Urban Sound Classification

Machine learning, statistical learning, deep learning and artificial intelligence

January 2022

Abstract: Automatic urban sound classification is a growing area of research with applications in several fields of our everyday life. In this paper, Tensorflow2 will be used in order to train neural networks for the classification of sound events based on audio files from the UrbanSound8K dataset. Two neural networks architectures for multi-class classification will be implemented: the multilayer perceptron model and the convolutional neural network. Furthermore, several features extraction methodologies implemented in the librosa library will be examined: in particular, the MFCCs and the STFT Chromagram will be chosen. The different results obtained will be presented and will be compared to each other.

Keywords: Neural networks, urban sound, multi-class classification

Martina Corsini ID 944506 Camilla Gotta ID 945522
Università degli Studi di Milano
Data Science and Economics

Contents

List of Figures

List of Tables

1 Introduction

Automatic urban sound classification is a growing area of research with applications in multimedia retrieval and urban informatics [1]. In this paper, we will use Tensorflow2 in order to train neural networks for the classification of sound events based on audio files from the UrbanSound8K dataset.
Neural networks can be fed with raw data or proper features can be extracted. We will use the librosa library, a Python package for music and audio analysis, to extract and compare several useful features that are commonly used in the field of sound events classification. Specifically, we will focus on three kind of representation of an audio signal: The waveplots, the basic representation of an audio signal in the time domain; the spectrograms, useful techniques for visualizing the frequency spectrum of a sound; the chromagrams, namely the transformation of a signal's time-frequency properties into a temporally varying precursor of pitch. In our analysis, we will use two features extraction methodologies: the Mel-frequency cepstral coefficients (MFCCs) and the Short time Fourier transformation (STFT) chromagram.
Going further, two different artificial neural network architectures will be shown so to explain how they work: the multilayer perceptrons and the convolutional neural networks. Furhtermore, their training parameters will be discussed in order to document their influence on the final predictive performance.
Finally, after a brief description of the dataset, the findings will be presented and discussed.

2 Theoretical background

2.1 Feature extraction methodologies

In the task of working with audio data, one of the biggest challenges is to prepare an audio file in order to make it understandable by the machine [2]. We can consider audio files in time domain and frequency domain (Fig. 1). When we are considering an audio signal in the time domain, the independent variable is time and the value of the Y-axis depends on the changing of the signal with respect to time [3]. In contrast, the frequency domain, that is more used in examining audio signals, considers the frequency of the signal as the independent variable. Its graph shows how much of the signal lies in each frequency band over a range of frequencies [2].

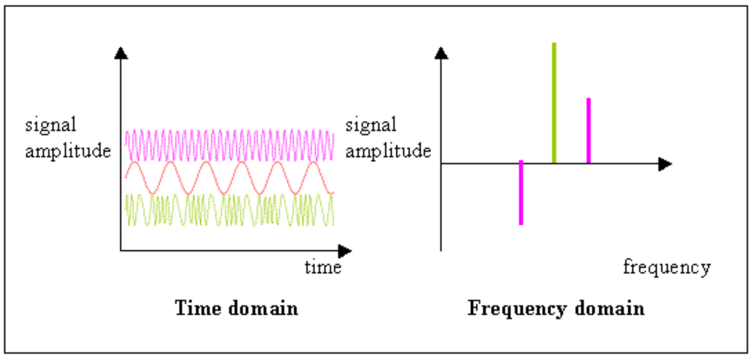


Figure 1: Time domain - Frequency domain [4]

Thanks to Librosa library [5], a Python package for music and audio analysis, it is possible to load the audio files into the machine so to make them readable by the machine itself: it simply takes values after every specific time step [6].
In the next paragraphs, we will see an overview of the various features available in the audio data framework and how these features can be useful for different aspects of speech and audio analysis.

2.1.1 Waveplot

The waveplot is the basic representation of an audio signal in the time domain (Fig. 2). It shows the loudness of sound wave changing with time. In this framework, amplitude = 0 represents silence. Since these amplitudes are not very informative, it is necessary to transform them into the frequency-domain in order to better understand the audio signal. Fourier transform is a mathematical concept that can convert a continuous signal from time-domain to frequency-domain [7].

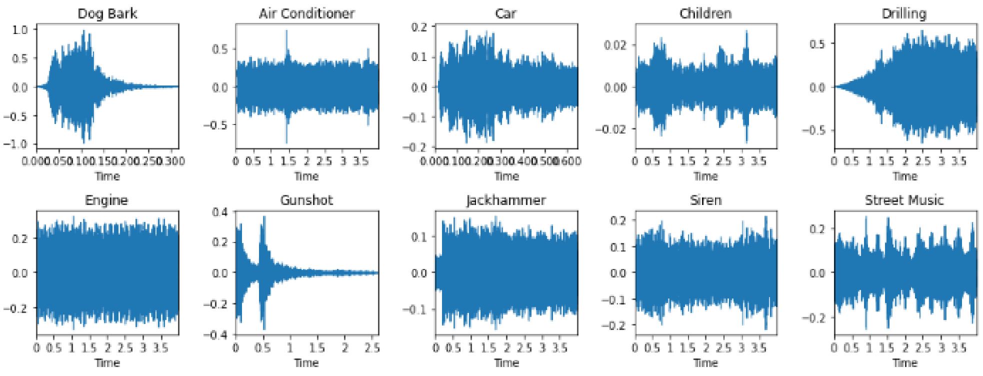


Figure 2: Waveplot

2.1.2 Spectrograms

Spectrograms are useful techniques for visualizing the frequency spectrum of a sound and how it varies during a very short period of time. A spectrogram uses a linearly spaced frequency scale [8]. A mel spectrogram (Fig.3) is a spectrogram in which frequencies are converted to the mel scale. The mel scale is a logarithmic transformation of the frequency of a signal [9]. The central idea of this transformation is that sounds of equal distance on the mel scale are perceived as being of equal distance by humans, since it is much harder for humans to be able to distinguish between higher frequencies and easier for lower frequencies [10].

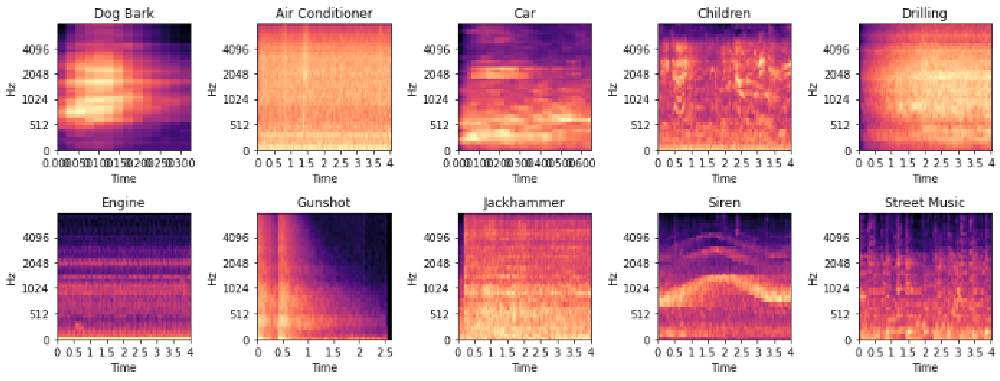


Figure 3: Mel Spectrogram

In our anaysis, we will use a similar technique known as Mel-Frequency Cepstral Coefficients (Fig. 4). It is based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency.
MFCCs are commonly derived as follows [11]:

- Take the Fourier transform of (a windowed excerpt of) a signal.
- Map the powers of the spectrum obtained above onto the mel scale, using triangular overlapping windows or alternatively, cosine overlapping windows.
- Take the logs of the powers at each of the mel frequencies.
- Take the discrete cosine transform of the list of mel log powers, as if it were a signal.
- The MFCCs are the amplitudes of the resulting spectrum.

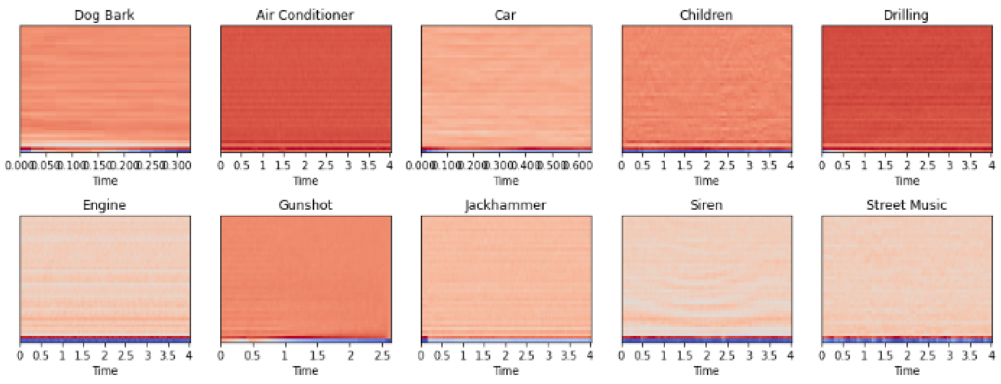


Figure 4: Mel frequency cepstral coefficients

2.1.3 Chromagrams

The chromagram is a transformation of a signal's time-frequency properties into a temporally varying precursor of pitch [12]. A pitch is the property of any sound or signal which allows us to order sounds on frequency-related scale, helping us in judging the sound as higher, lower, and medium in the sense associated with musical melodies. Therefore, chroma features capture harmonic and melodic characteristics of music, while being robust to changes in timbre and instrumentation. We have 12 pitches, that is, all the notes plus the minor/major keys regardless of the octave they are in. There are three types of chromagrams [13]:

- Short time Fourier transformation chromagram (Fig. 5): it is basically a chromagram that is computed from the power spectrogram of the audio files.

