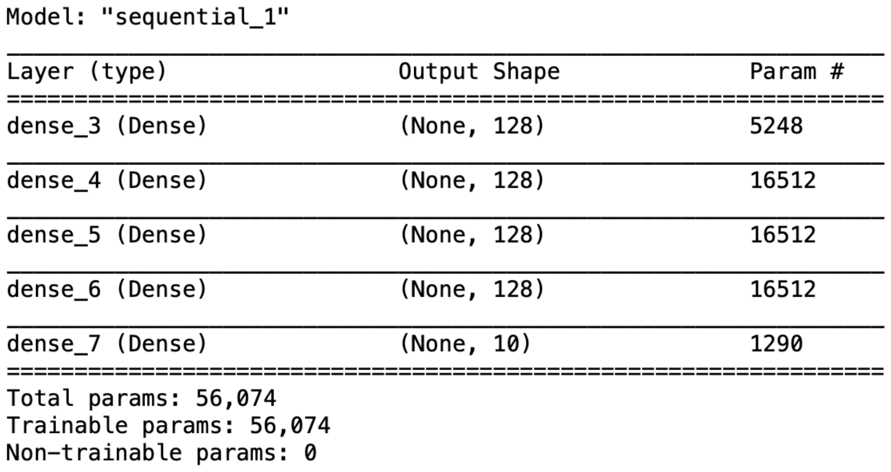


```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 128)               5248
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 23,050
Trainable params: 23,050
Non-trainable params: 0
```

Figure 3.1: Model 1

### 3.1.2 Model 1.2

The second NN is similar to the first model but it has more hidden layers (for a total of 3) each of which having 128 neurons.

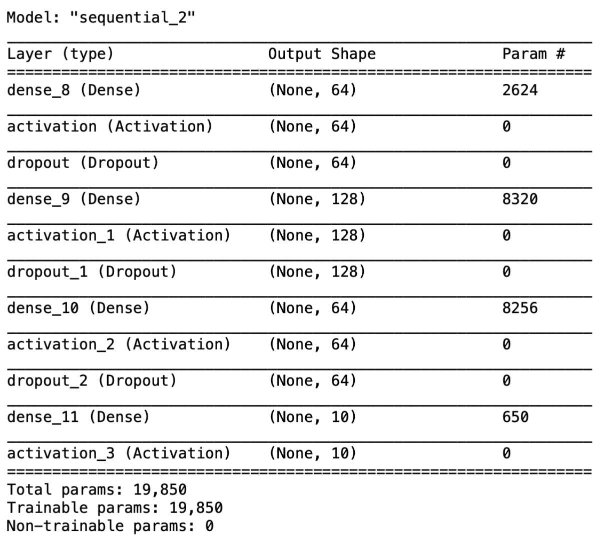


```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 128)               5248
_________________________________________________________________
dense_4 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_5 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_6 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_7 (Dense)              (None, 10)                1290
=================================================================
Total params: 56,074
Trainable params: 56,074
Non-trainable params: 0
```

Figure 3.2: Model 1.2

### 3.1.3 Model 2

The Model 2 is the medium one, having two hidden layers and a combination of different number of neurons per layer. From here on, to all the models will be applied a regularization technique in order to try to keep the overfitting problem under control. Thus, to do that, a Dropout rate of 0.3 will be added to each layer. *Dropout*, applied to a layer, consists of randomly ”dropping out” (i.e. set to zero) a number of output features of the layer during training. For instance, let’s assume that a given layer would normally have returned a vector $[1, 2, 3, 4, 5]$ for a given input sample during training; after applying dropout, this vector will have a few zero entries distributed at random, e.g. $[0, 2, 3, 0, 5]$. The ”dropout rate” is the fraction of the features that are being zeroed-out, which in usually set between 0.2 and 0.5. At test time, no units are dropped out, and instead the layer’s output values are scaled down by a factor equal to the dropout rate, so as to balance for the fact that more units are active than at training time. It’s also important to point out that this type of regularization usually improves the performance of large models. Therefore we expect the performance to improve as the models get bigger and bigger.
In particular, Model 2 is structured as follows:

