# Urban Sound Classification with Neural Networks

Statistical Methods for Machine Learning

Experimental Project Report

University of Milan

Department of Economics, Management and Quantitative Methods

Master's Degree in Data Science and Economics

Academic year 2021/2022

## Abstract

Every day we hear different sounds from different sources. We can recognize several random examples of sounds from the environment such as an ambulance, an engine or an airplane. But can we train a computer to classify these random sounds? If the answer is yes, it could be used for surveillance and monitoring or for other environmental researches such as the urban sound pollution in the cities, and it can be applied to the general audio recognition as well. The objective of this paper is to create a suited machine learning technique to classify urban sounds accurately. The dataset used is the UrbanSound8K audio dataset, comprised of 8732 audio clips of urban sounds divided in 10 classes. Two different approaches are followed: one using Artificial Neural Network (ANN) fed with raw data of extracted features of sound and one using Convolutional Neural Network (CNN) fed with images of the spectrums of sound. After testing the models with 5-fold cross validation, the algorithm that performed the best was the Artificial Neural Network, when paired with the use of 40 Mel-Frequency Cepstral Coefficients as input features.

# Table of Contents

# 1. Introduction

This section presents a general and brief overview of sound, its structure and features. The outline of the paper is then laid down.

Audio recognition algorithms are traditionally used for the tasks of speech and music signal processing. There are multiple applications already proposed in a big variety of industries, including surveillance, audio scene recognition for robot navigation, acoustic monitoring of natural and artificial environment. Specifically related to urban sounds which we are analyzing in this research, there are several benefits to study this type of dataset. For example, City noise managing significantly contributes to a healthy and safe living environment in the big cities. In travel centric systems, city sounds may enter the emerging solutions to develop and share journey experience. Assisting technologies for people with disabilities and navigation systems for blind or visually impaired people effectively incorporate urban sound models. The focus of this paper is to build a neural network to classify audio files from the UrbanSound8k dataset. Normally, the audio features are extracted by separating the audio signal considered into frames with a hamming window. A set of features are extracted for each frame and used for training and testing. Two approaches were followed. Firstly, an Artificial Neural Network was built using raw data extracted from the audio clips through the Librosa library. Secondly, a Convolutional Neural Network was built using images of the Spectrogram as 2D representation of the audio.

## 1.1 Sound and Features

Sound is a physical variation in pressure that propagates through a transmission medium over time. To process the sound with machine learning, it first must be converted to a digital format. The sound, that can be seen as acoustic data, is converted to analog electric signals by a microphone and then digitized using an Analog to Digital Converter (ADC).

Sound Waves are digitized by sampling them at discrete intervals of time. Diving the number of intervals by time it took to take them, one gets a number known as the sampling rate, which is typically of 44.1KHz for CD quality audio, meaning samples are taken 44,100 times per second. Each sample is the amplitude of the sound wave at a particular point in time, where the bit depth determines how detailed the sample will be. This is also known as the dynamic range of the signal (typically 16bit, which means a sample can range from 216 = 65,536 amplitude values). In the representation on Figure 2.3, it can be observed that sound is a one-dimensional sequence of numbers that represent the amplitude values of the sound wave at consecutive points in time, sometimes referred to as a waveform as it is shown in figure 1 and 2.

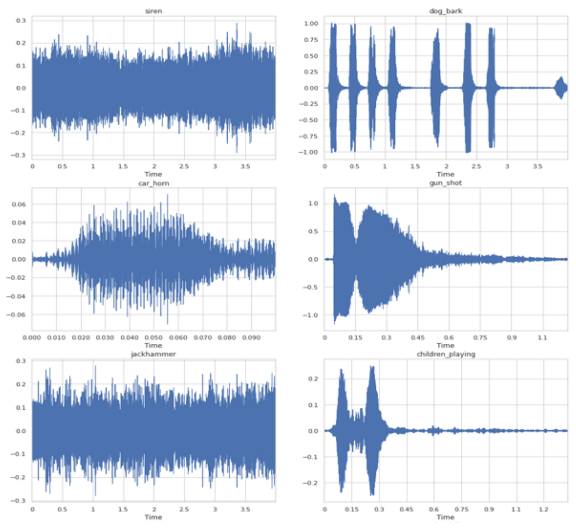

*Figure 1 Audio waveform plots*

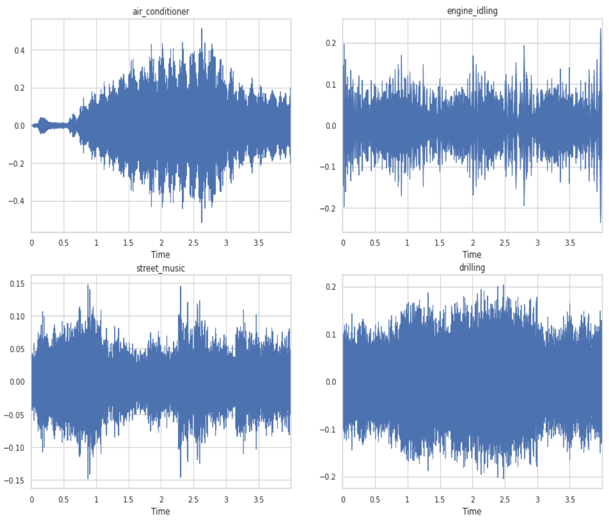

*Figure 2 Audio waveform plots*

Some of the main features that characterize sounds were considered when building the models. Here, they are briefly described.

### 1.1.1 Mel Frequency Cepstral Coefficients (MFCCs)

They are considered the most-used coefficients. The information of the rate of change in spectral bands of a signal is 43iven by its cepstrum. A cepstrum is basically a spectrum of the log of the spectrum of the time signal. The Mel-Frequency Cepstral Coefficients (MFCCs) are nothing but the coefficients that make up the Mel-frequency spectrum.