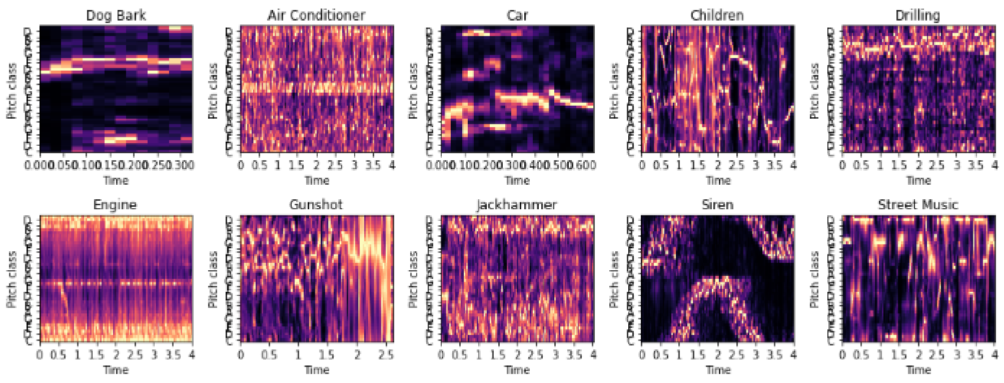


Figure 5: Short Time Fourier transformation chromagram

- Constant-Q chromagram (Fig. 6): it is a chromagram of constant-Q transform signal. This transformation of the signal takes part in the frequency domain and it is related to the Fourier Transform and Morlet Wavelet Transform.

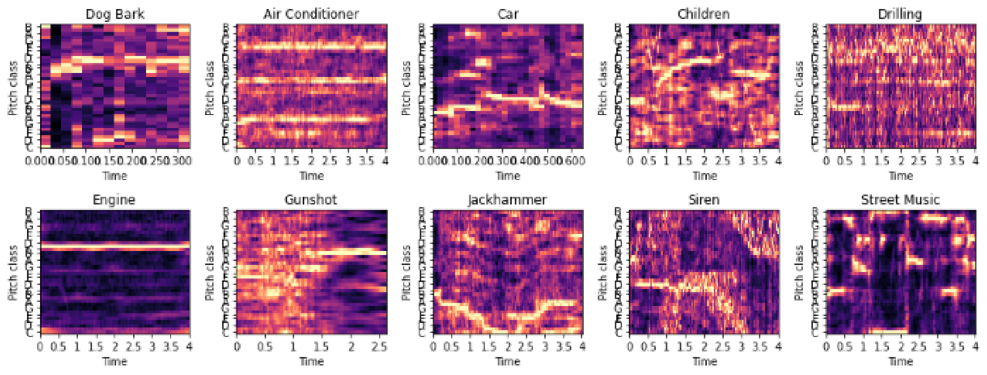


Figure 6: Constant-Q chromagram

- Chroma energy normalized statistics (CENS) chromagram (Fig. 7): as the name suggests, this chromagram is made up of signals energy form where furthermore transformation of pitch class by considering short time statistics over energy distribution within the chroma bands helps in obtaining CENS(Chroma energy normalized statistics). The CENS chromagram introduces additional post processing steps on the constant-Q chromagram to obtain features that are invariant to dynamics and timbre [14].

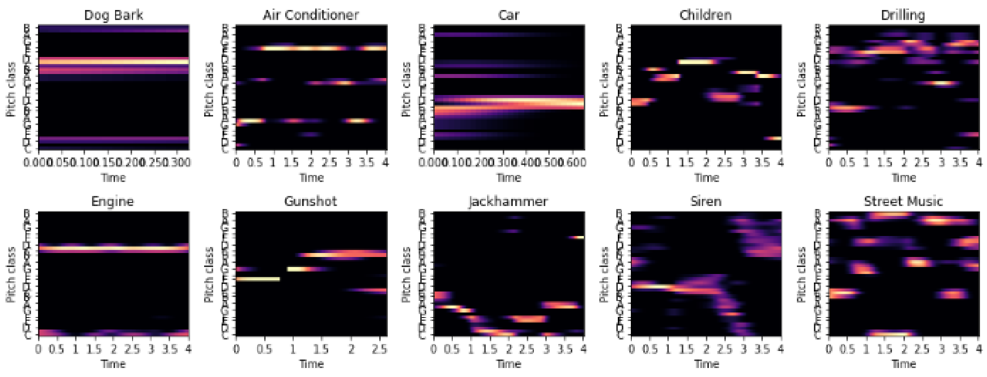


Figure 7: Chroma energy normalized statistics chromagram

2.2 Neural networks

Artificial neural networks are Machine Learning models that are very useful and used since they are versatile, powerful, and scalable. In this paper we will consider 2 kind of NNs: Multilayer perceptrons and convolutional neural networks. For this purpose, in this section we will follow the Geron textbook [15].

2.2.1 Multilayer perceptrons

The Perceptron is one of the simplest ANN architectures. It is based on an artificial neuron called a threshold logic unit (Fig. 8): the inputs and output are numbers and each input connection is associated with a weight.

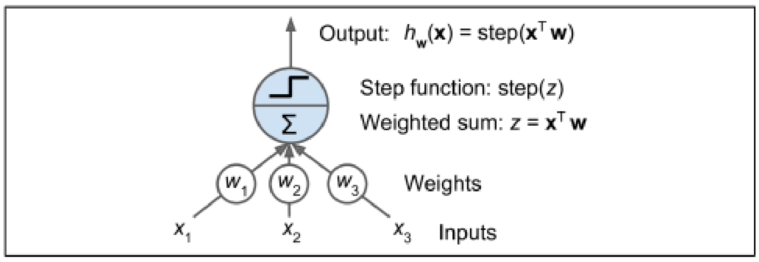

Output: $h_w(\mathbf{x}) = \text{step}(\mathbf{x}^T \mathbf{w})$
Step function: step(z)
Weighted sum: $z = \mathbf{x}^T \mathbf{w}$
w_1
w_2
w_3
Weights
x_1
x_2
x_3
Inputs

Figure 8: Threshold logic unit [15]

The TLU computes a weighted sum of its inputs

$$z = w_1x_1 + w_2x_2 + ... + w_nx_n = x^Tw$$

then applies a step function to that sum and outputs the result:

$$h_w(x) = step\,(z), \text{ where } z = x^Tw.$$

A Perceptron (Fig. 9) is simply composed of a single layer of TLUs, with each TLU connected to all the inputs.

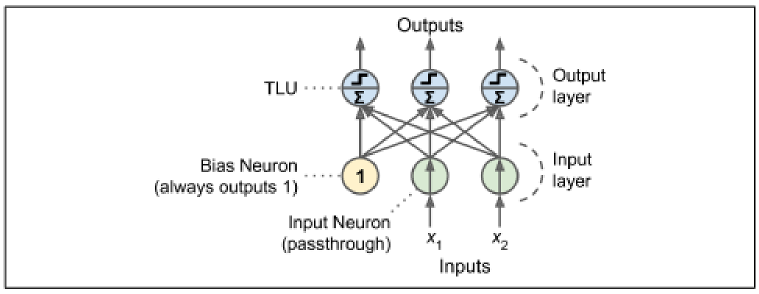


Figure 9: Perceptron [15]

When all the neurons in a layer are connected to every neuron in the previous layer (i.e., its input neurons), it is called a dense layer. To represent the fact that each input is sent to every TLU, it is common to draw special pass-through neurons called input neurons: they just output whatever input they are fed. Moreover, an extra bias feature is generally added ($x_0 = 1$): it is typically represented using a special type of neuron called a bias neuron, which just outputs 1 all the time.
A Multilayer perceptron (Fig. 10) is composed of one (pass-through) input layer (with one input neuron per input feature), one or more layers of TLUs, called hidden layers (typically between 1 and 5) with a number of neurons typically set between 10 and 100, and one final layer of TLUs called the output layer (with one output neuron per class). Every layer except the output layer includes a bias neuron and is fully connected to the next layer. The number of layers and the number of neurons per hidden layer must be chosen wisely as a very high number may introduce problems like over-fitting and a lower number may cause a model to have high bias and low potential model [16]. The optimal number depends on the data size used for training.
To train MLPs in an efficient way the back-propagation algorithm was created and it is simply a Gradient Descent using an efficient technique for computing the gradients automatically: in just two passes through the network (one forward, one backward), it is able to compute the gradient of the network's error with regards to every single model parameter. In other words, it can find out how each connection weight and each bias term should be tweaked in order to reduce the error. Once it has these gradients, it just performs a regular Gradient Descent step, and the whole process is repeated until the network converges to the solution.

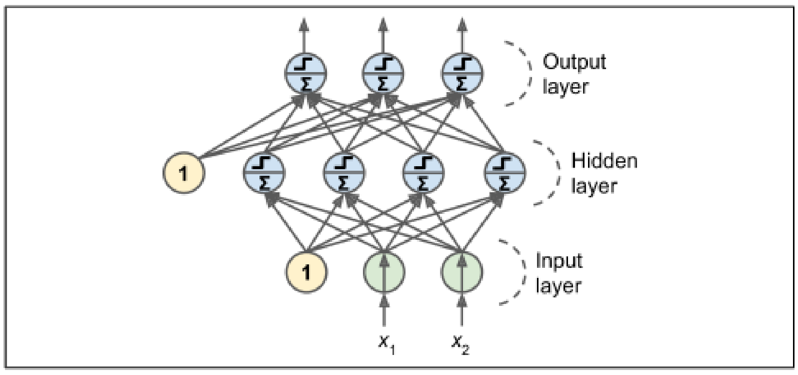


Figure 10: Multilayer perceptron [15]

Activation function for hidden layer In a neural network, the activation function of a node defines the output of that node given an input or set of inputs: it is responsible for transforming the summed weighted input from the node into the activation of the node or output for that input [17]. The choice of activation function for both the hidden layers (All hidden layers typically use the same activation function) and the output layer has a large impact on the capability and performance of the neural network (Fig. 11).

- The simplest activation function is referred to as the linear activation, where no transform is applied at all. A network comprised of only linear activation functions is very easy to train, but cannot learn complex mapping functions [17].
- Typically, differentiable nonlinear activation functions are used in the hidden layers of a neural network as they allow the nodes to learn more complex structures in the data. Traditionally, two widely used nonlinear activation functions are the sigmoid and hyperbolic tangent activation functions. The sigmoid or logistic activation function takes any real value as input and outputs values in the range 0 to 1. The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to 0.0 [18].
- The hyperbolic tangent function, or tanh for short, is a S-shaped nonlinear activation function, continuous and differentiable, and outputs values between -1.0 and 1.0, helping speed up convergence.
- Both the sigmoid and Tanh functions show the vanishing gradients problem: a deep multilayer feed-forward network is unable to propagate useful gradient information from the output end of the model back to the layers near the input end of the model. The result is that it becomes challenging for the model to learn on a given dataset, or that models prematurely converge to a poor solution. [19]. The solution is to use the rectified linear activation function, or ReL for short. A node or unit that implements this activation function is referred to as a rectified linear activation unit, or ReLU. This is a simple calculation that returns the value provided as input directly, or the value 0 if the input is 0 or less [17]. It can be can described as follow:

  $$ReLU\,(z) = max\,(0, z)$$

  The function is linear for values greater than zero, meaning it has a lot of the desirable properties of a linear activation function when training a neural network using back-propagation [20]. Yet, it is a nonlinear function as negative values are always output as zero. Because the rectified function is linear for half of the input domain and nonlinear for the other half, it is referred to as a piece-wise linear function or a hinge function [17].

