Republic of Tunisia
Ministry of Higher Education and Scientific Research
University of Tunis El Manar
National Engineering School of Tunis

ICT Departement

End of year Project

Environmental sounds classification

Realized by

Khaoula Kadri

Selim Oualha

Class : 2^{nd} year IT 1

Supervised by

Mrs.Sonia Djaziri-Larbi

Defended On April 27th 2019

The Jury :

President	: Mr. Wady Naanaa
Reporter	: Mr. Ramzi Haddad

Academic Year 2018/2019

Acknowledgements

We would like to express our deepest gratitude and appreciation to our tutor Mrs Sonia Djaziri-Larbi for her constructive advices and supervising our work and for all the efforts she made along this project.

We also thank all the teachers that encouraged us and helped us during this period and to the members of jury for accepting to evaluate our project.

Finally, we want to express our best regards for everyone who might helped us through inspiring us to pick this topic or provide some advices to guide us and keep us on track and to all who had a hand in the realisation of this project.

Abstract

This project presents an overview on the sound classification concept that consist in assigning a label to an unknown audio signal. This task is based on extracting relevant features from audio signals and using them to identify to which class the sound most likely belongs to.

For this aim, we first extracted Mel Frequency Cepstral Coefficients features from a sound data. After performing this step, we feed these features to a Convolutional Neural Network model to classify the audio signal into separate classes.

Key Words : Mel Frequency Cepstral Coefficients - Audio Features - Classification - Deep learning - Convolutional Neural Networks

Table of contents

List of Figures v

Introduction viii

1 Audio classification 1
- 1.1 Introduction . . . 1
- 1.2 Audio Data . . . 1
- 1.3 Audio features . . . 2
  - 1.3.1 Mel Frequency Cepstral Coefficients (MFCC) . . . 2
    - 1.3.1.1 Mel Scale . . . 3
    - 1.3.1.2 Framing and Windowing . . . 4
    - 1.3.1.3 Fast Fourier Transform and power spectrum . . . 5
    - 1.3.1.4 Mel FilterBank . . . 5
    - 1.3.1.5 Mel Coefficients . . . 6
  - 1.3.2 MFCC implementation in Python . . . 7
- 1.4 Libraries . . . 7
  - 1.4.1 Librosa . . . 7
  - 1.4.2 PyAudioAnalysis . . . 8
  - 1.4.3 Essentia . . . 8
- 1.5 Audio classification applications . . . 8
- 1.6 Conclusion . . . 9

2 Convolutional Neural Networks 10
- 2.1 Introduction . . . 10
- 2.2 Artificial Neural Network . . . 10
  - 2.2.1 Brain Neural Network . . . 10
  - 2.2.2 Artificial Neural network . . . 10
  - 2.2.3 Gradient descent . . . 11
  - 2.2.4 Back-propagation . . . 13
    - 2.2.4.1 Our Neural Networks variables . . . 13
    - 2.2.4.2 The four fundamental equations of back-propagation . . . 13
    - 2.2.4.3 Training a model using back-propagation . . . 15
  - 2.2.5 The activation function . . . 15
    - 2.2.5.1 Linear activation function . . . 15
    - 2.2.5.2 Non-linear activation function . . . 16
  - 2.2.6 Cost function . . . 17
    - 2.2.6.1 Regression losses . . . 17

2.2.6.2 Classification losses . . . 18
2.2.7 Optimizer . . . 18
2.2.7.1 Gradient Descent variants: . . . 19
2.2.7.2 Gradient Descent optimization algorithms . . . 19
2.2.7.3 Which optimizer to use ? . . . 20
2.3 Convolutional Neural Networks . . . 20
2.3.1 General Architecture of CNN . . . 20
2.3.2 Convolutional layers . . . 21
2.3.2.1 Stride . . . 22
2.3.2.2 Padding . . . 22
2.3.3 Pooling layer . . . 22
2.3.4 Dropout Layer . . . 23
2.3.5 Fully connected layer . . . 24
2.4 Libraries . . . 24
2.4.1 TensorFlow . . . 24
2.4.2 Keras . . . 25
2.5 Conclusion . . . 25

3 Implementation **26**
3.1 Presentation of the project . . . 26
3.2 Data set . . . 26
3.3 Data preprocessing . . . 26
3.4 Features extraction . . . 28
3.5 ML or DL . . . 29
3.6 DL model . . . 32
3.7 Prediction results . . . 33
3.7.1 Data set ”instruments” . . . 33
3.7.2 Data set ”Environmental sounds” . . . 34
3.8 Conclusion . . . 35

Conclusion and perspectives **36**

Appendix **1**
.1 Activation functions . . . 1
.1.1 Sigmoid or Logistic function . . . 1
.1.2 Tanh . . . 1
.1.3 ELU(Exponential Linear Unit) . . . 2
.2 Momentum effect on the gradient descent . . . 2
.2.1 Momentum . . . 2
.2.2 Nesterov accelerated gradient: . . . 3
.3 Libraries . . . 4
.3.1 Pytorch . . . 4
.3.2 MXNet . . . 4

List of Figures

1.1 Plot of real audio signal [1] . . . 1
1.2 Audio features categories . . . 2
1.3 MFCC block diagram . . . 3
1.4 Mel scale [2] . . . 3
1.5 Hanning Window [3] . . . 4
1.6 Windowed signal shape [4] . . . 4
1.7 Power spectral density (Periodogram)[5] . . . 5
1.8 With 10 triangular filters [6] . . . 6
1.9 Visualisation of MFCC coefficients of a piano recording[7] . . . 6
1.10 Librosa logo[8] . . . 8
1.11 PyAudioAnalysis logo[9] . . . 8
1.12 Essentia logo[10] . . . 8

2.1 Analogy between the Biological Neuron and the Artificial Neuron [12] . . . 11
2.2 Neural Networks layers [13] . . . 11
2.3 Visualisation of different learning rate constant effect on the model [14] . . . 12
2.4 Visualisation of the ball rolling down in case of $\mathbb{R}^2$[15] . . . 12
2.5 Linear function [16] . . . 16
2.6 Relu function [16] . . . 17
2.7 LeakyRelu function [16] . . . 17
2.8 Fluctuations of the cost function [19] . . . 19
2.9 Comparison of Adam to Other Optimization Algorithms [21] . . . 20
2.10 CNN architecture example [22] . . . 21
2.11 Computing one convolution [23] . . . 21
2.12 Convolving an input image with a filter [24] . . . 22
2.13 One step stride [25] . . . 22
2.14 One border padding [26] . . . 23
2.15 Max Pooling example [27] . . . 23
2.16 Drop-out example [28] . . . 23
2.17 Over-fitting effect on a model [29] . . . 24

3.1 visualization of instruments length difference . . . 27
3.2 visualization of Environmental sounds length difference . . . 28
3.3 acoustic guitar wave form . . . 29
3.4 acoustic guitar MFCC . . . 29
3.5 ”writing” wave form . . . 29
3.6 ”writing” MFCC . . . 29

3.7 Difference between Machine Learning and Deep Learning 30
3.8 SVM confusion matrix for instruments database 31
3.9 DL model architecture 32
3.10 Instruments Confusion matrix 33
3.11 Environmental sounds confusion matrix 34

12 Sigmoid function [16] 1
13 Tanh function [16] 2
14 ELU function [16] 2
15 Momentum effect on SGD 3
16 Momentum update [14] 3
17 Nesterov Momentum update [14] 4

Introduction

Human beings can identify sounds easily. Indeed, recognizing the sound of closing door, telling the difference between a woman's voice and a man's voice, are daily and unconscious jobs for us thanks to our brain and cognitive learning.

However, the task gets harder when sounds seem to be very similar or when there is background noise.

Therefore, machines are built for such reasons by miming the brain into their computational systems. But they still have not caught up to a level where they can identify sounds accurately. Because unlike the advancement seen in Computer Vision, audio signal classification still under development.

In order to enhance sounds recognition we had to use image classification as a step to perform audio classification.

In this context, our project is based on extracting sounds features and representing them as an image using Mel Frequency Cepstral Coefficients (MFCC). Furthermore, the classification of sounds is based on their image representation using a Deep Learning architecture such as Convolutional Neural Networks (CNN), one of the most known algorithms in image classification.

This document contains three chapters which resume the technologies we used, the theoretical explanations and our work to approach our project goals.

The first chapter is dedicated to the extraction of audio features. It briefly describes the audio data and how to extract MFCC features from signals. It also presents the MFCC calculation steps, starting from windowing techniques to Mel FilterBank application.

The second chapter shows the significant improvement of Computer Vision using Deep Learning, especially CNN. Moreover, it presents specific details of CNN architecture, especially how it deals with images which made it the ultimate choice to work with in computer vision problems, audio classification, speech recognition, etc.