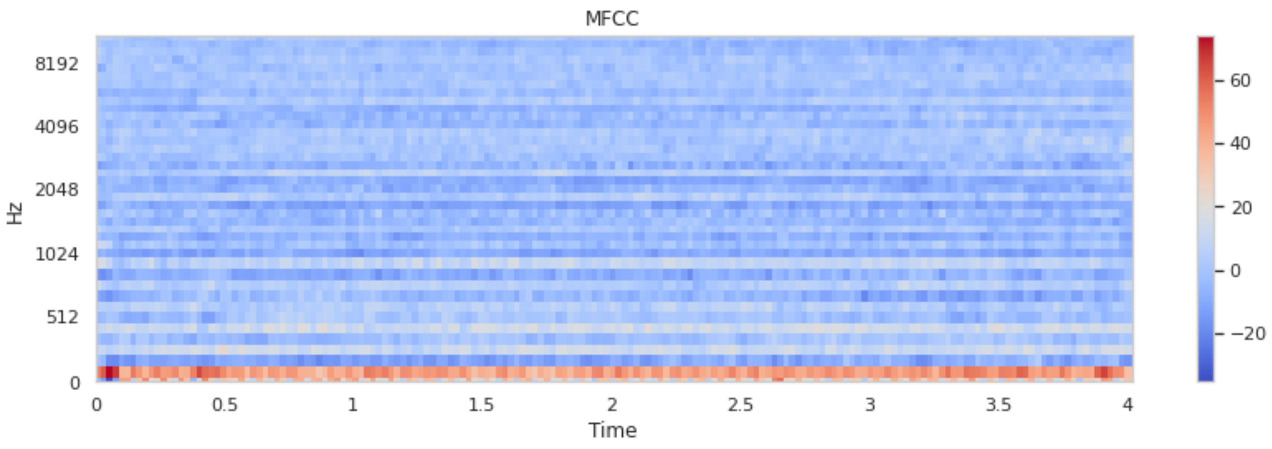

*Figure 3 MFCCs feature plot*

### 1.1.2 Chroma Features (CHROMA)

In music, the term chroma feature or chromagram closely relates to the twelve different pitch classes. Chroma-based features, which are also referred to as "pitch class profiles", capture harmonic and melodic characteristics of music, while being robust to changes in timbre and instrumentation. In particular, the chroma vector is a 12-element representation of the spectral energy, where the bins represent the 12 equal-tempered pitch classes of western-type music (semitone spacing).

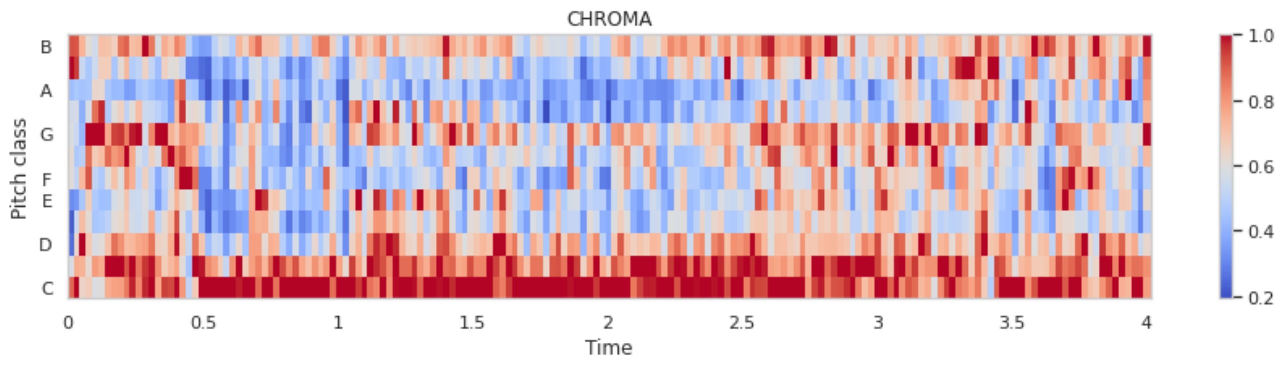

*Figure 4 CHROMA feature plot*

### 1.1.3 Spectral Centroid (SC) and Spectral Bandwidth (SB)

The Spectral Centroid provides the center of gravity of the magnitude spectrum. In other words, it gives the frequency band where most of the energy is concentrated. Mathematically, the spectral centroid is the weighted mean of the frequency bins.

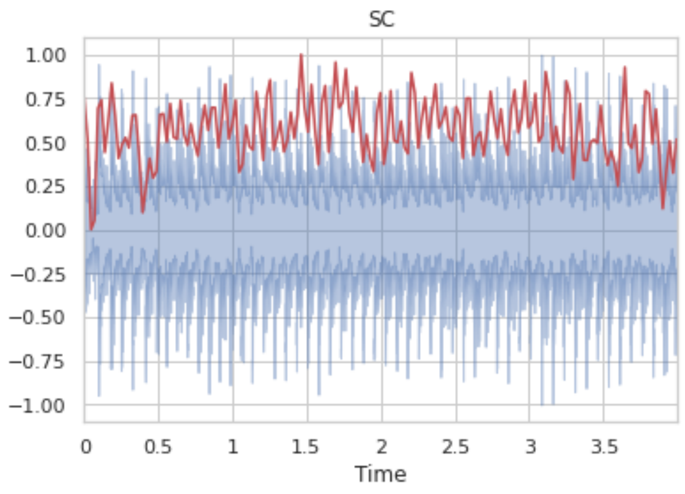

*Figure 5 Spectral Centroid plot*

The spectral bandwidth or spectral spread is derived from the spectral centroid. It is the spectral range of interest around the centroid, that is, the variance from the spectral centroid. The bandwidth is directly proportional to the energy spread across frequency bands. Mathematically, it is the weighted mean of the distances of frequency bands from the Spectral Centroid.