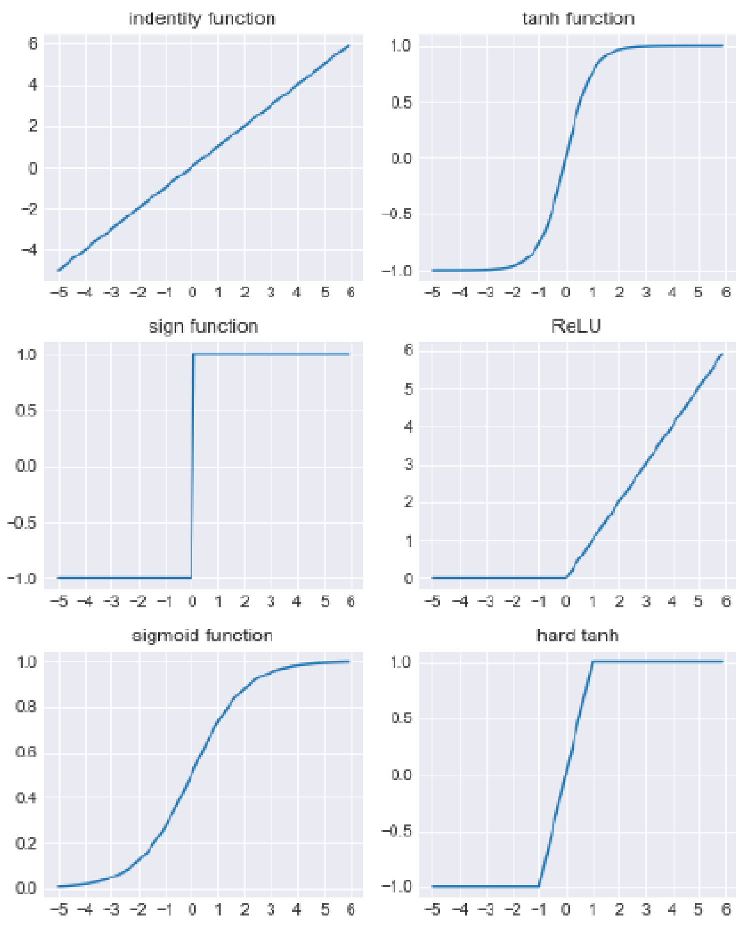


Figure 11: Activation functions [15]

Activation function for output layer The activation function used in the output layer of neural network models for multi-class classification problems where class membership is required on more than two class labels is the softmax function [21]. It can be seen as a generalization of the sigmoid function which was used to represent a probability distribution over a binary variable [20]: it calculates the relative probabilities to determine the final probability value and similar to the sigmoid activation function it returns the probability of each class.

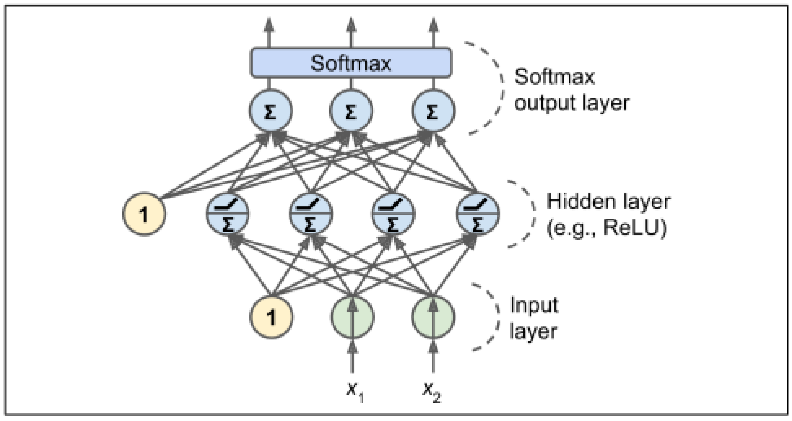


Figure 12: Multilayer perceptrons with ReLu and softmax [15]

Therefore, given a instance x, once it has been computed the score of every class for the instance, it is possible to estimate the probability p_k that the instance belongs to class k by running the scores through the softmax function: it computes the exponential of every score, then normalizes them dividing by the sum of all the exponentials.

$$p_k = \sigma\left(s\left(x\right)\right)_k = \frac{exp(s_k(x))}{\sum_{j=1}^{k} exp(s_j(x))}$$

- k is the number of classes.
- $s\left(x\right)$ is a vector containing the scores of each class for the instance x.
- $\sigma\left(s\left(x\right)\right)_k$ is the estimated probability that the instance x belongs to class k given the scores of each class for that instance.
- The exponential acts as the non-linear function.

Loss function: Regarding the loss function, since we are predicting probability distributions, the cross-entropy (also called the log loss) is generally a good choice:

$$J(\theta) = - \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} y_k^{(i)} \log(\hat{p}_k^{(i)})$$

Cross entropy is frequently used to measure how well a set of estimated class probabilities matches the target classes.

Optimization function: The goal of optimization is to efficiently calculate the parameters/weights that minimize the loss function [22]. The basic approach by which the optimization procedure can be performed is the regular gradient descent. The objective function used in gradient descent is the loss function which we want to minimize [23]. Since it takes always small and regular steps without considering where the earlier gradients were, it could be very slow. Therefore, one can consider to add a momentum/a in order to make the process faster. Furthermore, another improvement that can be done is to correct the direction earlier to point toward the global optimum, as the AdaGrad algorithm does. The problem of this algorithm is that it ends up stopping entirely before reaching the global optimum. RMSProp, on the other side, accumulates only the gradients from the most recent iterations. The combination of these two concepts, namely the use of momentum of the AdaGrad algorithm and the RMSProp process, is at the root of the ADAM (Adaptive moment estimation) optimizer used in this work. Empirical results demonstrate that Adam works well in practice and compares favorably to other stochastic optimization methods [24]. The default value of the learning rate with the Adam optimizer is 0.001 and in practice it works for most cases, so it is usually recommended to leave this hyper-parameter at its default value [25].

Dropout The dropout is one of the most popular regularization techniques for deep neural networks, used in order to try to keep the overfitting problem under control. At every training step, every neuron (including the input neurons, but always excluding the output neurons) has a probability p of being temporarily "dropped out, " meaning it will be entirely ignored during this training step, but it may be active during the next step. The hyper-parameter p is called the dropout rate, and it is typically set between 10% to 50%. After training, neurons don't get dropped anymore.

2.2.2 Convolutional neural networks

Convolutional neural networks (CNN) are a type of deep learning algorithm that typically makes good classifiers and performs particularly well with image classification, but also with sound recognition tasks, since one can extract features from audio datasets which look like images and shape them in order to feed them into a CNN [26].
CNNs are inspired by the observation of how the image recognition is organized in a biological brain. In particular, biological neurons in the visual cortex have a small local receptive field that works within a small localized area. The receptive fields of different neurons may overlap, and together they tile the whole visual field. Moreover, some neurons react only to images of horizontal lines, while others react only to lines with different orientations. Finally, some neurons react to complex patterns generated by combinations of the lower-level patterns.
CNN is a mathematical construct that is typically composed of three types of layers (Fig. 13): convolution, pooling, and fully connected layers. The first two, convolution and pooling layers, perform feature extraction, whereas the third, a fully connected layer, maps the extracted features into final output, such as classification [27].

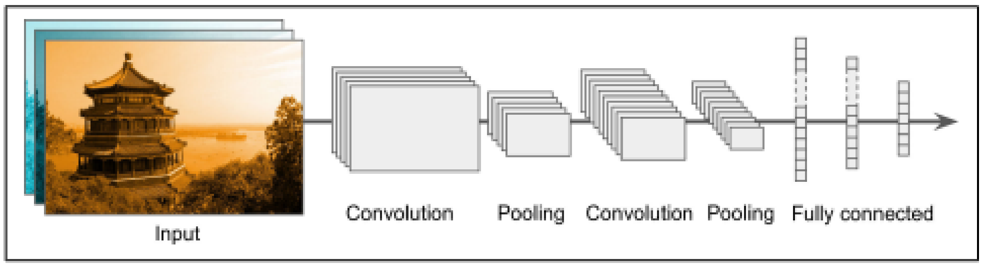


Figure 13: Convolutional neural network architecture [15]

The basic pipeline for CNN is as follows [28]:

- Input an image.
- Perform Convolution operation to get an activation map.
- Apply the pooling layer to make our model robust.
- Activation function (mostly ReLu) is applied to avoid non-linearity.
- Flatten the last output into one linear vector.
- The vector is passed to a fully connected artificial neural network.
- The fully connected layer will provide a probability for each class that we're after.
- Repeat the process to get well defined trained weights and feature detectors.

We should use cross-validation to find the right values as done for the multilayer perceptron model. However, in the CNN model this process is very time-consuming, so it could be useful to make some assumptions on the hyperparamer values based on the evidences encountered in the practice.

Convolutional layer The most important building block of a CNN is the convolutional layer: neurons in the convolutional layer are not connected to every single pixel in the previous layer, but only to pixels in their receptive fields. Then, each neuron in the second convolutional layer is connected only to neurons located within a small rectangle in the first layer. Therefore, CNN automatically and adaptively learn spatial hierarchies of features, from low- to high-level patterns: the most simple features are captured by the layers near the input, the most complex by the deeper one.

Convolutional Kernel A filter, or convolutional kernel, is a matrix of weights with which we convolve on the input. The filter on convolution provides a measure for how close a patch of input resembles a feature. The weights in the filter matrix are derived while training the data. Smaller filters collect as much local information as possible, bigger filters represent more global, high-level and representative information. Note in general we use filters with odd sizes [29]. A common mistake is to use convolution kernels that are too large. It is generally preferable using a convolutional layer with small kernels, such as 3×3 kernels: it will use less parameters and require less computations, and it will usually perform better. One exception to this recommendation is for the first convolutional layer: it can typically have a larger kernel (e.g., 5×5), usually with stride of 2 (or even more): this will reduce the spatial dimension of the image without losing too much information.