The last chapter presents our implementation of audio signal classification, using MFCC features. Then, based on these features we classify sounds by feeding them into a CNN model. The data base was too large which made us divide the audio samples into two main categories: 17 classes of instrumental audio sounds and 24 classes of environmental audio sounds.

Chapter 1

Audio classification

1.1 Introduction

The necessary requirements for successfully applying Machine Learning or Deep Learning depends on the problem being solved. In other words, it depends on data being processed.

In this chapter we present a brief overview on how important data preprocessing is. Besides, it holds about what is the main techniques used to extract audio features and the most used libraries in audio processing.

1.2 Audio Data

Audio data are like most of real-world-data, unstructured and large. They contain a lot of information most likely redundant and unclean which make audio classification complex.

These data are packaged in audio files with recognized format, more specifically, there are three major types of audio file formats, WAV (Waveform Audio File) format, WMA (Windows Media Audio) format and MP3 (MPEG-1 Audio layer 3) format. Moreover, the most used for tasks such as audio classification is WAV file which is uncompressed format that preserves the quality and prevents from any loss of information.

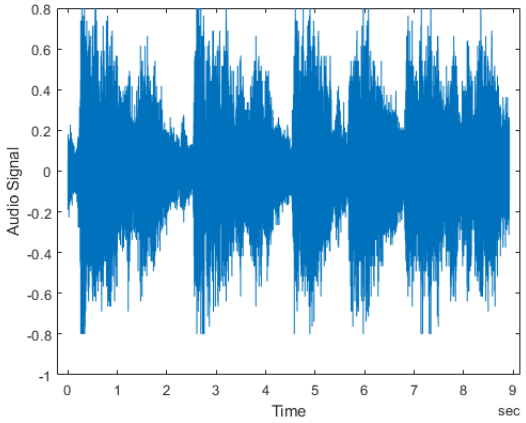


Figure 1.1: Plot of real audio signal [1]

1.3 Audio features

As mentioned previously, audio data are represented in a way that is not suitable for classification algorithms. In order to enhance the classification process, and in particular classification algorithm's accuracy, the first step should be pre-processing audio signals. This step allows cleaning data by removing noise, interferences and redundancies.

Features extraction is a mean of dimensionality reduction, as well extracting more useful information hidden in the signals by avoiding unnecessary or redundant informations.

The audio features that are used in classifying sounds are mainly divided into two categories: Frequency Domain Features and Time Domain Features.

Practically, time domain features are extracted directly from the samples of the audio signal, such as the short-term energy and short-term zero-crossing rate. Frequency domain features are extracted from spectral representation of the audio, such as the Fourier Transorm.

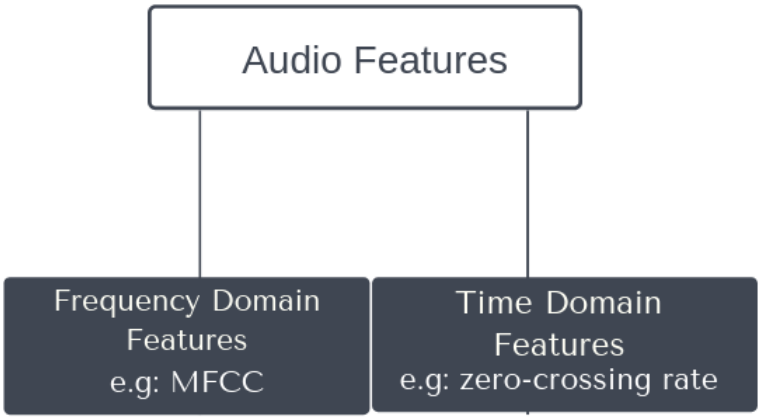


Figure 1.2: Audio features categories

Unlike time domain features that provide a simple way to analyse the audio, frequency domain features offer more precision in audio analysis. That's why we chose one of the most used frequency domain features, the Mel Frequency Cepstral Coefficients.

1.3.1 Mel Frequency Cepstral Coefficients (MFCC)

MFCCs, are widely used in automatic speech recognition and Music Information Retrieval(MIR). It is a prevalent feature first introduced by Davis and Mermelstein in the 1980[s], based on frequency domain using Mel scale.

Figure 1.3 shows the cycle of calculating MFCC coefficients from a given signal.

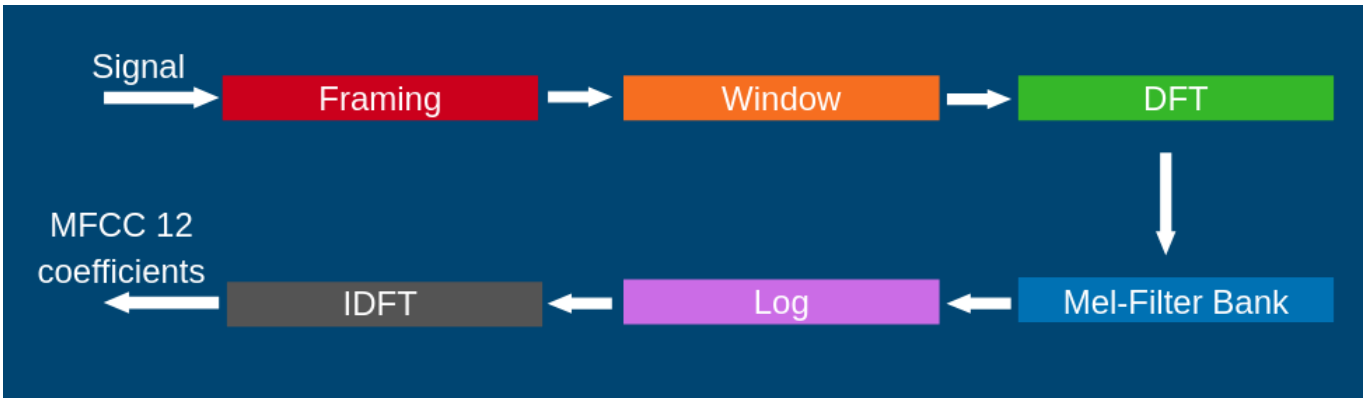


Figure 1.3: MFCC block diagram

This section will go over the main aspects of MFCCs, and how to implement them.

1.3.1.1 Mel Scale

Mel scale is a scale that reflects how the human ear perceive sound and relates that perception to its actual measured frequency. Its a psychoacoustic scale that was defined after doing numerous experiments on humans.

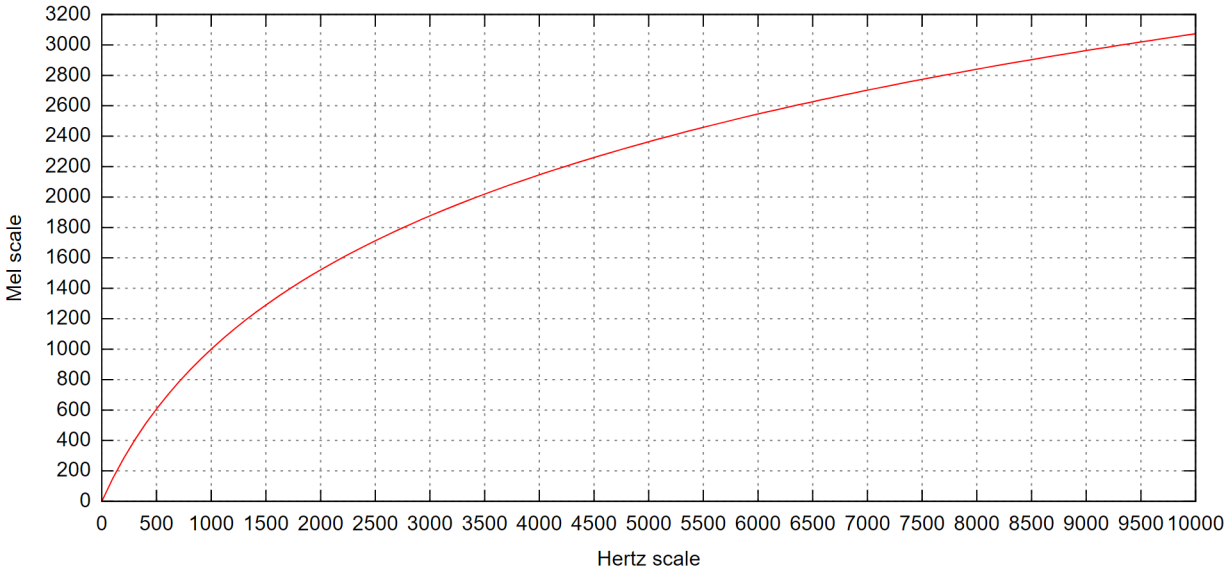


Figure 1.4: Mel scale [2]

As shown in figure 1.4, the human perception of sound is linear until 1000 Hz and logarithmic upwards. In other words, the human ear is better at capturing the small changes in pitch at lower frequencies than at high frequencies.