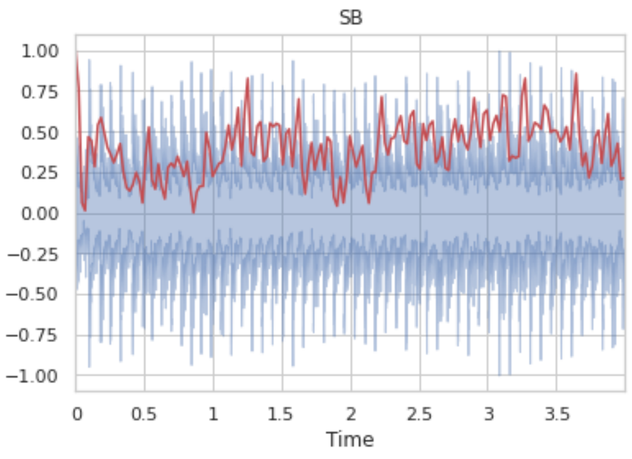

*Figure 6 Spectral Bandwidth plot*

### 1.1.4 Zero-Crossing Rate (ZCR)

It is simply the number of times a waveform crosses the horizontal time axis.

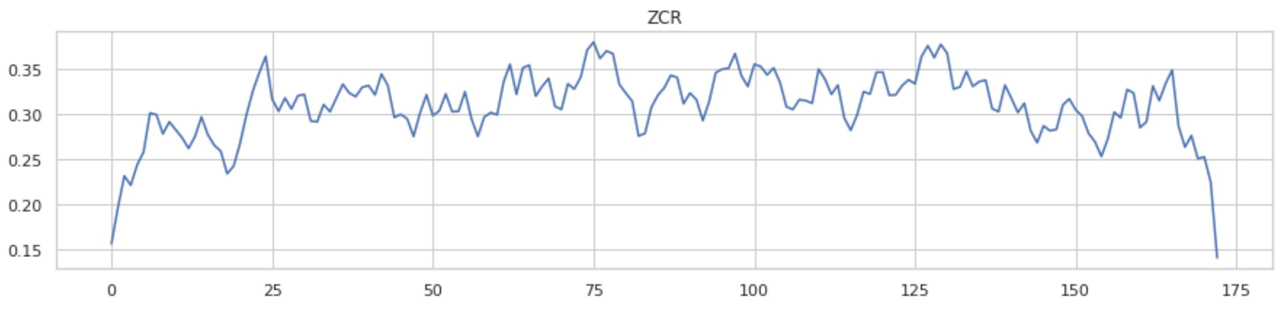

*Figure 7 Zero Crossing Rate plot*

### 1.1.5 Root-Mean-Square Energy (RMSE)

It is based on all samples in a frame. It acts as an indicator of loudness, since higher the energy, louder the sound. It is not very sensitive to outliers.

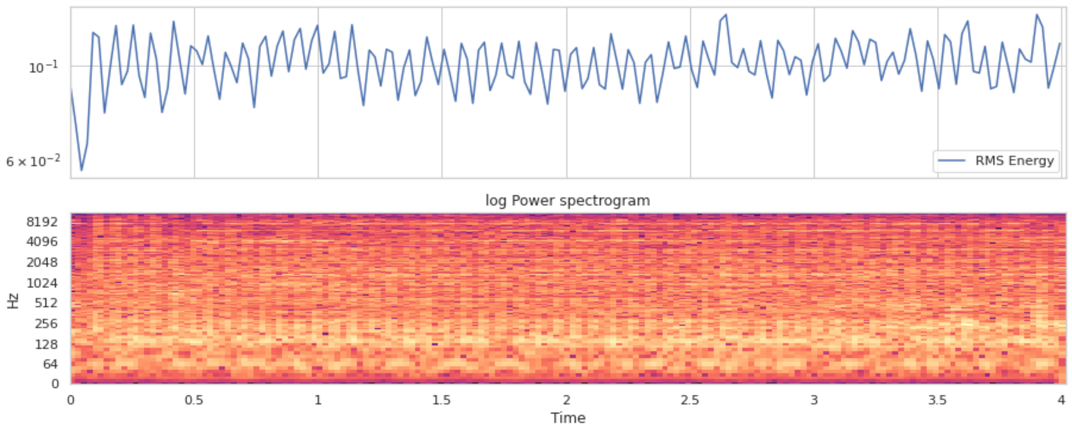

*Figure 8 RMSE plot*

## 1.2 Paper Structure

The project is structured as follows:

In chapter II, the dataset composition is outlined, and its characteristics are summarized. In chapter III, the fundamental steps to prepare the data for the analysis are run through as well as the process of selecting the features for an alternative analysis. In chapter IV, the machine learning models are presented along with the architecture choice, the hyperparameters tuning and the performances. In chapter V, the conclusions of the study are exposed with the results, the inherent limitations and the recommendations for future work.

# 2. Dataset and Exploratory Data Analysis (EDA)

In this section, the original structure of the dataset *UrbanSound8k* is decomposed in detail. In the meanwhile, the findings of the Exploratory Data Analysis (EDA) are shared.

## 2.1 Dataset Overview

The UrbanSound8k contains 8732 audio files in the format .wav divided in 10 classes, each one representing a different type of city sound, for instance, we can find car horns, dogs barking, sirens, etc.

### 2.1.1 Duration

In sound we have a strict relation between frequencies and amplitudes over time and having a dataset of varying sound durations means varying input vector sizes. To overcome this problem, padding is used: for all the samples of length less than the maximum length, zeros are inserted until it reaches the maximum value. This is a commonly used technique, and it does not alter the performance of the network. Each audio clip is of length <= 4s and the distribution can be seen below.

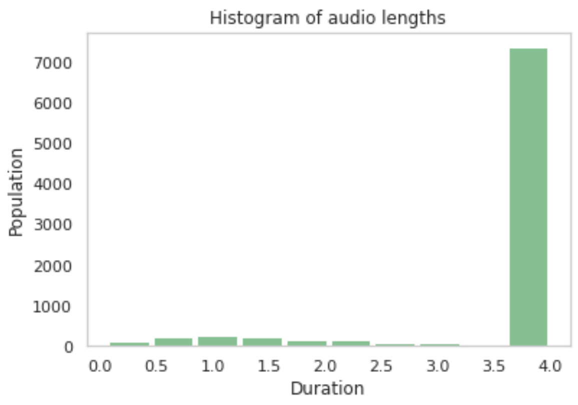

*Figure 9 Duration distribution*

### 2.1.2 Bit Depth

Different bit depths mean higher resolution but also different scales. The 66% of the dataset has a bit depth of 16 while 32% has a bit depth of 24%.