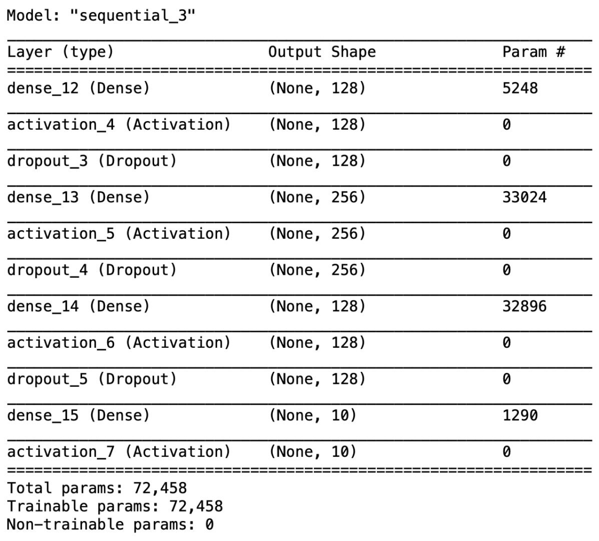


```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 64)                2624
_________________________________________________________________
activation (Activation)      (None, 64)                0
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_9 (Dense)              (None, 128)               8320
_________________________________________________________________
activation_1 (Activation)    (None, 128)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_10 (Dense)             (None, 64)                8256
_________________________________________________________________
activation_2 (Activation)    (None, 64)                0
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_11 (Dense)             (None, 10)                650
_________________________________________________________________
activation_3 (Activation)    (None, 10)                0
=================================================================
Total params: 19,850
Trainable params: 19,850
Non-trainable params: 0
```

Figure 3.3: Model 2

### 3.1.4 Model 2.2

This model is similar to Model 2 with the same number of hidden layers but, this time, each of them has a higher number of neurons.

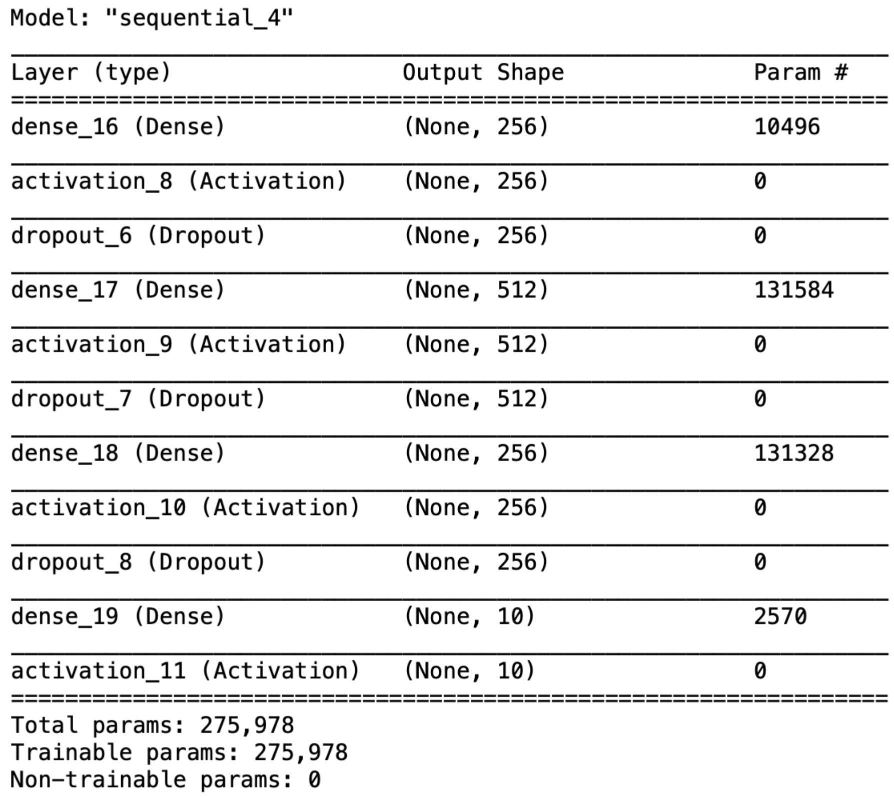


```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_12 (Dense)             (None, 128)               5248
_________________________________________________________________
activation_4 (Activation)    (None, 128)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_13 (Dense)             (None, 256)               33024
_________________________________________________________________
activation_5 (Activation)    (None, 256)               0
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_14 (Dense)             (None, 128)               32896
_________________________________________________________________
activation_6 (Activation)    (None, 128)               0
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_15 (Dense)             (None, 10)                1290
_________________________________________________________________
activation_7 (Activation)    (None, 10)                0
=================================================================
Total params: 72,458
Trainable params: 72,458
Non-trainable params: 0
```