Stride It is also possible to connect a large input layer to a much smaller layer by spacing out the receptive fields. The shift from one receptive field to the next is called the stride (Fig. 14). In other words, the stride is the number of pixels that the analysis window moves on each iteration. A stride of 2 means that each kernel is offset by 2 pixels from its predecessor.

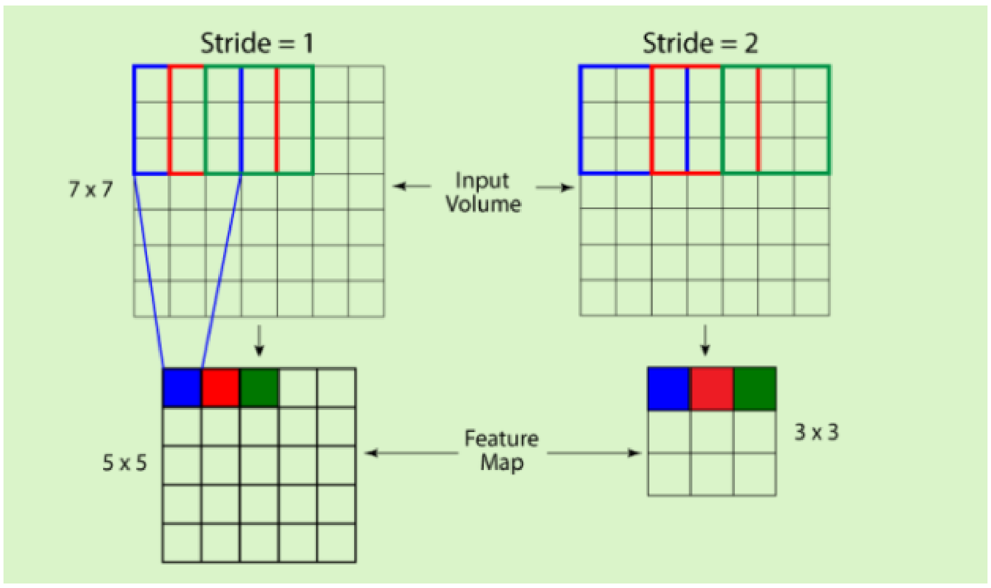


Figure 14: Stride [30]

Padding One can consider to add a frame in which the matrix is put, known as padding. The use of this operation helps to improve the scan of borders when larger filters are considered. In our model, padding has been set to ”SAME” for the convolutional layers:

If set to ”VALID”, the convolutional layer does not use zero padding, and may ignore some rows and columns at the bottom and right of the input image, depending on the stride.

If set to ”SAME” (Fig. 15), the convolutional layer uses zero padding if necessary. In this case, the number of output neurons is equal to the number of input neurons divided by the stride. Then zeros are added as evenly as possible around the inputs.

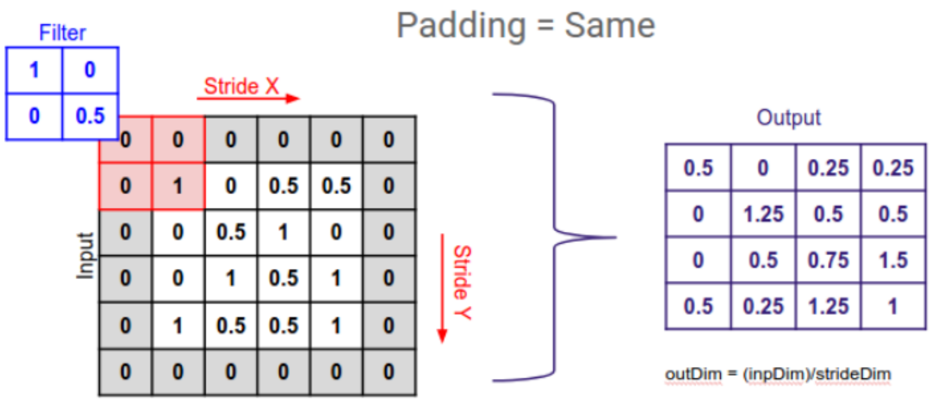


Figure 15: Padding= ”SAME” [31]

Pooling layer The goal of a pooling layer is to shrink the input image in order to reduce the computational load, the memory usage, and the number of parameters (thereby limiting the risk of overfitting). Pooling layers have the same parameters as a convolution layer. We use a 2x2 filter size with a stride of 2. Padding is set to ”SAME”. The max pooling layer (Fig. 16), the most common type of pooling layer, calculates the maximum, or largest, value in each patch of each feature map [29]. Other than reducing computations, memory usage and the number of parameters, a max pooling layer also introduces some level of invariance to small translations. By inserting a max pooling layer every few layers in a CNN, it is possible to get some level of translation invariance at a larger scale and a small amount of rotational invariance and scale invariance. Such invariance (even if it is limited) can be useful in cases where the prediction should not depend on these details, such as in classification tasks.

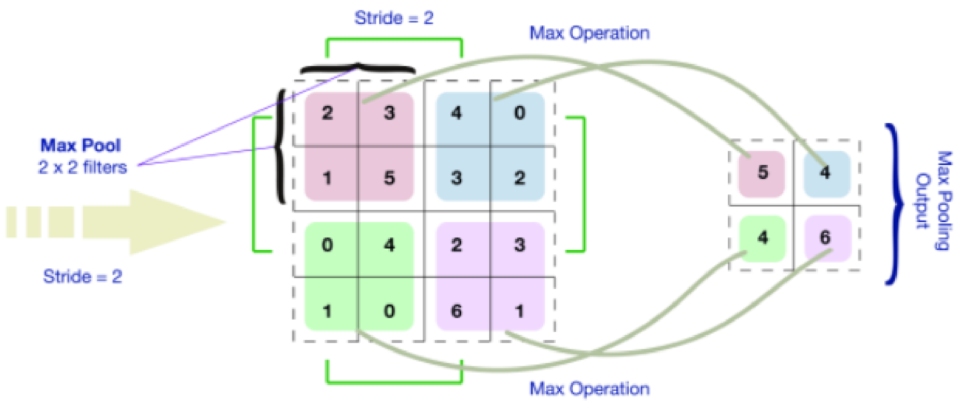


Figure 16: Max pooling layer [32]

Number of filters It is a common practice to double the number of filters after each pooling layer: since a pooling layer divides each spatial dimension by a factor of 2, we can afford doubling the number of feature maps in the next layer, without fear of exploding the number of parameters, memory usage, or computational load.

Last layer Last layer is the fully connected network, composed of some hidden dense layers and a dense output layer. We must flatten its inputs, since a dense network expects a 1D array of features for each instance. We also add some dropout layers in order to reduce overfitting.

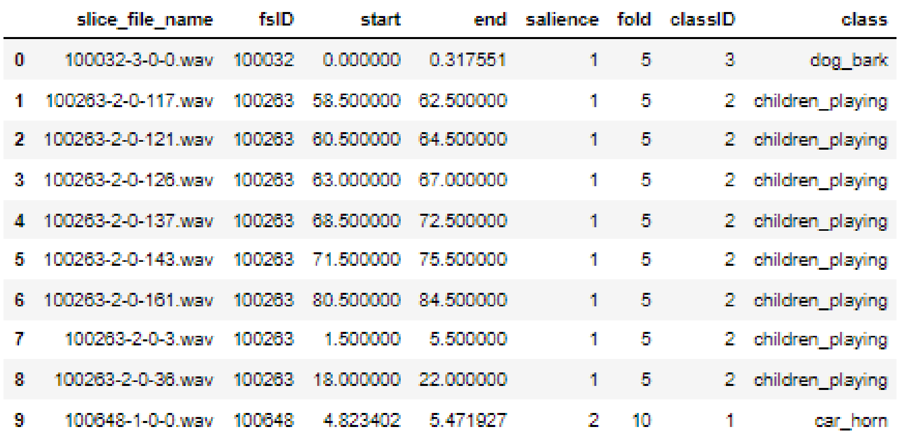
3 Dataset description

The dataset used in this project is UrbanSound8k [1]. It contains 8732 sound excerpts (<= 4 seconds) of urban sounds extracted from the Freesound repository, a repository containing more than 160, 000 audio clips. The sounds are labeled with 10 classes drawn from the urban sound taxonomy:

- air conditioner,
- car horn,
- children playing,
- dog bark,
- drilling,
- enginge idling,
- gun shot,
- jackhammer,
- siren,
- street music

Since the labels are categorical, we transformed them into binary variables through the One-Hot Econding technique, creating as many dummy variables as categories are and, for each row, setting to 1 the dummy corresponding the category to which the file belongs and 0 the others. In addition to the sound excerpts, a CSV file containing metadata about each excerpt is also provided (Fig. 17). The CSV file contains 8 columns:

- slice_ file_ name: The name of the audio file,
- fsID: The Freesound ID of the recording from which this excerpt is taken,
- start: The start time of the slice in the original Freesound recording,
- end: The end time of slice in the original Freesound recording,
- salience: A (subjective) salience rating of the sound (1 = foreground, 2 = background),
- fold: The fold number (1-10) to which this file has been pre-allocated,
- classID:A numeric identifier of the sound class,
- class:The class name.



	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

Figure 17: Dataset

Going further with the explanatory data analysis, it turns out that the class labels are unbalanced, as shown in Fig. 18.