Following is a formula used in converting a frequency measured in Hertz (f) to a Mel scale:

$$Mel(f) = 2595log(1 + \frac{f}{700})$$

As its name tells Mel Frequency Cepstral Coefficients are based on Mel scale, they're actually calculated from a spectral representation of the audio in mel scale.

1.3.1.2 Framing and Windowing

The first step in calculating the MFCC is slicing the signal into short-time frames. Typically these frames are sized 20 to 24 ms with 50% overlap between neighbour frames.

After cutting the signal into frames, each frame is multiplied by a finite-length analysis window. The idea behind this operation is that a signal is constantly changing over time, so by applying the Fast Fourier Transform (FFT) directly on signal we would lose the shape of a spectral envelope of the signal over time by producing spectral leakage.

Actually windowing reduces the discontinuities of signal at the endpoints due to the amplitude of a window function that varies gradually to zero at the edges.

There are several different types of window functions, we mention Hamming and Hanning window.

Figure 1.5 below is a representation of a Hanning window, while figure 1.6 shows how signal is modified after applying the Hanning window.

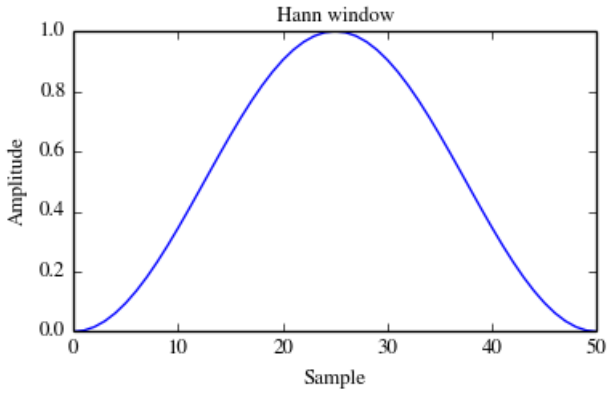


Figure 1.5: Hanning Window [3]

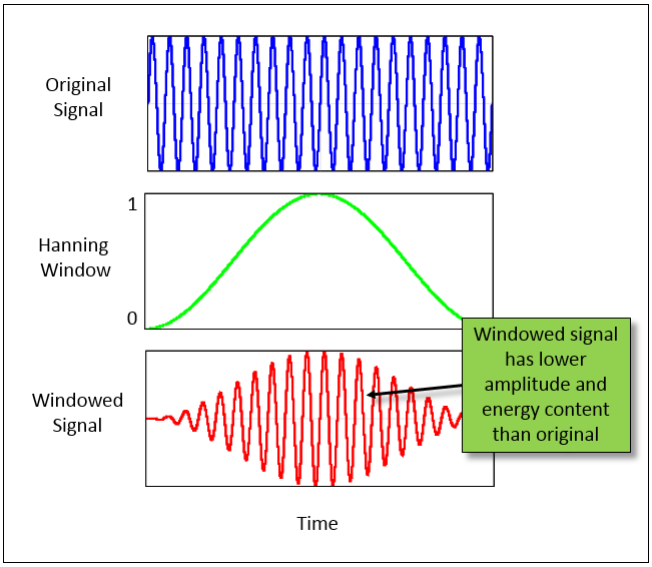


Figure 1.6: Windowed signal shape [4]

1.3.1.3 Fast Fourier Transform and power spectrum

Denoting by N the size of the windowed signal frames, we apply the FFT on each frame to get the spectral representation of each frame indexed by i. Then we calculate the power spectrum(periodogram) of each frame using the following formula:

$$P_i(f) = \frac{|FFT(x_i)|^2}{N}$$

where x_i is the ith frame of signal x, f the frequency and N the size of each frame.

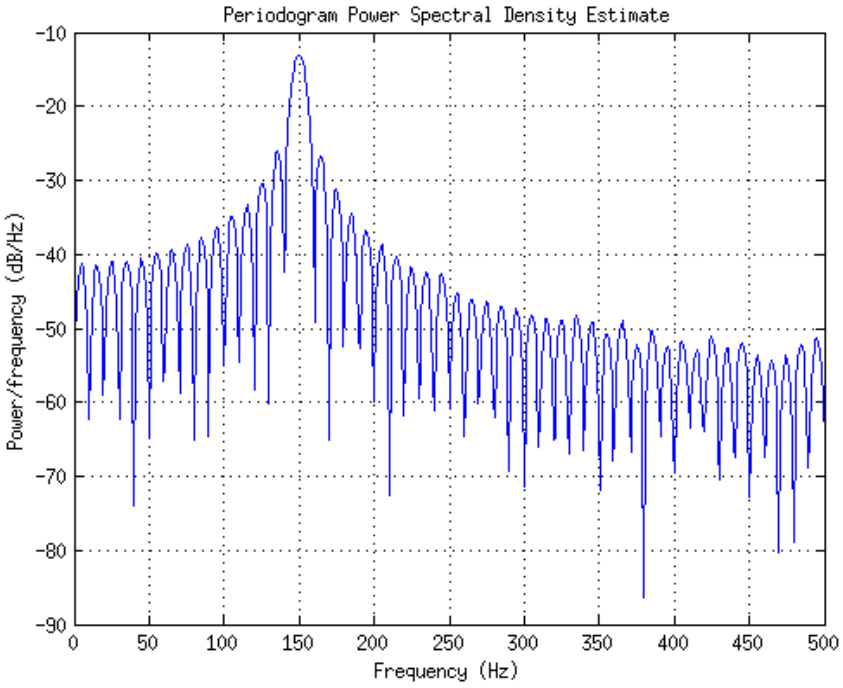


Figure 1.7: Power spectral density (Periodogram)[5]

1.3.1.4 Mel FilterBank

The following step is to apply a mel filter bank to the PSDs of signal frames, typically 40 filters on a Mel-scale.

Each filter in the filter bank is triangular having a response of 1 at the center frequency and decreasing linearly towards 0 until it reaches the center frequencies of the two adjacent filters where the response is 0, as shown in this figure.

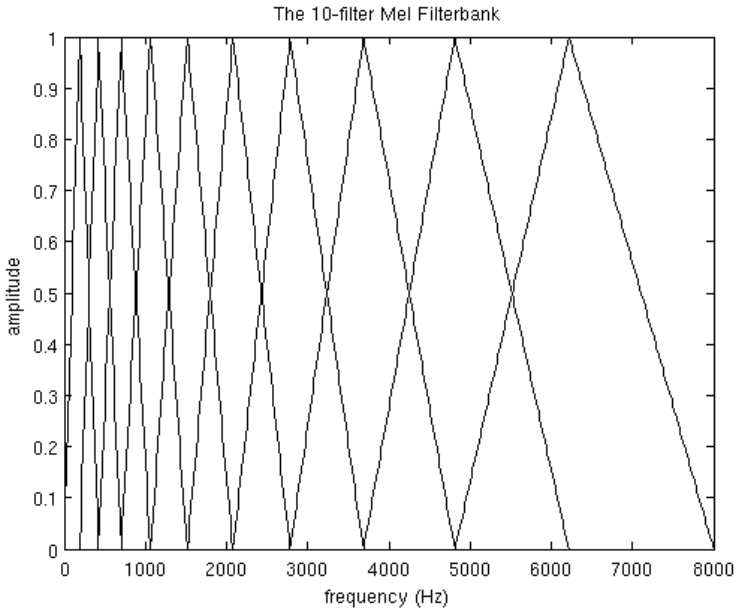


Figure 1.8: With 10 triangular filters [6]

Then the PSD output of each filter is summed to obtain one PSD value per mel sub-band, denoted by P_{sd}, sb = 1..r, where r is the number of mel filters.

1.3.1.5 Mel Coefficients

In order to go back to the 'time' domain, we apply an inverse FFT to the logarithm of the PSD : the MFCC.

Typically, for Audio Classification(AC), we keep the first 2-13 cepstral coefficients and we discard the rest. The reasons for eliminating the higher coefficients is that because they represent fast changes in the filter bank energies and these changes actually degrade the AC performance.

An illustration of MFCCs is given by Fig. 1.9, obtained with "piano" record, with duration 3.5 seconds.