Figure 3.4: Model 2.2

### 3.1.5 Model 3

It is the biggest one, with again two hidden layers but with a higher number of neurons per layer with respect to the previous one.

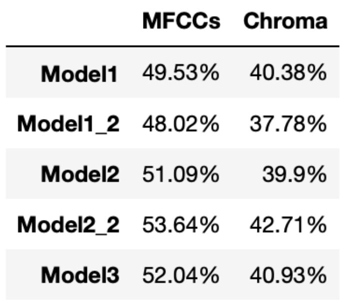


```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_16 (Dense)             (None, 256)               10496
_________________________________________________________________
activation_8 (Activation)    (None, 256)               0
_________________________________________________________________
dropout_6 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_17 (Dense)             (None, 512)               131584
_________________________________________________________________
activation_9 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_7 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_18 (Dense)             (None, 256)               131328
_________________________________________________________________
activation_10 (Activation)   (None, 256)               0
_________________________________________________________________
dropout_8 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_19 (Dense)             (None, 10)                2570
_________________________________________________________________
activation_11 (Activation)   (None, 10)                0
=================================================================
Total params: 275,978
Trainable params: 275,978
Non-trainable params: 0
```

Figure 3.5: Model 3

## 3.2 Training and Testing over the folds

First, we train our five models over the folds number 1, 2, 3, 4 and 6 for both the extracted features MFCCs and Chromagram and we also test them over the folds 5, 7, 8, 9, and 10. Then, we calculate the average accuracy of each neural network tested on each fold for each of the two features, getting the following result:



|  | MFCCs | Chroma |
|---|---|---|
| **Model1** | 49.53% | 40.38% |
| **Model1_2** | 48.02% | 37.78% |
| **Model2** | 51.09% | 39.9% |
| **Model2_2** | 53.64% | 42.71% |
| **Model3** | 52.04% | 40.93% |

Figure 3.6: Average accuracy of each model for each feature

As we can see from figure 3.6 the Model 2.2 has the highest value of average accuracy for both the features. By the way, the accuracy is equal only to, respectively, 53.64% and 42.71% for the MFCCs and Chromagram, which are relatively low values: they means that, when we extract the MFCCs feature from an audio file and we use Model 2.2 (which we have seen being the best one in terms of performance) to classify and predict data, it only correctly outputs (on average) the half of the results, while extracting the Chromagram and implementing again Model 2.2 we get an average of only 4 correct output over 10.
Moreover, it could be interesting also to view the behavior of Model 2.2 (for each feature, as always) in terms of training and validation accuracy, to detect if our model is subject to underfit or overfit issues.