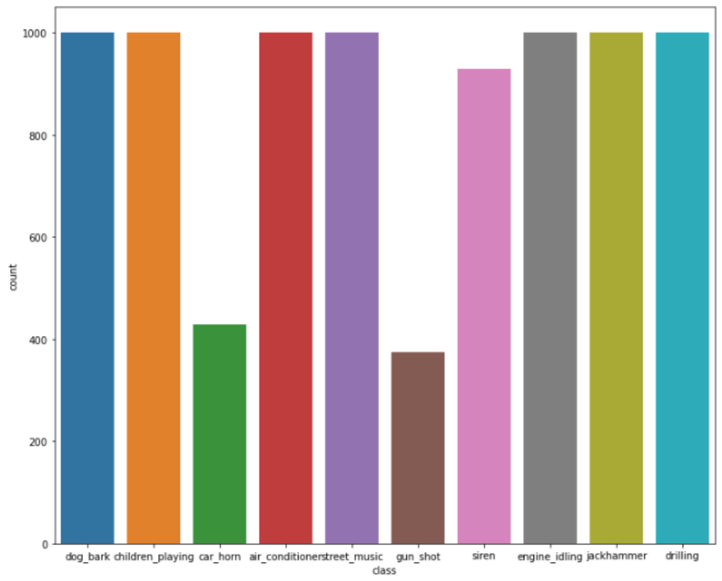
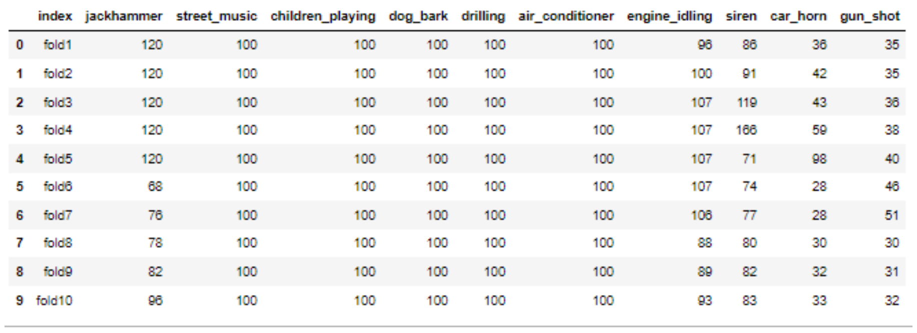


Figure 18: Labels distribution

Although 7 out of the 10 total classes have exactly 1000 samples, and siren is not far off with 929, the remaining two (car horn, gun shot) have significantly less samples at 429 and 374 respectively. This problem is solved by using the folds n. 1, 2, 3, 4, 6 as training set and the remaining folds n. 5, 7, 8, 9, 10 as test set according to the dataset authors' note stating that the number of slices-per fold for each sound class are balanced among the folds [1], as Fig. 19 shows. The splitting is done before working on the data with the librosa function in order to normalize the sets separately, since no data manipulation should depend on test set information.



	index	jackhammer	street_music	children_playing	dog_bark	drilling	air_conditioner	engine_idling	siren	car_horn	gun_shot
0	fold1	120	100	100	100	100	100	96	86	36	35
1	fold2	120	100	100	100	100	100	100	91	42	35
2	fold3	120	100	100	100	100	100	107	119	43	36
3	fold4	120	100	100	100	100	100	107	166	59	38
4	fold5	120	100	100	100	100	100	107	71	98	40
5	fold6	68	100	100	100	100	100	107	74	28	46
6	fold7	76	100	100	100	100	100	106	77	28	51
7	fold8	78	100	100	100	100	100	88	80	30	30
8	fold9	82	100	100	100	100	100	89	82	32	31
9	fold10	96	100	100	100	100	100	93	83	33	32

Figure 19: Labels distribution in the folds

The audio samples are in the .wav format and these continuous waves of sounds are converted into a one-dimensional numpy array of digital values by sampling them at discrete intervals known as the sampling rate [33]. Such samples can be used to represent the amplitude of the wave at a particular time interval. Librosa uses a default sample rate of 22050 Hz, that assists in keeping the array size small by decreasing the audio quality but greatly reducing the training time [34]. Moreover, using the librosa.load() function with the mono parameter set to true, the two channels of the stereo audio signal are combined to make an one dimensional numpy array, thus flattening the audio signal into mono. Furthermore, it also normalizes the data so that the values can be represented in a range between -1 and 1, as shown in Fig. 20.

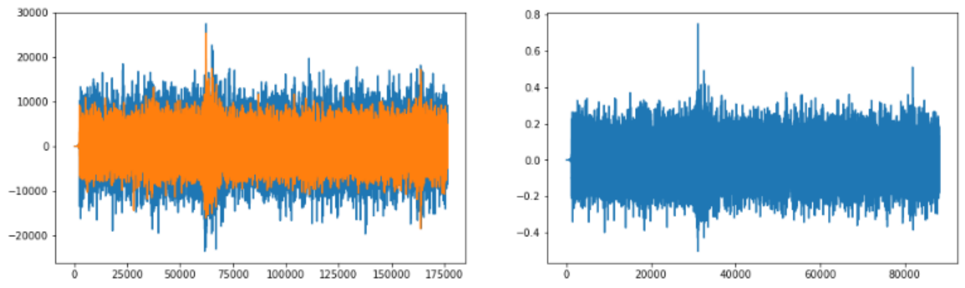


Figure 20: Two vs mono audio signal

4 Findings

4.1 Feature extraction

In our analysis, we decided to compare the results obtained using two different features: the MFCCs and the STFT chromagram. The MFCCs are coefficients that exploit the mel scale instead of the linear scale used by the basic spectrograms, solving the fact that it is much harder for humans to be able to distinguish between higher frequencies and easier for lower frequencies. Furhtermore, MFCC is a very compressible representation, often using just a small set of coefficients instead of 32-64 bands of the Mel spectrogram. In our analysis, we used 25 coefficients, according to choice of the creators of the dataset [1]. Note that MFCCs are not very robust in the presence of additive noise [35].
On the the other side, the chromagram could be an interesting feature since it captures harmonic and melodic characteristics of sounds: in this case we extracted 12 features, the deafult value on the librosa chroma_stft function [12].

4.2 Multilayer perceptron

In the first step of our project, we implemented the multilayer perceptron model using the MFCCs and the STFT chromagram. The number of input neurons is the number of input features extracted, namely 25 for the MFCCs and 12 for the STFT chromagram over 173 time frames. The number of output neurons is the number of classes, that are 10 in our dataset. We used the ReL activation function for the hidden layers, since it is the most used for the motivations that we explained previously in the paragraph 2.2.1, and the softmax function for the output layer, since we are facing a multi-class classification problem. The loss function is the categorical cross entropy. Finally, the optimization function that we chose is the Adam, since it is the optimizer that works in the better way compared to the other stochastic optimization methods [24]. We applied the hyper-parameter tuning using the K-fold cross-validation in order to find the number of hidden layers (in a range between 1 and 5), the number of neurons for each hidden layer (in a range between 10 and 100), the number of epochs, that is the number of times that the learning algorithm will work through the entire training dataset (in a range between 10 and 100), the batch size, namely the number of samples to work through before updating the internal model parameters (in a range between 10 and 100) and the dropout (in a range betweeen 0.1 and 0.5). The ranges chosen for the tuning of the hyper-parameters have been retrieved by the Geron textbook [15].

4.2.1 MFCCs

We found out that, for the multilayer perceptrons model using the MFCCs, 70 neurons for 4 hidden layers are the best parameters to use. The batch size is 80. The best value for the dropout is 0.2. During the tuning of the number of epochs, we found out that 50 is the optimal value for this parameter. However, we used the early stopping method, a form of regularization used to avoid overfitting when training a learner with an iterative method. With Early Stopping, you just stop training as soon as the test error reaches the minimum [36], in our case at 36 epochs, as shown in Fig. 21.

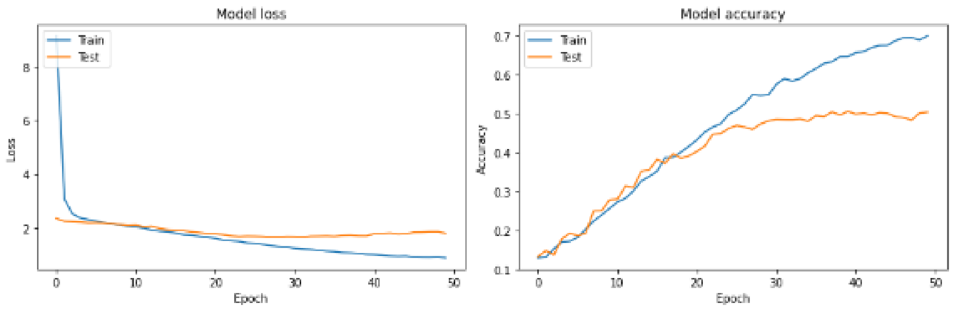
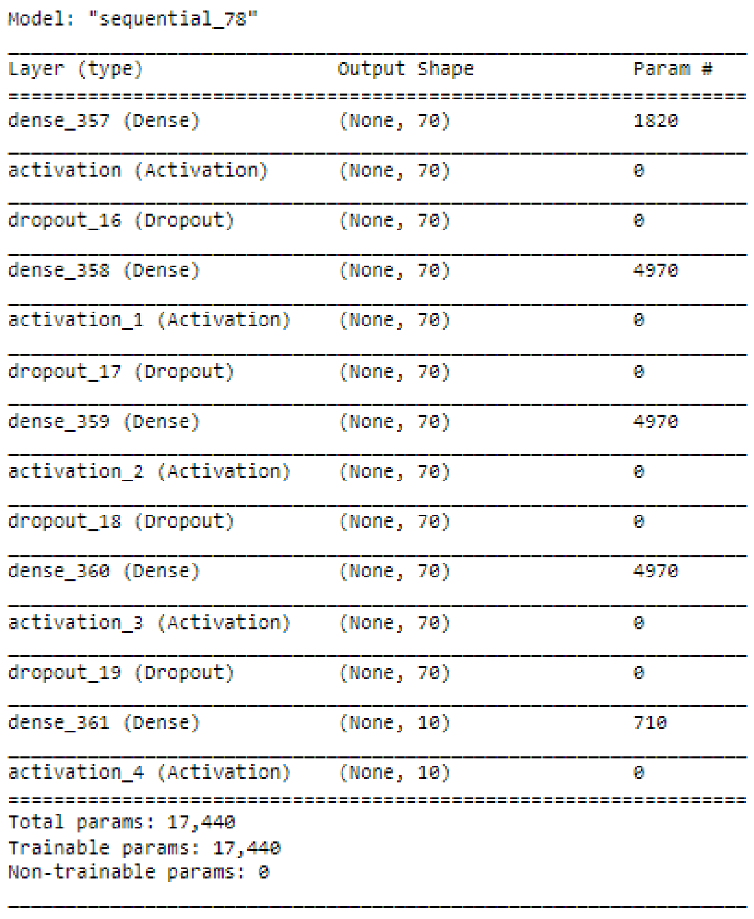


Figure 21: Multilayer perceptron with MFCCs, loss and model accuracy

The model is summarized as shown in Fig. 22.



```
Model: "sequential_78"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_357 (Dense)            (None, 70)                1820
_________________________________________________________________
activation (Activation)      (None, 70)                0
_________________________________________________________________
dropout_16 (Dropout)         (None, 70)                0
_________________________________________________________________
dense_358 (Dense)            (None, 70)                4970
_________________________________________________________________
activation_1 (Activation)    (None, 70)                0
_________________________________________________________________
dropout_17 (Dropout)         (None, 70)                0
_________________________________________________________________
dense_359 (Dense)            (None, 70)                4970
_________________________________________________________________
activation_2 (Activation)    (None, 70)                0
_________________________________________________________________
dropout_18 (Dropout)         (None, 70)                0
_________________________________________________________________
dense_360 (Dense)            (None, 70)                4970
_________________________________________________________________
activation_3 (Activation)    (None, 70)                0
_________________________________________________________________
dropout_19 (Dropout)         (None, 70)                0
_________________________________________________________________
dense_361 (Dense)            (None, 10)                710
_________________________________________________________________
activation_4 (Activation)    (None, 10)                0
=================================================================
Total params: 17,440
Trainable params: 17,440
Non-trainable params: 0
_________________________________________________________________
```