### 2.1.3 Sampling Rate

Higher sample rates also mean wider frequency ranges. The higher the sample rate used to record a sound, the more frequencies being captured. More precisely, according to the Nyquist Theorem, the highest frequency that can be captured in a recording is equal to the recording sample rate divided by 2. Most of the recognizable changes in sound occurs in the lower frequencies such as 22kHz. The sampling rate of the dataset varies from 8kHz to 192kHz, with 44.1kHz that accounts for 62% of the clips.

### 2.1.4 Maximum Amplitude

Each sound file will usually have a different maximum point of amplitude. Ideally, the amplitude values should be normalized to fit the same scale between all samples.

### 2.1.5 Channels

A channel is a representation of sound coming from or going to a single point. A single microphone can produce one channel of audio, and a single speaker can accept one channel of audio, for example. 91% of the audio files in the dataset has 2 channels (stereo).

### 2.1.6 Class Imbalance

As shown from the figure below, there is an imbalance in the dataset because of the classes *car_horn* and *gun_shot*. While the latter might occur less frequently per se, the recordings could depend also on the zone in which the microphone was positioned. This can lead to poor performances on these two categories; therefore, it is taken into consideration while training.

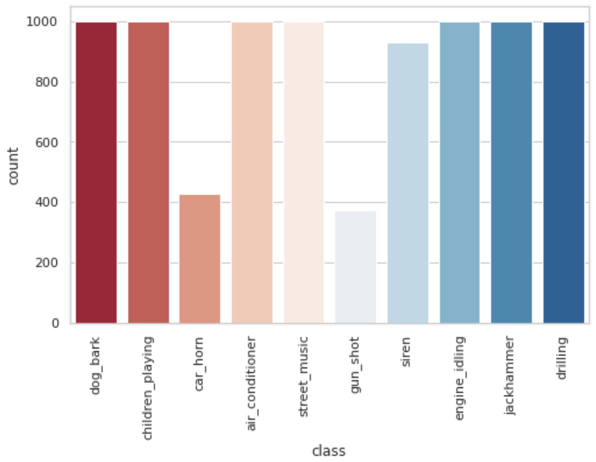

*Figure 10 Class imbalance*

# 3. Feature Extraction and Data Pre-processing

The data set is already shuffled and separated into 10 folds for a 10-fold cross validation and the authors of the data recommend not reshuffling the data during training due to the distribution of the sound classes within each fold.

*Training Set:* the fold one, two, three, four and six are taken

*Test Set:* the fold five, seven, eight and nine are taken

However, the data provided by audio files cannot be understood by the models directly. Feature extraction is a very important part in analyzing and finding relations between different things(cambia things). The presented methodology is composed of two main parts. The first part is to extract relevant features from audio files in the form of raw data. The second part is to represent the features of audio through images. Afterwards, both approaches are pre-processed in order to be fed to the neural networks.

## 3.1 Feature Extraction as Raw Data

Extracting features from audio files is not straightforward, nonetheless there are a collection of audio characteristics that are commonly used in audio machine learning applications that were presented in chapter I.

To extract information from audio files Librosa is used. With the help of Librosa Library following features are extracted:

1. *Mel-Frequency Cepstral Coefficients:* the first 40 coefficients are extracted (Mean and Standard deviation)
2. *Chromagram:* the first 12 chromas are considered (Mean and Standard deviation)
3. *Root-Mean-Square Error*
4. *Zero-Crossing Rate*
5. *Roll-off Frequency*
6. *Spectral Centroid and Spectral Bandwidth*

Each feature consists of an array of arrays containing measurements. This approach resulted in 114 components feature vectors.

## 3.2 Raw Data Pre-processing

Since the ranges among the feature vectors components vary too much, standardization with the *StandardScaler* function from scikit-learn package is applied. This function centered the dataset with mean approximately 0 and standard deviation 1.

The labels which represent the class of sounds are encoded from categorical to numerical data type. As a further step, Principal Component Analysis (PCA) is performed to have an additional perspective. From 114 features, the dimensions were reduced to 90 with the 90% variance criterium.

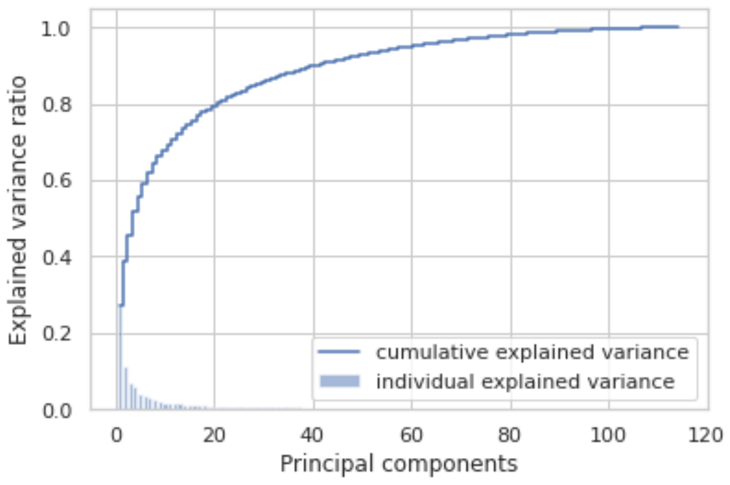

*Figure 11 PCA variance explanation*

## 3.3 Feature Extraction as Images

The main idea is to use audio features and consider them in a 2-D space, where the value of a single cell can be viewed as a pixel. The Short-Time Fourier Transform (STFT) is a powerful general-purpose tool for audio signal processing. It defines a particularly useful class of time-frequency distributions which specify complex amplitude versus time and frequency for any signal.