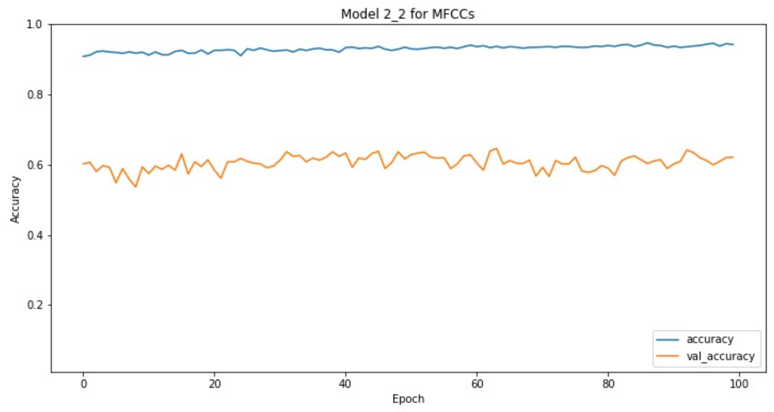


Figure 3.7: History of Model 2.2 for the MFCCs

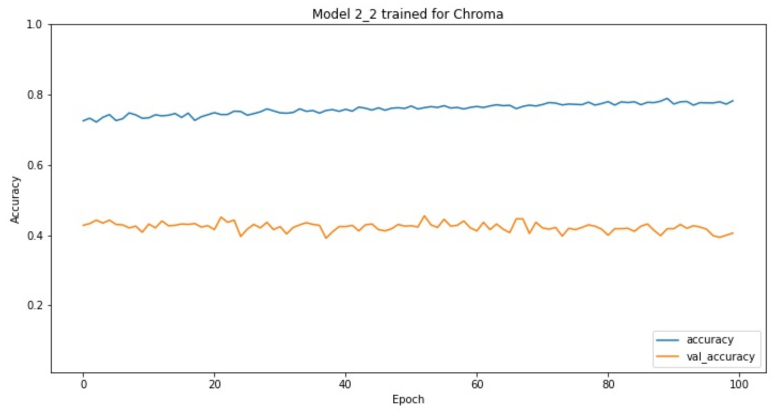
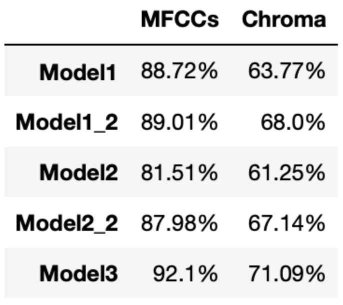


Figure 3.8: History of Model 2.2 for the Chromagram

As shown in figures 3.7 and 3.8, in both the situations our Model 2.2 is overfitting because the accuracy measured against the training set in very good while the accuracy measured against a validation set (in this case we have used, respectively, fold7 and fold9) is definitely not as good. This result could be either due to the limited number of data present in the training set with respect to the number of parameters of the model, causing the fact that the neural network is memorizing the training data instead of learning the more-general mapping from features to labels, or, due to a large difference between the data in the test set with respect to the data in the training set. Thus, we could have two main solutions for this unpleasant situation: we could reduce the complexity of the model (this solution has been tested during the experimental phase, leading to even worse results than the one shown above) or increase the size of the training set.
Moreover, Model 2 tested over all the testing folds yields to the same overfitting problem, excluding the possibility that this issue could be due to an excessive data diversity between training and test sets.

## 3.3 Increasing the training set size

Now, instead of using the pre-specified folds, we join them into a one, big, dataset for then splitting it into a training set (composed by the 80% of the entire dataset) and a test set (composed by the remaining 20%). Then, we train again all our models over this new training set, we test them using the remaining 20% of the data and we calculate as usual the average accuracy for each extracted feature. The result of these analyses is shown in a compact way by figure 3.9, which allows us to interpret the meaning is an easier way.



| | MFCCs | Chroma |
|---|---|---|
| **Model1** | 88.72% | 63.77% |
| **Model1_2** | 89.01% | 68.0% |
| **Model2** | 81.51% | 61.25% |
| **Model2_2** | 87.98% | 67.14% |
| **Model3** | 92.1% | 71.09% |

Figure 3.9: Average accuracy of each model for each feature

As we can see, this time the model with the highest average accuracy is Model 3 (the most complex one) for both MFCCs and Chromagram, with a test accuracy of respectively 92.1% and 71.09%. As usual, we focus on this model since it seems to be the best one, and we inspect the relation between the training and validation accuracy over the epochs for both the features.