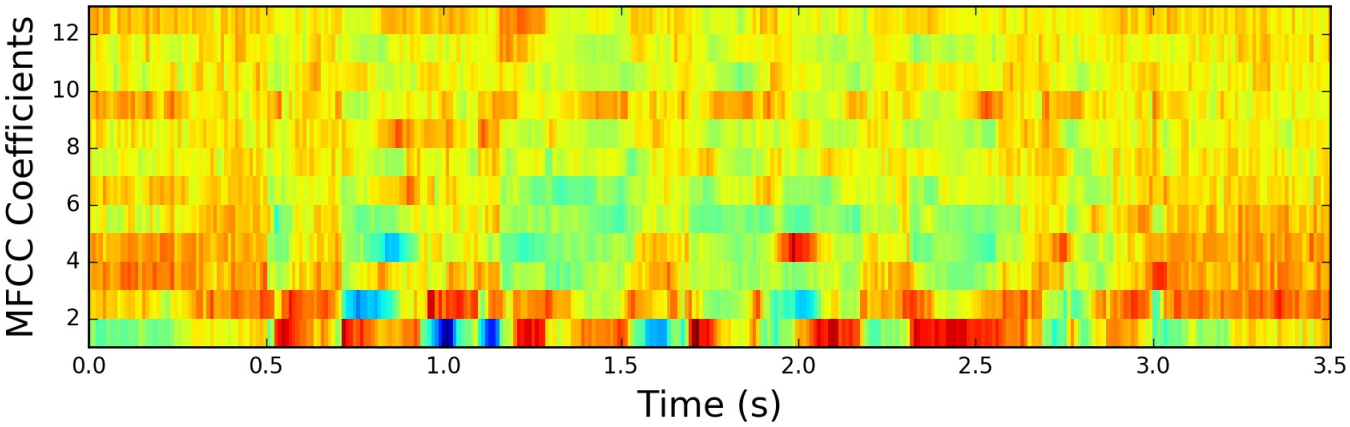


Figure 1.9: Visualisation of MFCC coefficients of a piano recording[7]

1.3.2 MFCC implementation in Python

MFCC features are implemented widely in Python due to libraries like Python_Speech_Feature and librosa. The function below is provided by Python-Speech-Features package, it automatically computes MFCC features from the audio signal:

python_speech_features.base.mfcc(*signal*, *samplerate=16000*, *winlen=0.025*, *winstep=0.01*, *numcep=13*, *nfilt=26*, *nfft=512*, *ceplifter=22*, *winfunc= function(lambda)*

The parameters we have:

- *signal* - is the input audio signal
- *samplerate* - defines how many samples in audio signal over time
- *winlen* - is the size of the analysis window, by default it equals to 0.025s
- *winstep* - define the hopsize between consecutive windows, default is 0.01s
- *numcep* - is the number of cepstral coefficients to return, default is 13
- *nfilt* - the number of filters of the mel filterbank.
- *nfft* - refers to the size of FFT performed on signal frames,default is 512
- *winfunc* - determines the type of window function applyed on each frame

1.4 Libraries

Python is one of many languages to use in audio classification. In matter of fact, it has some advantages when it become a choice to work with. First of all it has a Huge, fast developing community providing plethora of powerful libraries. Secondly, its free open source unlike some other libraries that are quite expensive like MATLAB. Moreover, classification tasks are often requires deep learning and Python has great toolkit for it. Following the top python libraries used in audio classification.

1.4.1 Librosa

Librosa is a known python library for music and audio signal analysis. It provides the necessary modules to create MIR systems. Librosa offers a set of advanced audio input/output functionality, audio processing blocks. Besides it implements a large collection of spectral, temporal audio descriptors.

Figure 1.10: Librosa logo[8]

1.4.2 PyAudioAnalysis

PyAudioAnalysis is an open-source Python library licensed under the Apache license. It offers a wide range of audio analysis tasks such as temporal and spectral feature extraction, audio signals classification, audio events detection by eliminating silence periods from long recordings.

Figure 1.11: PyAudioAnalysis logo[9]

1.4.3 Essentia

Essentia is a open-source C++ library wrapped in python packages for audio analysis tasks and audio-based music information retrieval. Essentia can be considered as a collection of algorithms grouped in a library. As example of algorithms we find time domain descriptors, frequency domain descriptors, statistic descriptors of data , Standard signal processing blocks and O/I functions.

Figure 1.12: Essentia logo[10]

1.5 Audio classification applications

Imagine a machine that can identify a human's feelings, or perform a specific task and all of that through speech. Well that's not impossible with speech analysis and speech emotion recognition applications.

A plethora of different applications have popped up during these years, we mention real time applications such as acoustic monitoring, used to surveil patients and aging people

in hospitals. Moreover, audio event detecting that helps to detect gun shut or car crush sound to ameliorate the emergency service response.

Furthermore, music recognition applications like spotify and shazam that have shown a big success due to the big audience of users. Taking shazam as an example, it helps the user to identity songs through a piece of recorded music.

1.6 Conclusion

In order to achieve improvements in results shown after applying Deep Learning models over real word data, we must do cleaning data technique and that what is we called data preprocessing. After performing this step, obviously, its time to extract the relevant MFCC features to feed them into a Deep Learning algorithm.

In the upcoming chapter we will dig more about one of Deep Learning's architecture.

Chapter 2

Convolutional Neural Networks

2.1 Introduction

Deep Learning is a sub-field of Machine Learning. It consist on how could machine be able to understand and perform specific tasks through a learning process.

More specifically, Deep Learning is based on Artificial Neural Network (ANN) algorithm, which is likewise inspired by how human brain works.

This chapter presents two sections, the first will present the ANN architecture, and the second will focus on a specific type of ANN which is Convolutional Neural Network (CNN).

2.2 Artificial Neural Network

2.2.1 Brain Neural Network

The human brain is complex and considered as a powerful computational machine. The inner structure of it, is a network of cells, called Neurons. The basic unit is interconnected with a set of other neurons by Synapses. Each neuron passes a signal to another neuron through an interface called Axon terminal. That signal will be added to all of the other inputs of that neuron. If the sum of signals exceed a given threshold then it will cause the target neuron to fire an action signal forward.

2.2.2 Artificial Neural network

Motivated by how the human brain works, the ANNs has been invented. We can notice through the figure 2.1 the analogy between a biological neuron and an artificial neuron.

Furthermore the architecture of the ANN is modeled as shown by the figure 2.2, the first layer is the input layer containing the input values (information) of the input sample. The second layer is composed of one or multiple layers which are interconnected with the first layer. Every connection between two neurons is weighted by a weight or bias, each value of the neurons is computed through an activation function, we will enumerate and explain them in further section. Last but not least is the output layer, where the Neural Networks' model output value(s) according to the sample given as input.

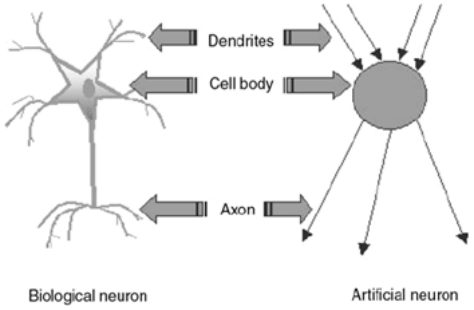


Figure 2.1: Analogy between the Biological Neuron and the Artificial Neuron [12]

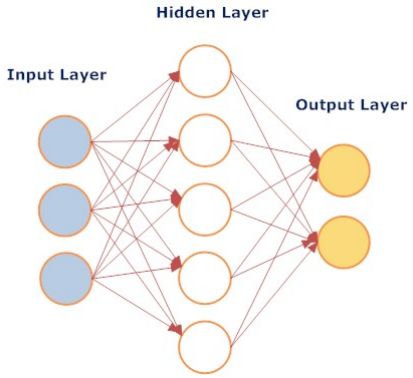


Figure 2.2: Neural Networks layers [13]

After gaining a global view on the ANN architecture, the question that still on, is how does it ANN learn through the input examples.

2.2.3 Gradient descent

Before going directly into how the Neural Networks learns, we are going to introduce the gradient descent because it is the backbone of the learning process.

The role of the gradient descent is to minimize a given objective function $J(\theta)$ given as parameters a vector $\theta \in \mathbb{R}^d$.

In fact, the gradient descent role is to adjust the parameters of the cost function (the weights and the biases) in order to achieve the (local) minimum. Furthermore, in order to control the "speed" of the gradient descent we choose a value, defined as the learning rate. But we have to pick a reasonable value of the constant so we ensure that the model is learning effectively. The figure 2.3 shows, as an example, the effect of the value of the learning rate on a Neural Networks model.