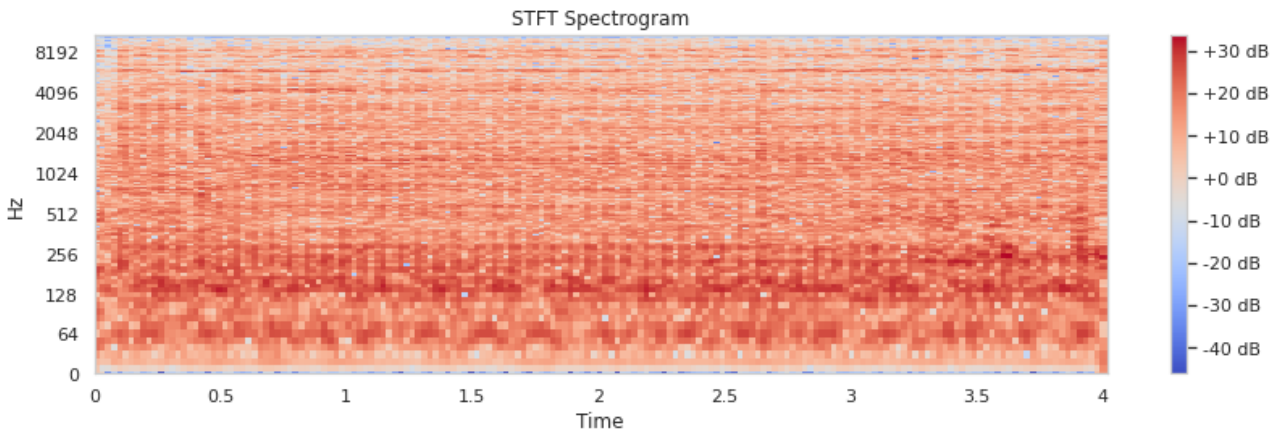

*Figure 12 STFT spectrogram*

## 3.4 Image Pre-processing

An image has three or one channel: red, green and blue, or grey. Image classification is a time-consuming task; therefore, images are converted to greyscale in order to reduce the weight. In particular, the use of the Librosa library in combination with the cv2 library allowed the transformation of the spectrum images such as scaling pixel values to prevent too wide ranges and resizing in order to have same length arrays (necessary to feed the neural network).

# 4. Model Definition

This Section gives an overview of the models used on the different training sets. First, Feedforward Artificial Neural Network (ANN) concept is used to create several perceptrons, while the Convolutional Neural Network (CNN) concept is used on the image dataset to create different combinations of layers. For both models we present the architecture, the initial results and the hyperparameter tuning phase. To build the actual models *TensorFlow* and *Keras* libraries are used.

## 4.1 Multi-layer Perceptron (MLP)

It is a neural network where the mapping between inputs and output is non-linear. A Multilayer Perceptron has input and output layers, and one or more hidden layers with many neurons stacked together. Neurons in a Multilayer Perceptron can use any arbitrary activation function. Multilayer Perceptron falls under the category of feedforward algorithms, because inputs are combined with the initial weights in a weighted sum and subjected to the activation function. Each layer is feeding the next one with the result of their computation, their internal representation of the data.

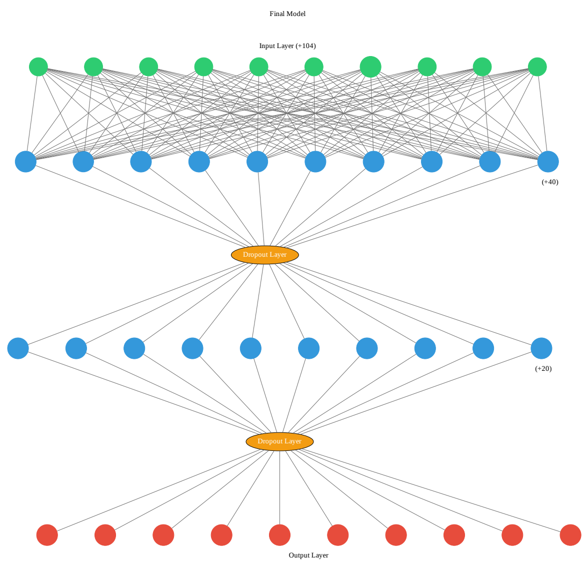

*Figure 13 Final MLP Structure*

The final model used for predicting raw data is a multilayer perceptron with dropout layers and custom weights to compensate for class imbalance.

### 4.1.1 Network structure

The starting point for the neural network structure is a reasonable network in terms of hidden neurons to prevent over-fitting, indeed a high number of units in the hidden layers would end up in learning too much from the dataset, leading to poor performances on the test sets. For this reason, the rule of thumb followed to decide hidden neurons quantity is the following:

$$\#Hidden\ neurons = 2/3\ (\#Input\ neurons + \#Output\ neurons)$$

The next step is to decide the hidden layers number. As using the rule presented above gives a quite small number of units, only two layers are considered, indeed, a higher quantity would mean having a real small number of neurons per layer. Applying this rule ended up in the following architectures on the 2 different training sets, where the output layer is fixed at 10:

| Training set | Input | 1st Hidden | 2nd Hidden |
|---|---|---|---|
| 114 features scaled | 114 | 50 | 30 |
| 90 features reduced with PCA | 90 | 40 | 30 |

### 4.1.2 Hyperparameter Tuning

#### *4.1.2.1 Optimizer*

Optimizers can be explained as a mathematical function to modify the weights of the network given the gradients and additional information, depending on the formulation of the optimizer. Optimizers are built upon the idea of gradient descent, the greedy approach of iteratively decreasing the loss function by following the gradient.

For this project, *Adam* optimizer was chosen. Adam combines the good properties of *AdaDelta* and *RMSprop*, and it is suited for most problems. Although *Stochastic Gradient Descent* might generalize better, *Adam* is more efficient and thus converges faster, giving more flexibility when working with large dataset and many variables.

In particular, the *learning_rate* that was found to be the best is 0.001 with other parameters left at default.