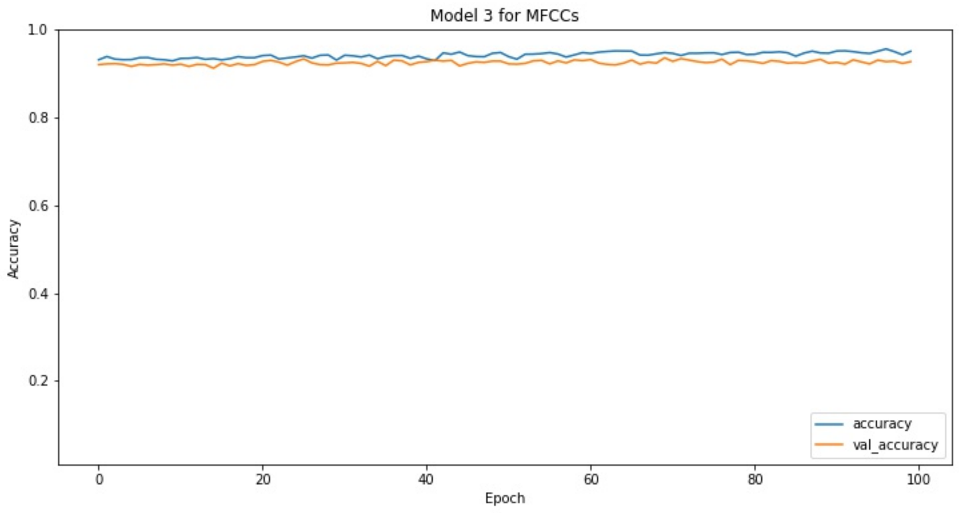


Figure 3.10: History of Model 3 for the MFCCs

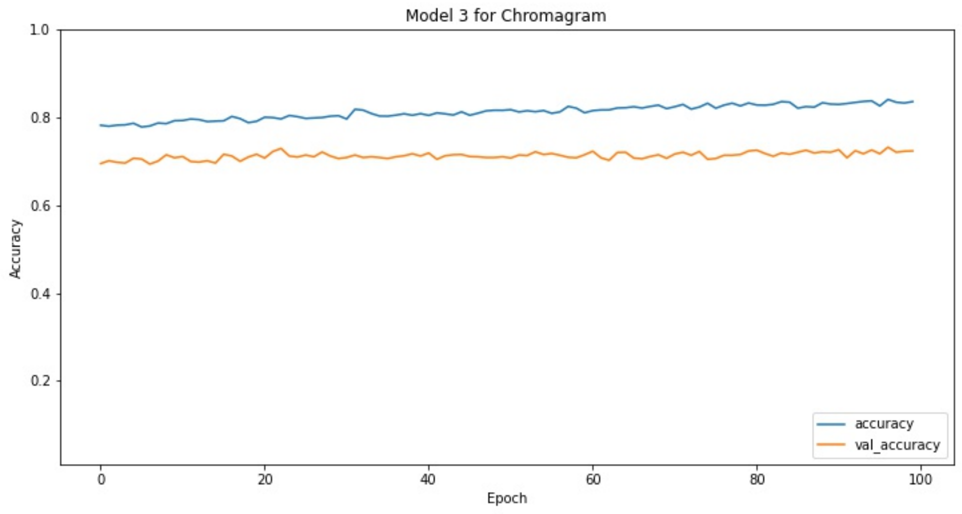


Figure 3.11: History of Model 3 for the Chromagram

As we can see from the above figures, the same neural network (Model 3) performs much better when the MFCC feature is extracted: the first obvious reason is that, as already pointed out, the validation accuracy is the 20% better with a value of 92.1% against the 71.09% of the Chromagram; the second important reason is that, as clearly shown in figure 3.10, the model is not overfitting, while in fig. 3.11 it definitely does.

## 4 Results

Throughout our analysis we have implemented five different neural networks over different training and test set sizes for two extracted features, namely the MFCC and the Chromagram. First, we have seen that using folds 1, 2, 3, 4 and 6 as training set and the remaining folds as test set led us to have Model 2.2 as the best one for both the features. Then, we have performed the same process over a different partition of the dataset, using the 80% of it for the training phase and the remaining 20% for the testing one, getting Model 3 as the best, again for both MFCC and Chromagram.
Now, it should be time to declare which one is actually the best neural

network for classifying and predicting audio files and, in order to that, we give a look to the history of our two candidates (for both the features):