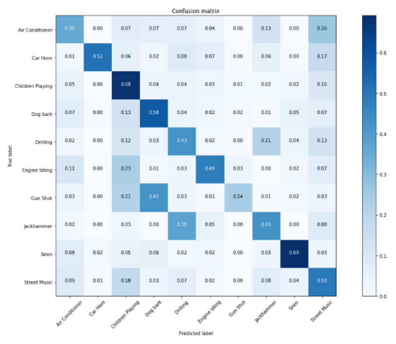
Figure 22: Multilayer perceptron with MFCCs

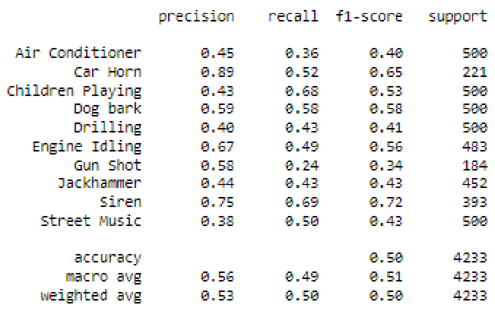
The model results with a training and test accuracy as shown in Table 1.

Training	Test
0.73	0.47

Table 1: Accuracy for Multilayer perceptron, MFCCs

Moreover the confusion matrix and classification report are shown in Fig. 23.





```
                  precision    recall  f1-score   support

 Air Conditioner       0.45      0.36      0.40       500
        Car Horn       0.89      0.52      0.65       221
Children Playing       0.43      0.68      0.53       500
        Dog bark       0.59      0.58      0.58       500
        Drilling       0.40      0.43      0.41       500
   Engine Idling       0.67      0.49      0.56       483
        Gun Shot       0.58      0.24      0.34       184
      Jackhammer       0.44      0.43      0.43       452
           Siren       0.75      0.69      0.72       393
    Street Music       0.38      0.50      0.43       500

        accuracy                           0.50      4233
       macro avg       0.56      0.49      0.51      4233
    weighted avg       0.53      0.50      0.50      4233
```

Figure 23: Multilayer perceptron with MFCCs- Confusion matrix and classification report

4.2.2 STFT Chromagram

We found out that the optimal parameters for the multilayer perceptrons model using the STFT Chromagram are: 100 neurons for 3 hidden layers. The batch size is 40. The dropout is 0.2. The optimal number of epochs is 50, but as shown in Fig. 24, the early stopping stops the learning at 24 epochs.

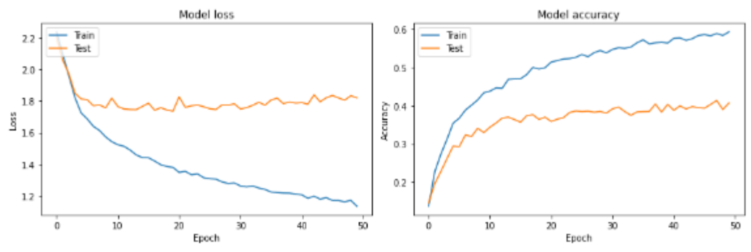
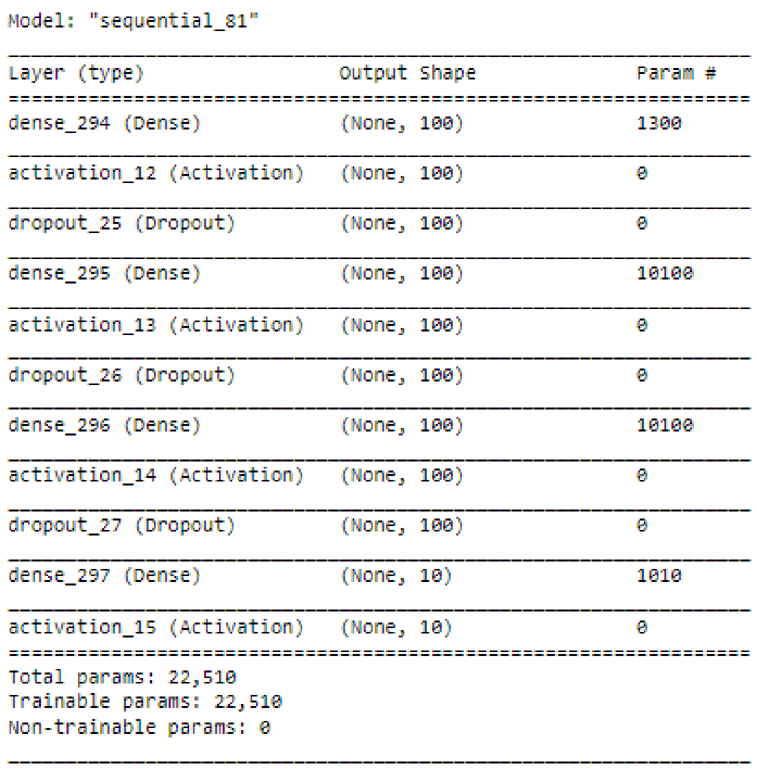


Figure 24: Multilayer perceptron with STFT Chromagram, loss and model accuracy

The model is summarized in Fig. 25.



```
Model: "sequential_81"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_294 (Dense)            (None, 100)               1300      
_________________________________________________________________
activation_12 (Activation)   (None, 100)               0         
_________________________________________________________________
dropout_25 (Dropout)         (None, 100)               0         
_________________________________________________________________
dense_295 (Dense)            (None, 100)               10100     
_________________________________________________________________
activation_13 (Activation)   (None, 100)               0         
_________________________________________________________________
dropout_26 (Dropout)         (None, 100)               0         
_________________________________________________________________
dense_296 (Dense)            (None, 100)               10100     
_________________________________________________________________
activation_14 (Activation)   (None, 100)               0         
_________________________________________________________________
dropout_27 (Dropout)         (None, 100)               0         
_________________________________________________________________
dense_297 (Dense)            (None, 10)                1010      
_________________________________________________________________
activation_15 (Activation)   (None, 10)                0         
=================================================================
Total params: 22,510
Trainable params: 22,510
Non-trainable params: 0
_________________________________________________________________
```

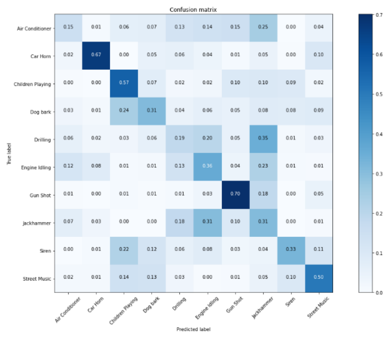
Figure 25: Multilayer perceptron with STFT Chromagram

The model results with a training and test accuracy as shown in Table 2.

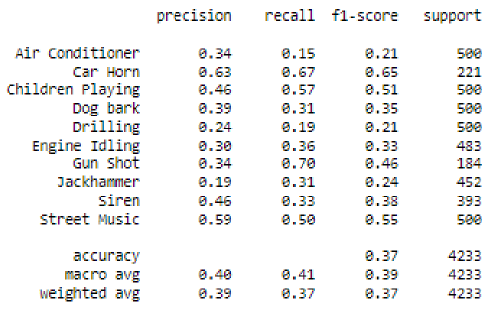
Training	Test
0.59	0.38

Table 2: Accuracy for Multilayer perceptron, STFT Chromagram

Moreover the confusion matrix and classification report are shown in Fig. 26.



	Air Conditioner	Car Horn	Children Playing	Dog bark	Drilling	Engine Idling	Gun Shot	Jackhammer	Siren	Street Music
Air Conditioner	0.15	0.01	0.06	0.07	0.13	0.14	0.15	0.25	0.00	0.04
Car Horn	0.02	0.67	0.00	0.05	0.06	0.04	0.01	0.05	0.00	0.10
Children Playing	0.00	0.00	0.57	0.07	0.02	0.02	0.10	0.10	0.09	0.02
Dog bark	0.03	0.01	0.24	0.31	0.04	0.06	0.05	0.08	0.08	0.09
Drilling	0.06	0.02	0.03	0.06	0.19	0.20	0.05	0.35	0.01	0.03
Engine Idling	0.12	0.08	0.01	0.01	0.13	0.36	0.04	0.23	0.01	0.01
Gun Shot	0.01	0.00	0.01	0.01	0.01	0.03	0.70	0.18	0.00	0.05
Jackhammer	0.07	0.03	0.00	0.00	0.18	0.31	0.10	0.31	0.00	0.01
Siren	0.00	0.01	0.22	0.12	0.06	0.08	0.03	0.04	0.33	0.11
Street Music	0.02	0.01	0.14	0.13	0.03	0.00	0.01	0.05	0.10	0.50



```
                  precision    recall  f1-score   support

 Air Conditioner       0.34      0.15      0.21       500
        Car Horn       0.63      0.67      0.65       221
Children Playing       0.46      0.57      0.51       500
        Dog bark       0.39      0.31      0.35       500
        Drilling       0.24      0.19      0.21       500
   Engine Idling       0.30      0.36      0.33       483
        Gun Shot       0.34      0.70      0.46       184
      Jackhammer       0.19      0.31      0.24       452
           Siren       0.46      0.33      0.38       393
    Street Music       0.59      0.50      0.55       500

        accuracy                           0.37      4233
       macro avg       0.40      0.41      0.39      4233
    weighted avg       0.39      0.37      0.37      4233
```

Figure 26: Multilayer perceptron with STFT Chromagram- Confusion matrix and classification report

4.3 Convolutional neural networks

The structure of the convolutional neural networks that we implemented is taken as in the example exhibited in the Geron book [15]. To adapt this solution to our case and to get better results, we tried to change the values of some hyper-parameters, namely the number of epochs, the dropout value, the number of filters and the number of layer.
The first Conv2D layer receiving the input shape consists of 64 filters (then, it will be doubled after each pooling layer, as we explained in the theoretical section 2.2.2), a kernel size of 5 (in all the other convolutional layer the size will be 3, since the first convolutional layer typically has a larger kernel) and stride set to 2. Padding has been set equal to 'same', which produces an output of the same height and weight as the input. The activation function we used for the convolutional layers is the ReL function [34]. Since the image is grey-scale, then the channel argument takes a value of 1; if it was coloured, then it would take a value of 3, one for each of Red, Green and Blue channels
Each Conv2D layer is followed by a MaxPooling2D layer, used in reducing the dimension of the input shape fed through the convolutional layer with a pool size of 2 and a strides of 2 with the padding set to valid.
Finally, we have the fully connected network, composed of some dense hidden layers and a dense output layer. Two dense hidden layers (fully connected layers) and an output layer have been used in our models: respectively the dense hidden layers will have 128 and 64 nodes correspondingly. The activation function we used for the dense hidden layers is the ReL function. A dropout of 0.3 is added in each dense hidden layer. The output layer having as activation function the softmax function consists of 10 output neurons that correspond to the number of classes. The model predicts the choice with the highest probability.
In the first implementation of the convolutional neural network, we used 3 convolutional layers, each of them followed by a corresponding pooling layer, and 2 dense hidden layers. The number of epochs corresponds to 20. Consequently, we changed some hyper-parameters: in the second model we implemented, a dropout of 0.3 is added in the convolutional layers (while in the original one the dropout was not inserted); in the third model, we used 4 convolutional layers instead of the 3 of the original model; in the fourth model, we used just 1 dense hidden layer instead of 2; in the last model the number of epochs decreased to 10 epochs.