#### *4.1.2.2 Initializer*

Weight initializers represent how the initial values of a neural network layer's weight matrix are set. The initializer function chosen for the project is *truncated_normal*. It initializes weights with uniform distribution with mean = 0 and stddev = 0.05, however the values more than 2 stddev from the mean are discarded and redrawn.

### *4.1.2.3 Activation and loss functions*

The activation function in a neural network defines how the weighted sum of the input is transformed into an output from a node or nodes in a layer of the network. The choice for the activation function of hidden layers fell on the *Rectified Linear Units (ReLU),* which attempts to resolve the vanishing gradient problem. The output layer uses a *SoftMax* activation function which is used for multiclass classification problems.

The loss function in a neural network quantifies the difference between the expected outcome and the outcome produced by the model. *Sparse Categorical Crossentropy* loss was chosen as it is well suited for multiclass classification. Since the classes are integers and not one-hot encoded, the sparse version is preferred. The same reasoning was applied in considering the accuracy evaluation metric.

### *4.1.2.4 Epochs and batch size*

An epoch means training the neural network with all the training data for one cycle. An epoch is made up of one or more batches of the same size. To prevent overfitting, the model was trained first on 300 epochs, which represent the general rule of thumb to start with 3 times the input size. The batch size was 128. Different combinations were tested, and the most promising one was with 400 epochs and a batch size of 64.

### *4.1.2.5 Class Weights*

The idea is to penalize errors on not well represented classes. The following formula is applied with the function *compute_weight_class* from sklearn package:

$$w_i = \frac{\#total\ samples}{\#classes * \#samples\ of\ class\ i}$$

Weights were assigned to the classes to compensate for the imbalance of the distribution as follow:

| Classes | Weight |
|---|---|
| 0 | 0.8998 |
| 1 | 2.1630 |
| 2 | 0.8998 |
| 3 | 0.8998 |
| 4 | 0.8998 |
| 5 | 0.8702 |
| 6 | 2.3679 |
| 7 | 0.8210 |
| 8 | 0.8394 |
| 9 | 0.8998 |

As expected, larger weights were assigned to less numerous classes, in this way the model focuses on reducing these errors. This procedure resulted in a more precise definition of loss and accuracy.

### 4.1.3 Training Set Results

| Models | Train Accuracy | Train Loss |
|---|---|---|
| Single-Layer Perceptron (with Original Features) | 99% | 0.00043 |
| Single-Layer Perceptron (with PCA Selected Features) | 99% | 0.00083 |
| Multi-Layer Perceptron (with Original Features) | 99% | 0.00038 |
| Multi-Layer Perceptron (with PCA Selected Features) | 99% | 0.00048 |
| Multi-Layer Perceptron with Dropout (with Original Features) | 96% | 0.11583 |
| Multi-Layer Perceptron with Dropout (with PCA selected Features) | 95% | 0.15271 |
| Multi-Layer Perceptron with Dropout and Weights (with Original Features) | 93% | 0.34400 |
| Multi-Layer Perceptron with Dropout and Weights (with PCA selected Features) | 69% | 0.74905 |

As reported in the table above, the best performer was the multi-layer network with dropout. This is because the first models are clearly biased and overfitted. But if we consider the imbalance of the dataset, then the multi-layer perceptron with dropout and class weights is the one that best represent our problem. The accuracy and loss curves reflect how the curve should look like.

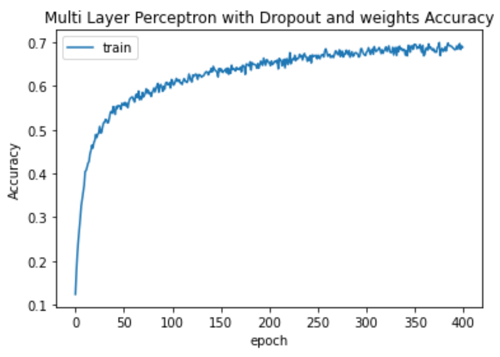

*Figure 14 Final MLP accuracy plot*

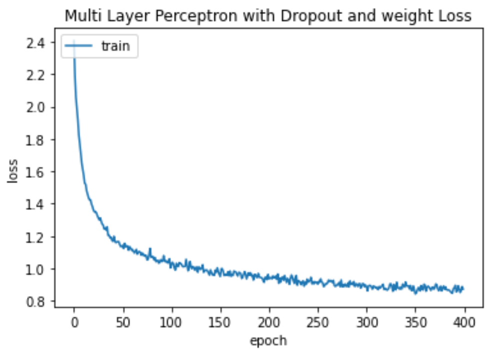

*Figure 15 Final MLP loss plot*

Note that all the random seeds used by TensorFlow are fixed to make results reproducible. This step is necessary as many parameters initial value is random, for instance, the neural network weights.

## 4.2 Convolutional Neural Network (CNN)

In the case of CNN, convolution is applied to the input data where the filter, called Kernel, scans the information and produces a feature map. To perform convolution, the kernel goes over the input image, doing matrix multiplication element after element. The result for each receptive field is written down in the feature map. Then, the arrays which contain pixels are flattened and fed to the perceptron part of the model, which classifies the input into 10 classes.

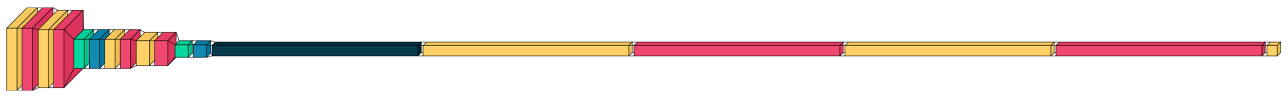

*Figure 16 Final CNN structure*

The final model used for predicting images is a convolutional neural network with pooling, batch normalization, dense and dropout layers.

### 4.2.1 Network structure

The structure of a neural network for image classification consists of convolutional layers followed by pooling layers and densely connected ones at the end before the output layer. The idea is the one of image compression: starting from the images the layers are more and more compressed into lower dimensional ones.