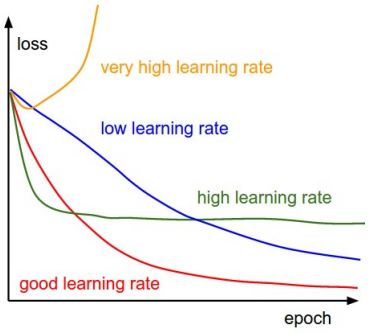


Figure 2.3: Visualisation of different learning rate constant effect on the model [14]

In order to visualize and explain the gradient descent process in a simple way we are going to see an example in the case of the cost function C has only two parameters v1,v2) $\mathbb{R}^2$. As a start, like in practice, we assign random values for v1 and v2 then, through the gradient descent, the model adjust the values of v1 and v2 to minimize C. The figure 2.4 illustrates the situation so clearly where the ball is located at (v1,v2,C(v1,v2)) position and the model eventually adjust the values of v1 and v2 to make the ball roll down to the bottom (the minimum).

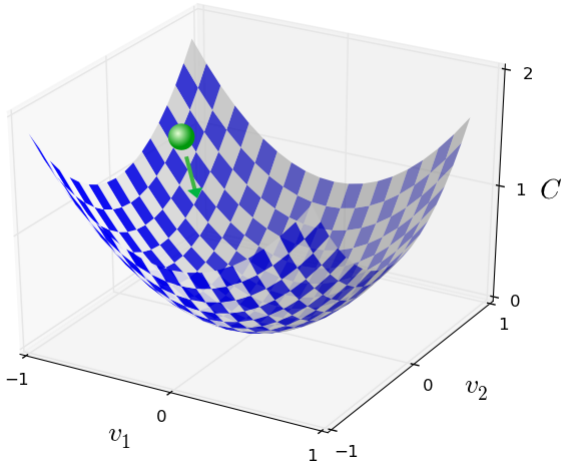


Figure 2.4: Visualisation of the ball rolling down in case of $\mathbb{R}^2$[15]

In order to understand more the gradient descent we will look into the theoretic behind what we just said. The main objective is to minimize our cost function. To do so we need to find a ΔC to ensure that C decreases implies to find the adequate values of $\Delta v1$ and$\Delta v2$. The equation 2.1 resumes what we just said using the gradient:

$$\Delta C \approx \frac{\partial C}{\partial v1}\Delta v1 + \frac{\partial C}{\partial v2}\Delta v2 \tag{2.1}$$

furthermore we denote the gradient vector by ∇C:

$$\nabla C \equiv (\frac{\partial C}{\partial v1}, \frac{\partial C}{\partial v2})^T \tag{2.2}$$

The equation 2.1 becomes:

$$\Delta C \approx \nabla C.\Delta v \tag{2.3}$$

Now we define Δv so that the value ΔC is negative and also depending on the constant η (learning rate) as we see in the equation 2.4:

$$\Delta v = -\eta.\nabla C \tag{2.4}$$

And equation 2.3 becomes:

$$\Delta C \approx -\eta\|\nabla C\|^2 \tag{2.5}$$

As we know the cost function parameters in an ANN are way more than just two parameters. Consequently, in the case of an ordinary ANN the gradient descent vector has d parameters where $d = m > 2$. The equation 2.2 becomes:

$$\nabla C \equiv (\frac{\partial C}{\partial v1}, ..., \frac{\partial C}{\partial vm})^T. \tag{2.6}$$

2.2.4 Back-propagation

Now we understood how the gradient descent works, we will explain in this section how the ANN update the values of weights and biases through the back-propagation in order to reduce the cost function and ameliorate the model performance.

2.2.4.1 Our Neural Networks variables

Before we start to explain how back-propagation works we will define the variables that we will use in this section:
-L: Number of layers in the neural network and $l \in \{1, .., L\}$
-${a_j}^l$: The j^{th} neuron in the l^{th} layer.
-${w_{jk}}^l$: The weight from ${a_k}^{l-1}$ neuron to ${a_j}^l$ neuron.
-${b_j}^l$: The bias connected to the ${a_j}^l$.

We define also the activation function between the layers f knowing that

$${a_j}^l = f(\sum_k {w_{jk}}^l {a_k}^{l-1} + {b_j}^l)$$

In order to simplify this equation we note that ${z_j}^l = \sum_k {w_{jk}}^l {a_k}^{l-1} + {b_j}^l$ with $l \in \{1..L\}$ k is the number of neurons in the $(l-1)^{th}$ layer and j is the neuron connected to in the l^{th} layer. which give us $a^l = f(z^l)$

2.2.4.2 The four fundamental equations of back-propagation

Obviously, the cost function depends on the variation of all the weights and the biases in the Neural Networks. Also, to ensure the good learning process of the model, we need to know the effect of the variation of each variable on the cost function, for this reason the gradient is very important in the ANN. In addition, it is important to compute the values of $\frac{\partial C}{\partial {w_{jk}}^l}$, $\frac{\partial C}{\partial {w_{jk}}^l}$ and $\frac{\partial C}{\partial {z_j}^l}$ to minimize the cost function according to the variation of the Neural Networks variables. For simplification reasons we define the error ${\delta_j}^l = \frac{\partial C}{\partial {z_j}^l}$.

The back-propagation algorithm uses four fundamental equation :

First equation: An equation for the error in the output layer ${\delta_j}^L$

$$\delta^l = \nabla_a C \circ f'(z^L) \tag{2.7}$$

Where $\nabla_a C$ is whose components are the partial derivatives of $\frac{\partial C}{\partial {a_j}^L}$.

To obtain the equation 2.7 we start from ${\delta_j}^L = \frac{\partial C}{\partial {z_j}^L}$ and by applying the chain rule, we will obtain ${\delta_j}^l = \frac{\partial C}{\partial {a_j}^L}\frac{\partial {a_j}^L}{\partial {z_j}^L}$. Knowing that $a^L = f(z^L)$ we get the equation:

$$\delta^L = \frac{\partial C}{\partial {a_j}^L} f'(z^L) \tag{2.8}$$

In this equation 2.8 the first term $\frac{\partial C}{\partial {a_j}^L}$ measures how fast the cost is changing with the variation of ${a_j}^L$ and the second term $f'(z^L)$ measures how fast the output activation function f is changing at z^L.

Equation 2.8 is component wise expression, However, the matrix-based form will be as the equation 2.7.

Second equation: An equation for the error ${\delta_j}^l$ in terms of the error in the next layer ${\delta_j}^{l+1}$

$$\delta^l = (w^{l+1})^T \delta^{l+1} \circ f'(z^l) \tag{2.9}$$

The equation 2.9 show us the dependency of the error of δ^l of the layer l^{th} with δ^{l+1} of the next layer. Before moving forward lets explain how we got this equation.
We start with applying the chain rule

$$\begin{aligned} {\delta_j}^l &= \tfrac{\partial C}{\partial {z_j}^l} \\ &= \textstyle\sum_k \tfrac{\partial C}{\partial {z_k}^{l+1}} \tfrac{{z_k}^{l+1}}{\partial {z_j}^l} \\ &= \textstyle\sum_k \tfrac{{z_k}^{l+1}}{\partial {z_j}^l} {\delta_k}^{l+1} \end{aligned}$$

We note that

$${z_k}^{l+1} = \sum_j {w_{kj}}^{l+1} {a_j}^l + {b_k}^{l+1} = \sum_j {w_{kj}}^{l+1} f({z_j}^l) + {b_k}^{l+1}$$

. Differentiating we obtain: $\frac{\partial {z_k}^{l+1}}{\partial {z_j}^l} = {w_{kj}}^{l+1} f'({z_j}^l)$ For this reason we obtain the equation

$${\delta_j}^{l\text{‘}} = \sum_k {w_{kj}}^{l+1} {\delta_k}^{l+1} f'({z_j}^l)$$

the matrix-based form is the equation 2.9
Third equation: The error in terms of biases:

$$\frac{\partial C}{\partial {b_j}^l} = {\delta_j}^l \tag{2.10}$$

An equation for the rate of change of the cost with respect to any bias in the network. We can easily understand how this equation is deduced using the chain rule.

fourth equation: The error in terms of weights:

$$\frac{\partial C}{\partial {w_{jk}}^l} = {a_k}^{l-1}{\delta_j}^l \tag{2.11}$$

In order to obtain the equation we simply have to use chain rule as usual:

$$\begin{aligned} \frac{\partial C}{\partial {w_{jk}}^l} &= \frac{\partial C}{\partial {z_j}^l}\frac{\partial {z_j}^l}{\partial {w_{jk}}^l} \\ &= {\delta_j}^l {a_k}^{l-1} \end{aligned}$$

2.2.4.3 Training a model using back-propagation

The algorithm of the back-propagation is defined as follow:

1. Input x :set the corresponding activation a^1 of the first layer from the input sample.
2. For each training example apply these steps:
   -feed-forward: Computing the values ${z_x}^l$ and ${a_x}^l$ for $l \in \{1, .., L\}$
   -compute the vector δ^L using equation 2.7
   -Back-propagate the errors δ^l for $l = L-1, L-2, .., 2$ using the equation 2.9
3. Gradient Descent: for each $l = L-1, L-2, .., 2$ update the weights and biases by adding or subtracting the gradient of every variable, through the equations 2.10 and 2.16

We can see clearly why it is called back-propagation. The model is computing the error vector δ^l backward, starting from the output layer.