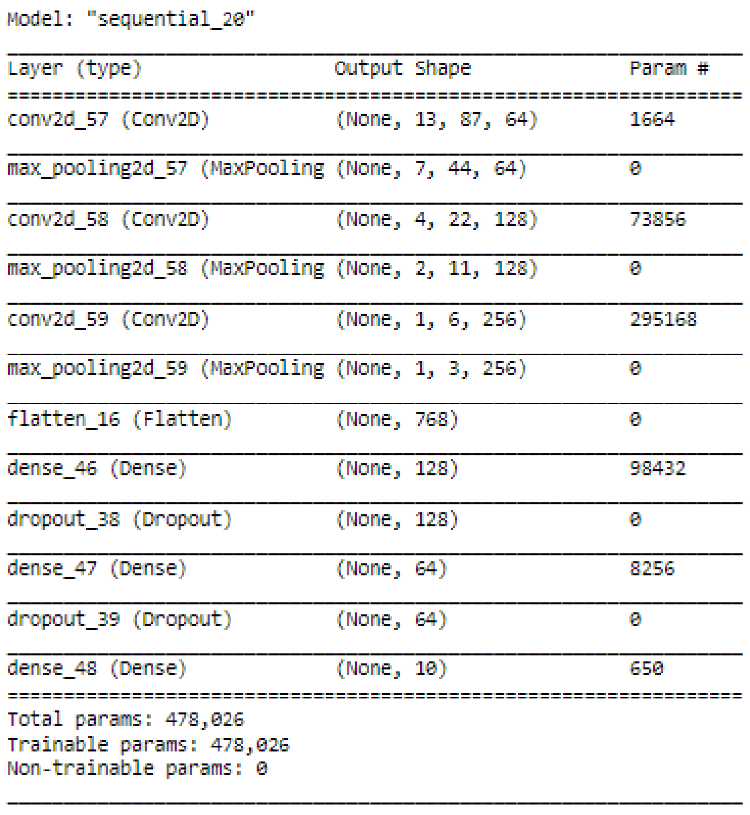
4.3.1 MFCCs

Table 3 shows the scores of the different trained models using as features the MFCCs: it is evident that there is not a meaningful increase or decrease of our measure of interest, the accuracy, since it floats around the same interval.

Among the trained models, the one that shows to have obtained the best measure of test accuracy is model 5, that is shown in Fig. 27.

Model	CONV2D/Dense Layer	Filters	Dropout	Epochs	Training	Test
Model 1	3 CONV2D/2 Dense	64-128-256/64-128	N/A	20	0.98	0.56
Model 2	3 CONV2D/2 Dense	64-128-256/64-128	0.3	20	0.92	0.57
Model 3	4 CONV2D/2 Dense	64-128-256-512/64-128	N/A	20	0.96	0.55
Model 4	2 CONV2D/1 Dense	64-128-256/128	N/A	20	0.97	0.56
Model 5	3 CONV2D/2 Dense	64-128-256/64-128	N/A	10	0.94	0.57

Table 3: Convolutional neural network comparison, MFCCs



```
Model: "sequential_20"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_57 (Conv2D)           (None, 13, 87, 64)        1664
_________________________________________________________________
max_pooling2d_57 (MaxPooling (None, 7, 44, 64)         0
_________________________________________________________________
conv2d_58 (Conv2D)           (None, 4, 22, 128)        73856
_________________________________________________________________
max_pooling2d_58 (MaxPooling (None, 2, 11, 128)        0
_________________________________________________________________
conv2d_59 (Conv2D)           (None, 1, 6, 256)         295168
_________________________________________________________________
max_pooling2d_59 (MaxPooling (None, 1, 3, 256)         0
_________________________________________________________________
flatten_16 (Flatten)         (None, 768)               0
_________________________________________________________________
dense_46 (Dense)             (None, 128)               98432
_________________________________________________________________
dropout_38 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_47 (Dense)             (None, 64)                8256
_________________________________________________________________
dropout_39 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_48 (Dense)             (None, 10)                650
=================================================================
Total params: 478,026
Trainable params: 478,026
Non-trainable params: 0
_________________________________________________________________
```

Figure 27: Convolutional neural network with MFCCs

After compiling and fitting the best model in the data, the final plot of the model loss and model accuracy is shown in Fig. 28.

Moreover the confusion matrix and classification report are presented in Fig. 29.

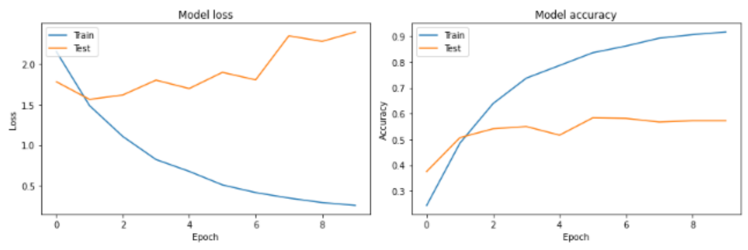
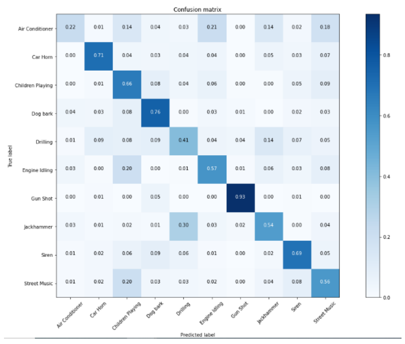
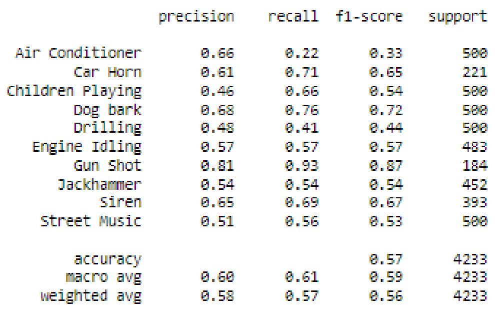


Figure 28: CNN with MFCCs, loss and model accuracy (5)





```
                  precision    recall  f1-score   support

 Air Conditioner       0.66      0.22      0.33       500
        Car Horn       0.61      0.71      0.65       221
Children Playing       0.46      0.66      0.54       500
        Dog bark       0.68      0.76      0.72       500
        Drilling       0.48      0.41      0.44       500
   Engine Idling       0.57      0.57      0.57       483
        Gun Shot       0.81      0.93      0.87       184
      Jackhammer       0.54      0.54      0.54       452
           Siren       0.65      0.69      0.67       393
    Street Music       0.51      0.56      0.53       500

        accuracy                           0.57      4233
       macro avg       0.60      0.61      0.59      4233
    weighted avg       0.58      0.57      0.56      4233
```

Figure 29: CNN with MFCCs - Confusion matrix and classification report

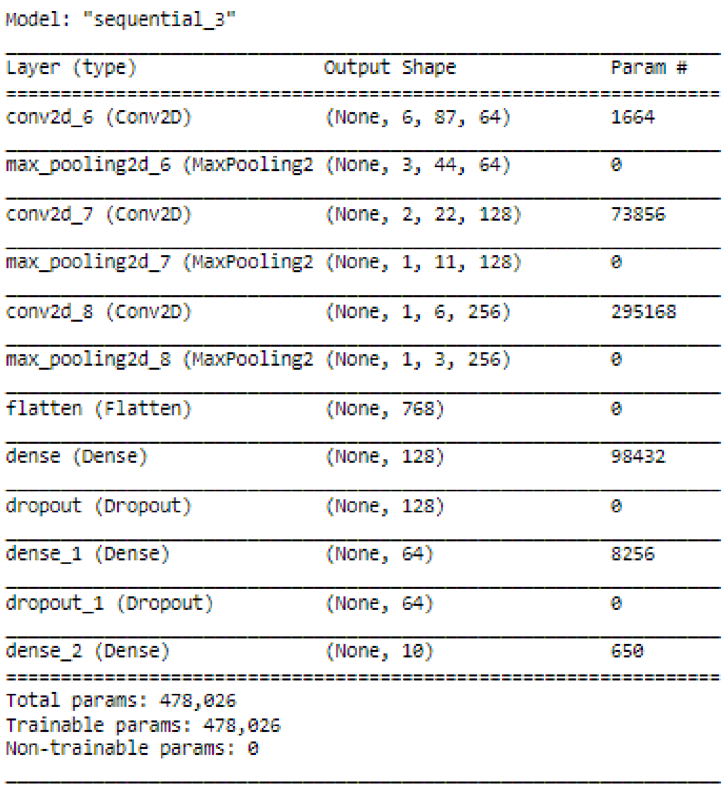
4.3.2 STFT Chromagram

Again, also for Short-time Fourier transformation chromagram of audio files we trained several convolutional neural network models, each of one is shown in Table 4, which reports their characteristics and scores: it is evident that there is not a meaningful improvement in the test accuracy, while the last model perfoms much worse.