Several models were created and implemented successfully as all-purpose models. A hidden pitfall of these models is that they are quite complex and therefore they should be trained for long time. As a consequence, we tried to implement our own model starting from a simple one towards more complex ones. The final model is *ConvConv - Pool - ConvConv - Pool* model with *Dropout* which derives from the VGG16 architecture:

[INPUT] →

→[CONV 1] → [ReLU] → [CONV2] → [ReLU]

→ [POOL 1] → [BATCH NORM 1]

→[CONV 3] → [ReLU] → [CONV4] → [ReLU]

→ [POOL 2] → [BATCH NORM 2]

→ [FLATTEN]

→ [DENSE 1] → [DROPOUT 1]

→ [DENSE 2] → [DROPOUT 2]

→ [OUTPUT]

### 4.2.2 Hyperparameter and Layers Tuning

#### *4.2.2.1 Kernel or Filter Size*

In a convolution, a convolution filter slides over all the pixels of the image taking their dot product. The linear combination of the pixels weighted by the convolutional filter extracts features from the image. Most of the useful features in an image are usually local, and therefore, few local pixels at a time should be taken to apply convolutions. However, the filter must be big enough to capture the features, especially in the first layers (otherwise they would be lost since the beginning). Layers early in the network architecture (i.e., closer to the actual input image) learn fewer convolutional filters while layers deeper in the network (i.e., closer to the output predictions) will learn more filters. Notice that the output spatial volume is decreasing while the number of filters learned is increasing. Additionally, odd-sized filters symmetrically divide the previous layer pixels around the output pixel. Based on these notions, a filter size of 5x5 is chosen for the first 3 convolution while a smaller 3x3 filter is chosen for the 4th one in order to ease information channeling (learn high level features first and then the details).

#### *4.2.2.2 Activation Functions*

As for the multi-layer perceptron, *Rectified Linear Unit (ReLU)* is applied. This activation function not only handles the non-linearity of the image pixels, but also helps to prevent the exponential growth in the computations.

#### *4.2.2.3 Pooling*

Pooling layers provide an approach to down sampling feature maps by summarizing the presence of features in patches of the feature map. Down sampling is required to address a limitation of feature mapping: small movements in the position of the feature in the input image will result in a different feature map. *Maximum Pooling* is applied to the CNN model which calculates the maximum value for each patch of the feature map.

#### *4.2.2.4 Batch Normalization*

To address overfitting during training, the common technique of *Batch Normalization* is applied before the non-linear function. *Batch Normalization* is similar to *Dropout* in the multi-layer perceptron, which is then applied in the dense part of the network. This process makes the neural network faster and more stable through normalization of the batches and handles internal covariate shifts well.

#### *4.2.2.5 Dense Structure*

The reasoning used in tuning the layers of the dense structure of the CNN is similar to the one used for the MLP. It is summarized below:

| Dense Layers Input | Activation F. | Dropout |
|---|---|---|
| 6400 | ReLU | 0.5 |
| 4000 | ReLU | 0.5 |
| 10 | SoftMax | |

*Adam* optimizers is chosen again with a learning rate = 0,001 (other parameters at default). Loss and Accuracy functions are *SparseCategoricalCrossentropy* as output is multi-label. The model is trained for 50 *epochs* with a *batch size* of 64 after several experiments.

### 4.2.3 Training Set Results

| CNN Models | Train Accuracy | Train Loss |
|---|---|---|
| Con-Pool-Conv-Pool | 66% | 0.8910 |
| Conv-Conv_Pool-Conv-Conv-Pool | 95% | 0.1650 |
| Conv-Conv-Pool-Conv-Conv-Pool with Dropout | 72% | 0.8041 |

While the best performance is achieved by the second architecture, the third one with dropout should be preferred because of overfitting. The accuracy and loss curves have an ideal shape:

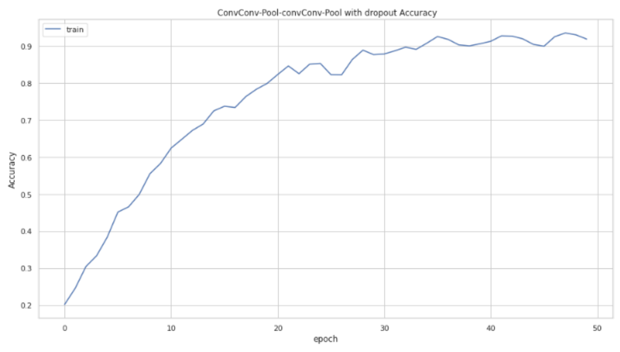

*Figure 17 Final CNN accuracy plot*

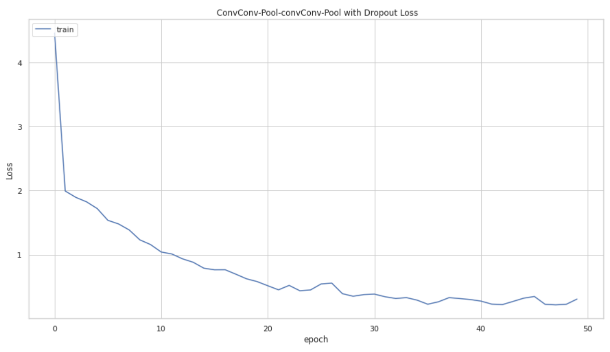

*Figure 18 Final CNN loss plot*

However, training a CNN is very expensive compared to the MLP neural networks. In fact, experimenting with different architectures and hyperparameters configurations is reduced to a great extent. Furthermore, we must keep in mind that the images were compressed to a very small dimension due to the processing capacity of the instruments we have used, resulting in a reduced quality. Compared to the MLP with dropout and weights, the results obtained with CNN are worse by far.

## 5. Conclusions

This last section presents the results obtained on the various test sets by the best performing models. Then are outlined the main limitations of this paper as well as the recommendations for future works.

### 5.1 Test Results

The test sets are composed by the folds within the *UrbanSound8k* that are fold 5, fold 7, fold 8, fold 9 and fold 10. The test sets for both types of neural networks are pre-processed following respectively the same procedures of the training sets.