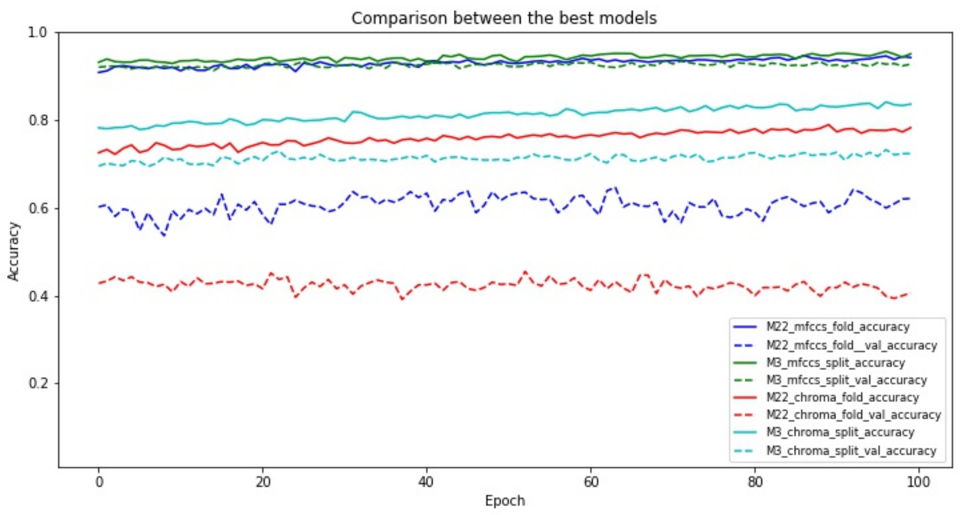
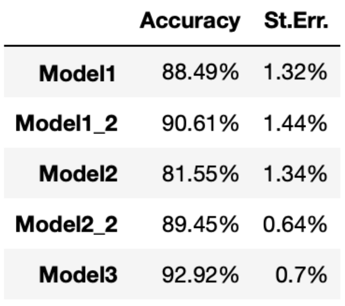


Figure 4.1: History of candidate models per feature

Again, it seems absolutely clear that Model 3 trained over the 80% of the dataset after having extracted the MFCC feature is the best one, thanks to the highest value of validation accuracy and the absence of overfitting. However, one must be careful before making such a statement: a couple more observations need to be made.
The accuracy values provided in figure 3.9 result from a single split of the dataset (into training and test set) on which the models have been trained and tested: this could be misleading. In fact, such high accuracy values could result from a similarity of the test data with the training ones: we need to be sure that our results are not due to a situation like this.

## 4.1 Validating the result

Therefore, it's essential to validate our result performing different splits of the dataset into training and test sets, maintaining the usual proportion of 80% and 20% respectively, for excluding the possibility that our result is provided by a particular partition of the dataset. For achieving this purpose, we perform a 10-fold cross-validation for all the models but, this time, only on the MFCC feature getting the following result:



| | Accuracy | St.Err. |
|---|---|---|
| **Model1** | 88.49% | 1.32% |
| **Model1_2** | 90.61% | 1.44% |
| **Model2** | 81.55% | 1.34% |
| **Model2_2** | 89.45% | 0.64% |
| **Model3** | 92.92% | 0.7% |

Figure 4.2: 10-fold cross-validation results

## 5 Conclusion

The first thing to notice from our analysis is that it doesn't matter what the dataset partition is, in all our results the feature MFCC outclasses the Chromagram and, therefore, it's rather easy to infer that it is the most characterizing feature for audio files, at least from a classification task point of view. In particular, Chromagram has the peculiarity of finding out a bunch of musical notes which mostly characterize a given sound and, in our case, we don't have categories of sounds close to the definition of "music" but, rather, we have something which is more similar to "noise": that's probably why the Chromagram is not the best feature for our purposes.
That's the reason why the 10-fold cross-validation of figure 4.2 has been done only for the MFCC feature.
Fortunately, the CV has confirmed our suspects, providing us Model 3 as the best one with a validation accuracy of 92.92% and a standard error of 0.7%, which are really good percentages.
Therefore, we can finally establishes that the neural network called "Model 3", trained on the 80% of the dataset after having extracted the MFCC feature is the best model among all our starting ones, because it provides the highest average validation accuracy, the lowest standard error and it is not subject to overfitting.

But why Model 3 and why with this data partition? Well, the main idea is the following: the higher the data points in the training set the more efficient the neural network will be and, the more data samples we have the more layers and nodes we can add up, with the result of having better performances. Thus, with this neural network we are able to make predictions with an average accuracy of, more or less, 92.92%.

## 5.1 Declaration

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.

## 5.2 Bibliography

- Lecture notes
- tensorflow.org