Model	CONV2D/Dense Layer	Filters	Dropout	Epochs	Training	Test
Model 1	3 CONV2D/2 Dense	64-128-256/64-128	N/A	20	0.89	0.48
Model 2	3 CONV2D/2 Dense	64-128-256/64-128	0.3	20	0.69	0.45
Model 3	4 CONV2D/2 Dense	64-128-256-512/64-128	N/A	20	0.88	0.46
Model 4	2 CONV2D/1 Dense	64-128-256/128	N/A	20	0.86	0.47
Model 5	3 CONV2D/2 Dense	64-128-256/64-128	N/A	10	0.52	0.36

Table 4: Convolutional neural network comparison, STFT Chromagram

After compiling and fitting the best model in the data (summarized in Fig. 30), the final plot of the model loss and model accuracy is shown in Fig. 31.



```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 6, 87, 64)         1664
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 3, 44, 64)         0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 2, 22, 128)        73856
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 1, 11, 128)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 1, 6, 256)         295168
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 1, 3, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 768)               0
_________________________________________________________________
dense (Dense)                (None, 128)               98432
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650
=================================================================
Total params: 478,026
Trainable params: 478,026
Non-trainable params: 0
_________________________________________________________________
```

Figure 30: Convolutional neural network with STFT Chromagram

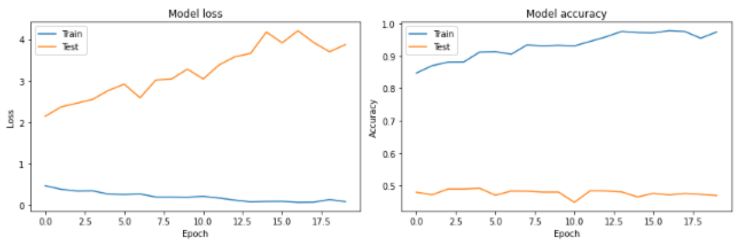
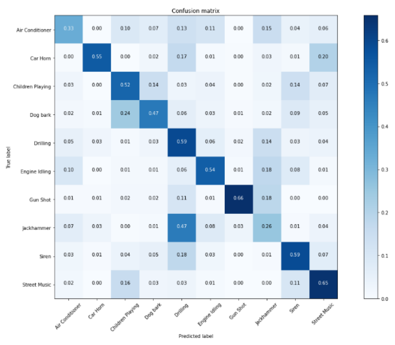


Figure 31: CNN with STFT Chromagram, loss and model accuracy (1)

Moreover the confusion matrix and classification report are presented in Fig.32.



```
                  precision    recall  f1-score   support

 Air Conditioner       0.51      0.32      0.39       500
        Car Horn       0.64      0.46      0.53       221
Children Playing       0.40      0.52      0.45       500
        Dog bark       0.49      0.52      0.50       500
        Drilling       0.41      0.37      0.39       500
   Engine Idling       0.47      0.54      0.50       483
        Gun Shot       0.72      0.70      0.71       184
      Jackhammer       0.35      0.28      0.31       452
           Siren       0.44      0.54      0.49       393
    Street Music       0.53      0.59      0.56       500

        accuracy                           0.47      4233
       macro avg       0.50      0.48      0.48      4233
    weighted avg       0.47      0.47      0.47      4233
```

Figure 32: CNN with STFT Chromagram- Confusion matrix and classification report

4.4 Cross-validation Results

This section provides an overview of the results we found during the validation of our models. In particular, this step it is important to validate our results implementing the model on different splits of the dataset into training and test sets, so to exclude the possibility that our results are provided by a particular partition of the dataset. For achieving this purpose, we perform a 10-fold cross-validation for all the models, as show in Table 5. However, it is important to note that in the accomplishment of this task the splittings of the dataset do not maintain the proportion of the rest of the work, so the higher value of average accuracy could be given by the fact that the training set is bigger than the one we used during the previous analysis. Moreover, since the folds are generated in a completely random way and since in our dataset there are multiple slices coming from the same original recording, we may end up with slices from the same recording used both for training and testing, which can lead to artificially high classification accuracies [1]. Due to excessive time that this process requires, we decided to perform this analysis only for the multilayer perceptron. The validation for the convolutional neural networks is left to further analysis using machines with better computational performance than the current ones.

Model	Accuracy	St.Deviation
MultiLayer Perceptron - MFCC	79.90%	2.19%
MultiLayer Perceptron - STFT	57.95%	2.12%

Table 5: 10-Fold Cross Validation results

5 Conclusions

In this project, we used Tensorflow2 in order to train neural networks for the classification of sound events based on audio files from the UrbanSound8K dataset.
In the first part of the paper, a theoretical background was given concerning the several ways in which different features that are commonly used in the field of sound events classification can be extracted. Specifically, we focused on three kind of representation of an audio signal: The waveplots, the spectrograms, the chromagrams. For the purpose of our analysis, we used two features extraction methodologies: the Mel-frequency cepstral coefficients (MFCCs) and the Short time Fourier transformation (STFT) chromagram. Going further, two different artificial neural network architectures were analyzed in a theoretical perspective: the multilayer perceptrons and the convolutional neural networks. Furthermore, their training parameters were discussed in order to document their influence on the final predictive performance.
In the second chapter of this paper, a brief description of the dataset and of the preprocessing part of this project was given: how the dataset was splitted, how the training set and the test set were normalized, how the audio signal was flattened into a mono audio signal in order to retrieve a one dimensional numpy array, how the categorical labels were transformed into binary variables through the One-Hot Econding technique.
Finally, the findings were be presented and discussed. We found out that the Mel frequency ceptral coefficients perform better than the STFT chromagram in the implementation of the neural network model for the analysis of our dataset. Furthermore, we had a proof that the convolutional neural networks give better results than the multilayer perceptron models for sound recognition tasks.

References

[1] J. Salamon, C. Jacoby and J. P. Bello, ”A Dataset and Taxonomy for Urban Sound Research”, Proceedings of the 22nd ACM International Conference on Multimedia, 2014

[2] Y. Verma, ”A Guide To Audio Data Preparation Using TensorFlow”, Developers Corner, 2021

[3] ”Time Domain vs. Frequency Domain of Audio Signals”, Lerning about Electronics, 2018

[4] Technical Editor, ”Difference time-domain frequency-domain”, 2017

[5] B. McFee, C. Raffel, D Liang, D. PW Ellis, M. McVicar, E.Battenberg, O. Nieto, “Librosa: Audio and music signal analysis in python”, Proceedings of the 14th python in science conference, pp. 18-25, 2015

[6] V. Lendave, ”Hands-On Guide To Librosa For Handling Audio Files”, 2021

[7] K.Chaudhary, ”Understanding Audio data, Fourier Transform, FFT and Spectrogram features for a Speech Recognition System. An introduction to audio data analysis (sound analysis) using python”, 2020

[8] L. Ireland, ”Classificazione del suono utilizzando il deep learning”, 2020

[9] Learning from Audio: The Mel Scale, Mel Spectrograms, and Mel Frequency Cepstral Coefficients, Toward Data Science, 2021

[10] A. Vidhya, ”Simplifying Audio Data: FFT, STFT MFCC”, 2020

[11] M. Sahidullah, G. Saha, ”Design, analysis and experimental evaluation of block based transformation in MFCC computation for speaker recognition”, Speech Communication 54 (4), pp. 543–565, 2012

[12] G. Wakefield, ”Chromagram visualization of the singing voice”, 1999

[13] Y. Verma, ”A Tutorial on Spectral Feature Extraction for Audio Analytics”, 2021

[14] M. Muller, S. Ewert, ”Chroma toolbox: matlab implementations for extracting variants of chroma-based audio features”, 12th International Society for Music Information Retrieval Conference, 2011

[15] A. Geron, ”Hands-On Machine Learning with Scikit Learn Keras and Tensorflow. Concepts Tools and Techniques to Build Intelligent Systems”, O’Reilly Media, 2019

[16] S. Ramesh, ”A guide to an efficient way to build neural network architectures-Part I: Hyperparameter selection and tuning for Dense Networks using Hyperas on Fashion-MNIST”, 2018

[17] J.Brownlee, ”A Gentle Introduction to the Rectified Linear Unit (ReLU)”, 2019

[18] J. Brownlee, ”How to Choose an Activation Function for Deep Learning”, 2021

[19] J. Brownlee, ”How to Fix the Vanishing Gradients Problem Using the ReLU”, 2020

[20] I. Goodfellow, Y. Bengio, A. Courville, ”Deep Learning, Adaptive Computation and Machine Learning series”, MIT Press, 2016

[21] J. Brownlee, ”Softmax Activation Function with Python”, 2020

[22] J. Brownlee, ”Gentle Introduction to the Adam Optimization Algorithm for Deep Learning”, 2017

[23] M. Stewart, ”Simple Guide to hyperparameter Tuning in Neural Networks”, 2019

[24] D. P. Kingma, J. Ba ”Adam: A Method for Stochastic Optimization”, 2014

[25] S. Lau, ”Learning Rate Schedules and Adaptive Learning Rate Methods for Deep Learning”, 2017

[26] P. Nandi, ”CNNs for Audio Classification. A primer in deep learning for audio classification using tensorflow”, 2021

[27] R. Yamashita, M. Nishio, R. K. G. Do, K. Togashi, ”Convolutional neural networks: an overview and application in radiology”, Insights into Imaging volume 9, pp. 611–629, 2018

[28] D. Trehan ”A perfect guide to Convolution Neural Networks”, 2020

[29] S. Ramesh, ”A guide to an efficient way to build neural network architectures-Part II: Hyperparameter selection and tuning for Convolutional Neural Networks using Hyperas on Fashion-MNIST”, 2018

[30] developersbreach.com

[31] Y. Verma, ”Guide to Different Padding Methods for CNN Models”, 2021

[32] S. Shankar, ”Max-pooling principles”, 2018

[33] M. Smales, ”Sound classification using deep learning”, 2019

[34] J. K.Das, A. Ghosh, A.K. Pal, A. Chakrabarty, ”Urban Sound Classification Using Convolutional Neural Network and Long Short Term Memory Based on Multiple Features”, 2020

[35] V. Tyagi, C. Wellekens, ”On desensitizing the Mel-Cepstrum to spurious spectral components for Robust Speech Recognition, in Acoustics, Speech, and Signal Processing”, IEEE International Conference on, vol. 1, pp. 529–532, 2005

[36] B. Chen, ”Early Stopping in Practice: an example with Keras and TensorFlow 2.0. A step to step tutorial to add and customize Early Stopping”, 2020