The MLP with Dropout Model performed better for the classification. The average accuracy and standard deviation of MLP and CNN model is as follows:

| Model | Avg. Accuracy | Avg. Standard Deviation |
|---|---:|---:|
| MLP with Dropout & weights | 57% | 0.0165 |
| CNN_3 | 39% | 0.0550 |

The below table depicts the accuracy of MLP and CNN_3 Model for each test fold.

| Test Dataset | MLP w/Dropout Accuracy | MLP w/Dropout Loss | CNN_3 Accuracy | CNN_3 Loss |
|---|---:|---:|---:|---:|
| Fold 5 | 53.90% | 3.920 | 37.8% | 4.156 |
| Fold 7 | 57.50% | 2.392 | 38.7% | 2.279 |
| Fold 8 | 58.90% | 7.323 | 51.7% | 1.888 |
| Fold 9 | 57.40% | 12.038 | 47.5% | 2.771 |
| Fold 10 | 57.40% | 2.533 | 46.8% | 1.976 |

As further method to evaluate the performance of the models, confusion matrices are computed as well as different metrics such as precision and recall. This is helpful because we can find out if the final models struggle to categorize among different types of sound.

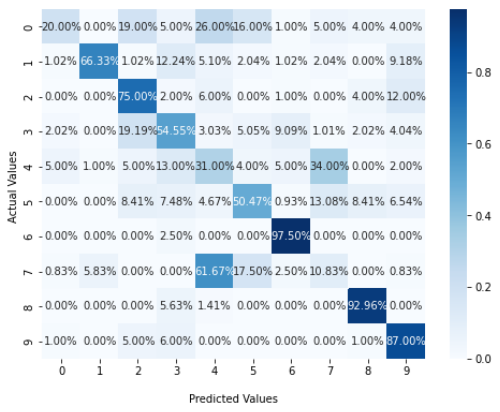

*Figure 19 MLP Confusion Matrix of test set*

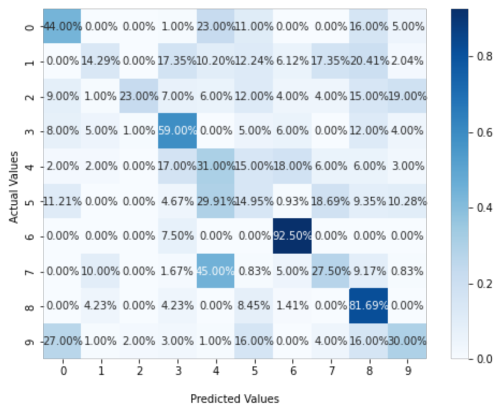

*Figure 20 CNN Confusion Matrix of test set*

As we can see, both models can predict with significant precision the categories of *gun_shot* and *jackhammer*.

## 5.2 Limitations and Recommendations for future work

There are two main limitations in this paper. The first one being what features of sound are considered and their variations. This limitation is proportional to the knowledge of the field. The other limitation is attributed to the tools used to conduct the research, which have limited capabilities in a limited time window, especially when it comes to train CNN models (usually trained for weeks).

Regarding the dataset, we considered the most popular features along their mean and standard deviation, however more features such as Tonnetz and for example the minimum and the maximum could have been considered. This is also directly related to *Data Augmentation*: it helps to generate synthetic data from existing data set such that generalization capability of model can be improved. Techniques such as noise injection, shifting time, changing pitch and speed should be implemented to have a more complete dataset in order to interpret correctly sounds directly from the environment. In fact, it should be taken into consideration that the dataset itself is imperfect because the audio files derive from different sources with different qualities (due to the digitalization). Regarding the machine learning part of the research, more computational power combined with a proper knowledge of the field would allow to construct a more suited and complex architecture for CNN, which should be trained for the proper amount of time. This will also allow to consider images with more channels (colors) and bigger size for increased learning capacity. A different approach with other types of neural networks such as *Recurrent Neural Network* could be also attempted. Additionally, we implemented PCA for dimensionality reduction, but it might not be indicated as the achieved performances are limited. Instead, other feature selection techniques could be attempted.

In conclusion, although our findings suggest that a multi-layer perceptron is more proficient in analyzing sounds with constraints in time and tools, we believe that with the right conditions a convolutional neural network might outperform it.

## 6. References

[1] Urban Sound Dataset. *https://urbansounddataset.weebly.com/urbansound8k.html*

[2] Wikipedia. Multilayer perceptron. *https://en.wikipedia.org/wiki/Mu ltilayer_perceptron.*

[3] Wikipedia. Stochastic gradient descent. *https://en.wikipedia.org/wik i/Stochastic_gradient_descent.*

[4] Wikipedia. Convolutional neural network. *https://en.wikipedia.org/w iki/Convolutional_neural_network.*

[5] Wikipedia. Cross validation. *https://en.wikipedia.org/wiki/Cross- validation_(statistics).*

[6] Wikipedia. Softmax function. *https://en.wikipedia.org/wiki/Softma x_function.*

[7] Wikipedia. Vanishing gradient problem. *https://en.wikipedia.org/wiki/Vanishing_gradient_problem.*

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

[9] Tensorflow. *https://www.tensorflow.org.*

[10] Keras. *https://keras.io.*

[11] Numpy. *https://numpy.org.*

[12] Pandas. *https://pandas.pydata.org.*

[13] Scikit learn. Standard scaler. *https://scikit-learn.org/stable/modu les/generated/sklearn.preprocessing.StandardScaler.html.*

[14] Scikit Learn. Pca. *https://scikit-learn.org/stable/modules/gene rated/sklearn.decomposition.PCA.html.*

[15] Scikit learn. Compute class weight. *https://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.compute_class _weight.html.*

[16] PCA. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html?highlight=pca#sklearn.decomposition.PCA

[17] Audio Features. *https://devopedia.org/audio-feature-extraction*

[18] Use of tempogram. *https://ieeexplore.ieee.org/document/7178003*

[19] Sound Features. *https://dosits.org/science/advanced-topics/introduction-to-signal-levels/*

[20] SoundFile. *https://pysoundfile.readthedocs.io/en/latest/*

[21] Pydub. *https://pypi.org/project/pydub/*