2.2.5 The activation function

As we saw in the previous section, in feed-forward, we need to compute the value of every neuron through the value of the previous neuron and the weight connecting the two neurons. And to compute these values we need functions known as activation functions or threshold functions or transfer functions. For this reason it is a very important unit in the ANN and to chose one function over an other is a critical choice in the Neural Networks design.

The activation functions are divided into two important categories **non-linear activation functions** and **linear activation functions**.

2.2.5.1 Linear activation function

It is easy to compute this function but we will have two major problems from this function:

1. Not possible to use back-propagation: The derivative of the linear function is the constant a. Consequently the back-propagation algorithm will not know which weight to change in order to minimize the cost function.

2. All the layers collapse: Since it is a linear function the fact to combine many linear function we will obtain a new linear function and transform all the layers into one layer which transform the neural network model into a linear regression model.

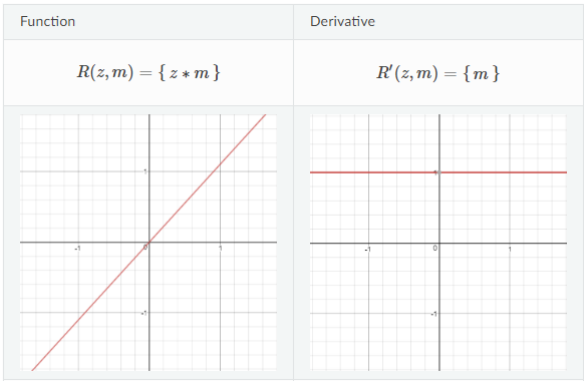


Figure 2.5: Linear function [16]

2.2.5.2 Non-linear activation function

Non-linear activation functions let us avoid the problems of linearity that we mentioned before. These functions allow the model to create complex mappings between the layers inputs and outputs, which are essential for learning and modeling complex and non-linear data.

We note that the non-linear functions are too many nowadays for this reason we will only introduce some of them in this section but you can find others in the appendix.

1. ReLu (Rectified Linear Unit)

   advantages :

   -computationally efficient.

   -avoids the vanishing gradient problem (partially).

   disadvantages :

   -It is only used in within hidden layers of a Neural Networks.

   -The dying ReLu Problem: when x<0 the gradient will be 0 which the weights will not get adjusted during the gradient descent to make the Neural Network learn.

   -The range of ReLu is [0,inf). The activation can blow up the activation.

2. LeakyReLu

   advantages :

   -Prevent dying Relu problem.

   disadvantages :

   -It can not be used for the complex Classification. It lags behind the Sigmoid and Tanh for some of the use cases.

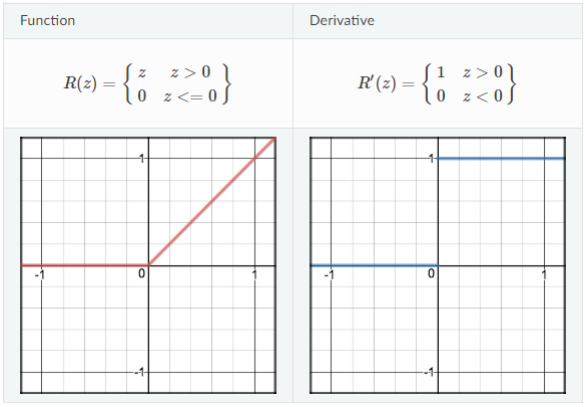


Figure 2.6: Relu function [16]

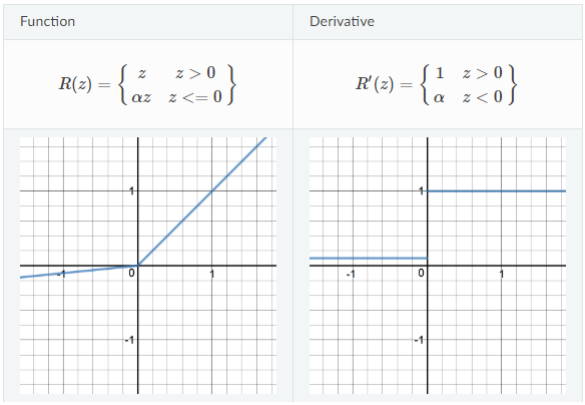


Figure 2.7: LeakyRelu function [16]

3. Softmax

   advantages :

   -Useful as output function when there are multiple classes through computing the probability of every class.

$$\sigma(v_i) = \frac{exp(v_i)}{\sum_k exp(v_k)} \forall i \in \{1, .., k\} \tag{2.12}$$

2.2.6 Cost function

The loss or cost function helps the Neural Network to compute the error of the output in order to make the model learn.

There two main categories for the cost or loss functions, regression costs and classification costs.

2.2.6.1 Regression losses

In regression, the model predict a continuous values. We will define the common loss functions knowing that y_i is the predicted value and $\hat{y}_i$ is the true value.

Mean Square Error/Quadratic Loss/L2 Loss:

$$MSE = \frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n} \tag{2.13}$$

Mean Absolute Error/L1 Loss:

$$MAE = \frac{\sum_{i=1}^{n}|\hat{y}_i - y_i|}{n} \tag{2.14}$$

Mean Bias Error:

$$MBE = \frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)}{n} \tag{2.15}$$

2.2.6.2 Classification losses

In classification models predict, eventually, to which class the sample belongs to, and to do that the model compute the probability of the sample belonging to a certain class.
In the case the prediction is binary:
Binary Logistic loss/ Cross Entropy Loss/Negative Log Like-hood:

$$CrossEntropyLoss = -(\hat{y}_i log(y_i) + (1 - \hat{y}_i) log(1 - y_i)) \tag{2.16}$$

The equation 2.16 can be seen as:

$$CE = \begin{cases} -log(y_i) & = & if \hat{y}_i = 1 \\ -log(1 - y_i) & = & if \hat{y}_i = 0 \end{cases}$$

In the case we have many classes in the model the loss function is called:

Categorical Cross Entropy loss:

$$CE = -(\sum_{i=1}^{k} \hat{y}_i log(y_i)) \tag{2.17}$$

In this case, y_1, . . .,y_k are the probabilities of the k classes (using Softmax activation function), and if the r^{th} class is the ground-truth class, then the loss function for a single instance is defined as follows:

$$CE = -log(y_r)$$

2.2.7 Optimizer

As we saw in the previous section the Deep Neural Networks learn through the gradient descent, and in this section we will introduce different optimizers which optimize the gradient descent to make the Neural Networks learn better.

2.2.7.1 Gradient Descent variants:

The common gradient descent used in Deep Neural Networks are:

Batch gradient descent:

Computes the gradient of the cost function to all the weights and biases of the Neural Networks for the entire training set. This method makes the learning so slow.

Stochastic gradient descent:

Stochastic Gradient Descent (SGD), performs an update on the parameters for each training example. This method is better than the previous one but there is redundancy in the learning process, it performs frequent updates with a high variance that cause the cost function to fluctuate as shown in the figure 14

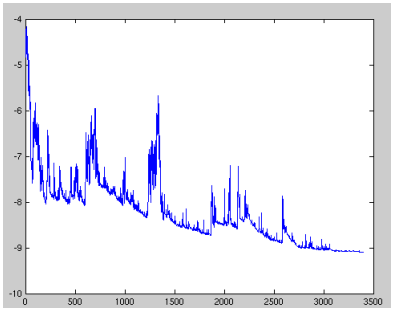


Figure 2.8: Fluctuations of the cost function [19]

Mini Batch gradient descent:

Performs an update for every mini-batch of n samples from the training set. With this method we reduce the high variance of parameters updates comparing to the SGD.

2.2.7.2 Gradient Descent optimization algorithms

In this section we will mention some optimizing algorithms that help to improve the Gradient Descent process to converge faster to the cost function (local) minimum.

Momentum and Nesterov Accelerated Gradient:

The momentum and especially the Nesterov Accelerated Gradient helps the gradient descent converge faster to the (local) minimum of the cost function. In the appendix .2 you can find the figures and the more details about that.

AdaGrad:

Adaptive Gradient Algorithm (Adagrad) adapts the learning rate at each iteration, performing increasingly smaller updates by accumulating all past squared gradients. But the disadvantage of this optimizer is that it stops the learning too early

AdaDelta:

Adadelta is an extension of Adagrad, seeking to reduce its aggressive and monotonically decreasing learning rate.

RMSprop:

Root Mean Square Propagation (RMSprop) uses a moving average of the squared gradients. It modulates the learning rate of each weight based on the magnitudes of its gradients, but not in a monotonical way.

Adam:

Adaptive Moment estimation (Adam) uses a moving learning rate based on squared gradients plus a momentum term (which can be seen as a combination of RMSProp and Momentum)

2.2.7.3 Which optimizer to use ?

To summarize, RMSprop is an extension of Adagrad, the same of for Adadelta. Adam, finally, adds bias-correction and momentum to RMSprop. So far RMSprop, Adadelta and Adam are very similar circumstances but in some way Adam slightly outperform all of them. the figure 2.10 illustrates it in the example of training a model with the popular MNIST database.

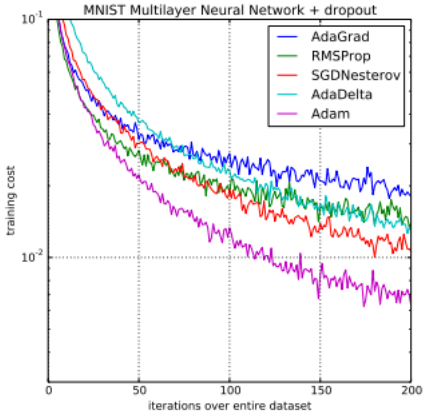


Figure 2.9: Comparison of Adam to Other Optimization Algorithms [21]

2.3 Convolutional Neural Networks

Convolutional Neural Network (CNN) architecture is the key to the tremendous success of Deep Learning in machine vision, because not only it reduces the number of parameters but also makes the Neural Networks learn faster.

Following, a deeper look on the characteristics of CNN architecture.

2.3.1 General Architecture of CNN

Unlike Fully Connected Neural Networks, the layers in CNN has three dimensions: width, height and depth (number of channels). Furthermore, the neurons in one layer do not connect to all the neurons in the next layer but only to a partial region of it. Lastly, the final output will be reduced to a single vector of probability scores, organized along the depth dimension.

As shown in the figure 2.10, a detailed architecture of CNN. It consists of taking an input image as a three dimensions array matrix of (pixels) then they are fed to convolution, pooling and fully connected layers, then we finish by applying Softmax function to classify that image.

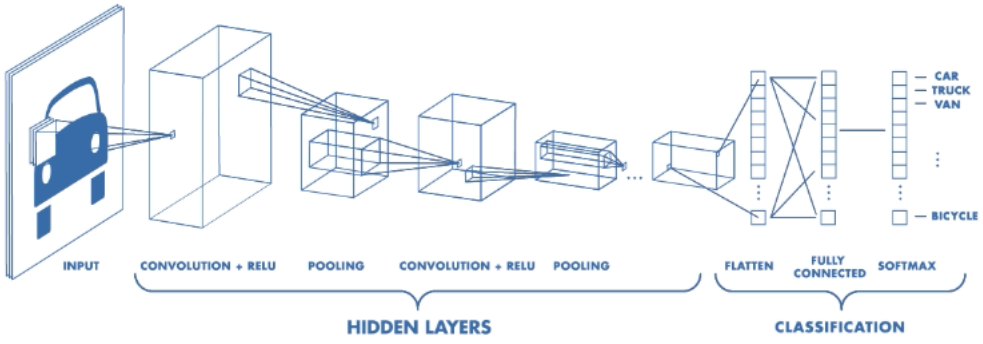


Figure 2.10: CNN architecture example [22]

2.3.2 Convolutional layers

A convolutional layer maps an input volume to an output volume through a set of learnable filters, which make up the parameters of the layer. Every filter must have the same depth as the input volume and the height and the width are small. During the feed-forward, we convolve each filter across the width and the height of the input volume and compute element-wise dot products between the entries of the filter and the input at any position.

The figure 2.11 shows an example of a computed output.

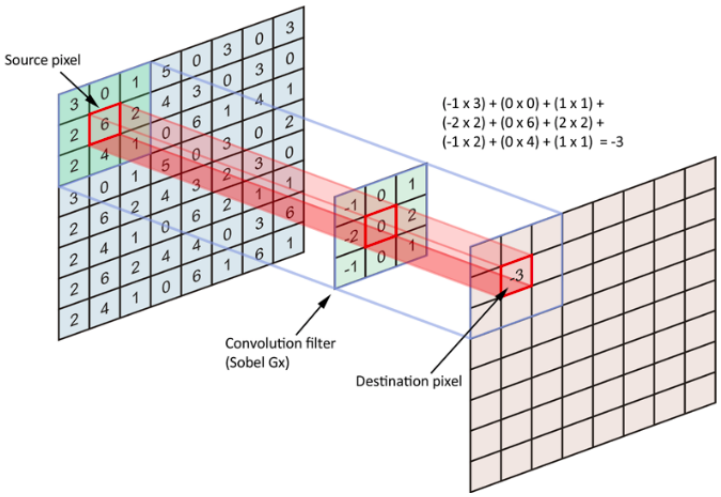


Figure 2.11: Computing one convolution [23]

As we slide the filter over the input volume we will produce a **2-dimensional** activation map that gives the responses of that filter at every spatial position 2.12. Each convolutional layer will have such a set of filters, and each of them will produce a separate 2-dimensional activation map. Then we stack these activation maps along the depth-dimension to produce the output volume.

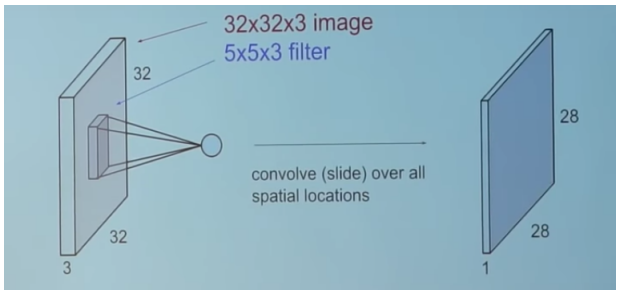


Figure 2.12: Convolving an input image with a filter [24]

2.3.2.1 Stride

The stride defines the number of pixels by which we move the filter when "sliding" it along the input volume. In the figure 2.13, a filter sized 3X3 is shifting by one pixel at a time over an input volume sized 7X7.

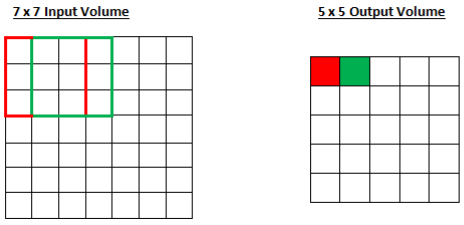


Figure 2.13: One step stride [25]

2.3.2.2 Padding

Let's take the example of a 4X4X3 input volume convolved with a 2X2X3 filter, through applying the equation 2.18, we get a 3X3X1 output volume. Obviously the spatial dimension of the output will decrease as were applying constantly Convolutional layers. We will arrive at point that the filter won't fit the input volume. So as a solution to control the spatial dimension of the output, we pad the picture around the border with zeros as shown in the figure 2.14.

$$O = \frac{I - K - 2P}{S} + 1 \begin{cases} O : output \\ I : input \\ K : filter \\ P : padding \\ S : stride \end{cases} \tag{2.18}$$

2.3.3 Pooling layer

Pooling layer consist of down-sampling an input. In other words, its functionality is to reduce the dimension of the input representation, without losing any important information. This is done in part to minimize the computational cost in the Neural Network, as well reducing the number of parameters.

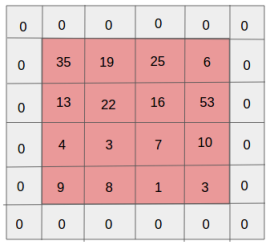


Figure 2.14: One border padding [26]

There are several function options to implement Pooling among which Max Pooling, the most common approach used, Average Pooling and Sum Pooling.

As an example for max pooling, lets take a max filter sized 2X2 and apply it over a matrix sized 4X4 that represent our input. The max filter matrix will move with a stride equally to the length of max filter, accordingly there will no overlap regions. And eventually, it outputs a matrix containing the max value of every sub-region as shown in the figure 2.15.

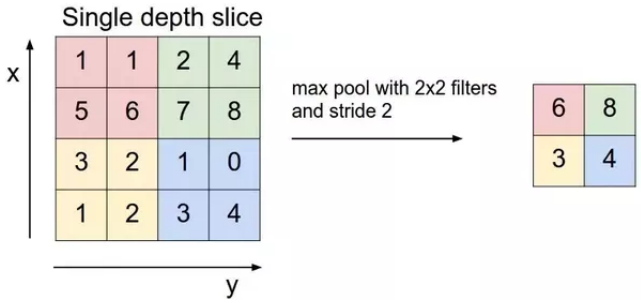


Figure 2.15: Max Pooling example [27]

2.3.4 Dropout Layer

The idea behind Dropout technique is to drop out units and weights associated to these units in deep learning model, so that it makes sure that the model is not getting too fitted to the training data and thus helps to avoid over-fitting problem.

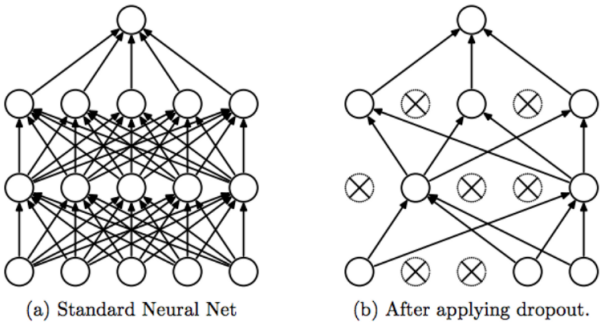


Figure 2.16: Drop-out example [28]

Speaking up of over-fitting, it is a problem in Deep Learning where the model get used to specific samples. As a result it will perform badly and will output inaccurate predictions that sometimes does not make any sense.

The figure 2.17 shows how the fact that training a model too much on a training set leads to over-fitting.

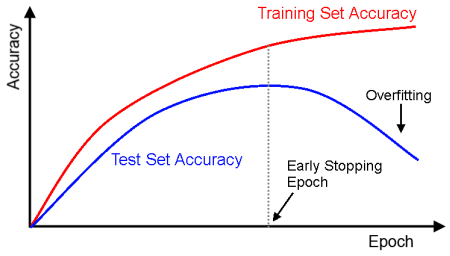


Figure 2.17: Over-fitting effect on a model [29]

2.3.5 Fully connected layer

Fully connected layer as its name implies has full connection to the output layer. After detecting the high level features, now it is time to prepare for classification. For that purpose, we use fully connected layer that takes as an input a one-dimensional vector (the three dimensional layer flattened) and outputs a one-dimensional vector with length equal to N. Here N represents the number of classes that the model will predict from.

2.4 Libraries

Deep learning frameworks are libraries that allows building, training and testing deep learning models more easily and quickly. Following is a list of top popular deep learning frameworks:

2.4.1 TensorFlow

TensorFlow is without doubt the most popular open source software library developed by Google Brain Team for high performance numerical computation using data flow graphs. its well known for its flexible architecture that allows deploying computation across a variety of platforms like CPUs or GPUs in a desktop, server, or mobile device without rewriting code. For that reason, recognizable brands like uber, Airbnb, dropbox and others have all decided to leverage this framework for their own services. Currently its best supported language is Python but there are also experimental interfaces available in C++, Java and GO. For visualizing results, TensorFlow offers TensorBoard that lets developers monitor model training process via various visualizations and its crucial part of its suite. Another crucial part is TensorFlow Serving, which allows developers to easily serve their models at scale in a production environment and includes distributed training.

2.4.2 Keras

Keras is a python-based deep learning library supported by TensorFlow, CNTK, Theano. It works a bit different than the other deep learning frameworks. Actually, keras does not handle the low-level computation like TensorFlow and for that it uses another tool named Backend. So Keras works as high-level API, developed with the purpose of enabling fast experimentation and allows rapid prototyping of neural network models. Besides, its capable of supporting both Convolutional Neural Networks and Recurrent Neural Networks. Its the right framework to begin with in creating models because Keras is easy to learn and use and allows fast deployment.

2.5 Conclusion

As we have seen in this chapter, due to the powerful and efficient computation in CNN we can conclude why is it the go-to model in Computer Vision problems. the CNN becomes more and more the go-to model in Computer Vision problems and that's due to its powerful and efficient computation.

As for the first chapter, we presented the importance of audio data preprocessing and the steps to calculate MFCC features. And the second chapter was about the theory behind ANN and CNN. Now we will cross over the steps of our implementation.

Chapter 3

Implementation

In this chapter we will talk about the project in details mentioning the main steps we made.

3.1 Presentation of the project

Audio classification is a fundamental problem in the field of signal processing with Artificial Intelligence. The task is essentially to extract features from audio (Mel-Frequency Cepstral Coefficient), and then predict, with a model, which class the audio belongs to. In order to classify sounds we chose Convolutional Neural Networks because the model will classify the audio according to the spectrogram generated as an image.

3.2 Data set

The data set is the most important element in DL. As a beginning, we start to exploit a data set called Environmental Sound Classification 50 [36] from Kaggle, containing 50 different sound classes with 40 sample each. This data set is characterized by a balanced number of samples as well as four seconds length in each audio file. Therefore, the main problem in this dataset is that the number of samples of each class is too small for a process of building and training a model.

For this reason we switched to a larger data set which is also from Kaggle website called FSDKaggle2018 [35]. This data set contains 41 different classes: 17 classes of different instruments with 4 521 training audio samples, as well as 24 different environmental sounds classes such as cough, gunshot with 4 952 training audio sample. Each audio file has 44100Hz as a sampling rate, moreover it has different lengths, the longest one has 30 seconds length while the shortest is 0.3 seconds. As well, we should mentioning the fact that the number of samples is different per class.

3.3 Data preprocessing

As we mentioned previously, FSDKaggle2018 data set is very large, but the problem is while working with real data we will always face problems of form uniformity. In our case, each audio file has a different length and this is a problem.

Since we are working with CNN, the input shape has to be the same for all the audio samples. For this reason, we have to find a solution regarding that in order to ensure that we can feed properly the MFCC features extracted from all the audio files into the CNN model.

In the beginning we started to visualize the difference in audio files lengths per class. Figure 3.9 and 3.2 show the obvious unbalance of the audio sounds lengths with the box plot for the instruments and the environmental sounds. We note also that the length's mean for the instruments is 5.9 seconds and for the environmental sounds is 7.59 seconds. These information were not enough to let us decide what length to pick for the audio sounds.

Therefore, we removed all the silence parts from the audio files and check the new length of the files. As a result, We found out that some of the audio files are not loud enough, and that may lead the machine to confuse between two different sounds from two different classes. So to avoid that we decided to remove them from the training set.

After applying these changes on each audio file, the length's mean of the audio files from each category changed: for the instruments it became almost 4 seconds and for the environmental sounds it became 3.7 seconds. So from this it came the idea of choosing 4 seconds length from all the audio files. Audio files longer than 4 seconds will be clipped so that only the first 4 seconds will be used, while for the audio files that are shorter than 4 seconds, will be duplicated till it create a clip 4 seconds in length.

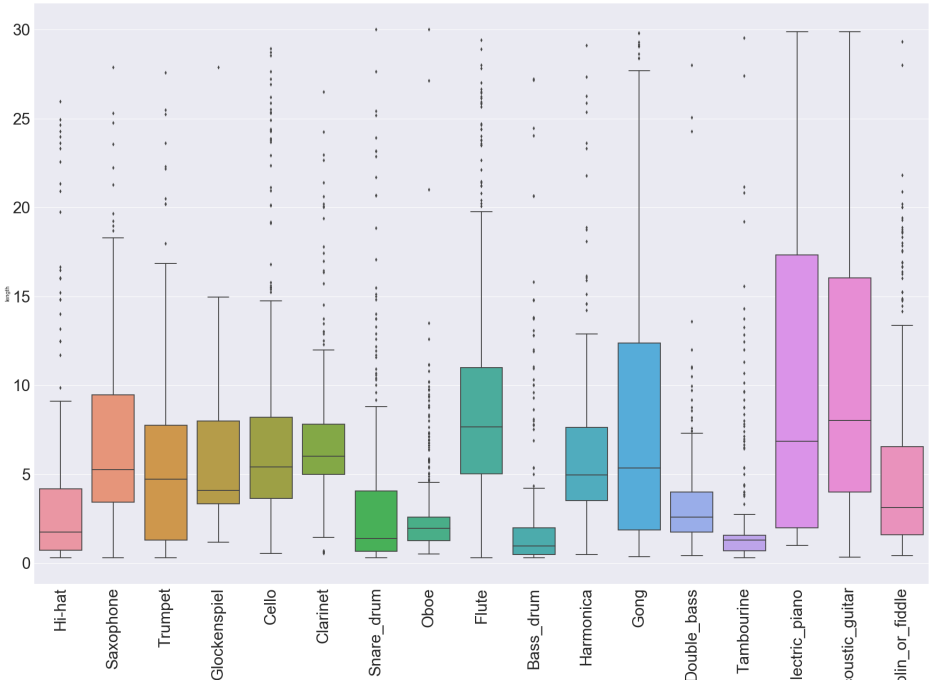


Figure 3.1: visualization of instruments length difference

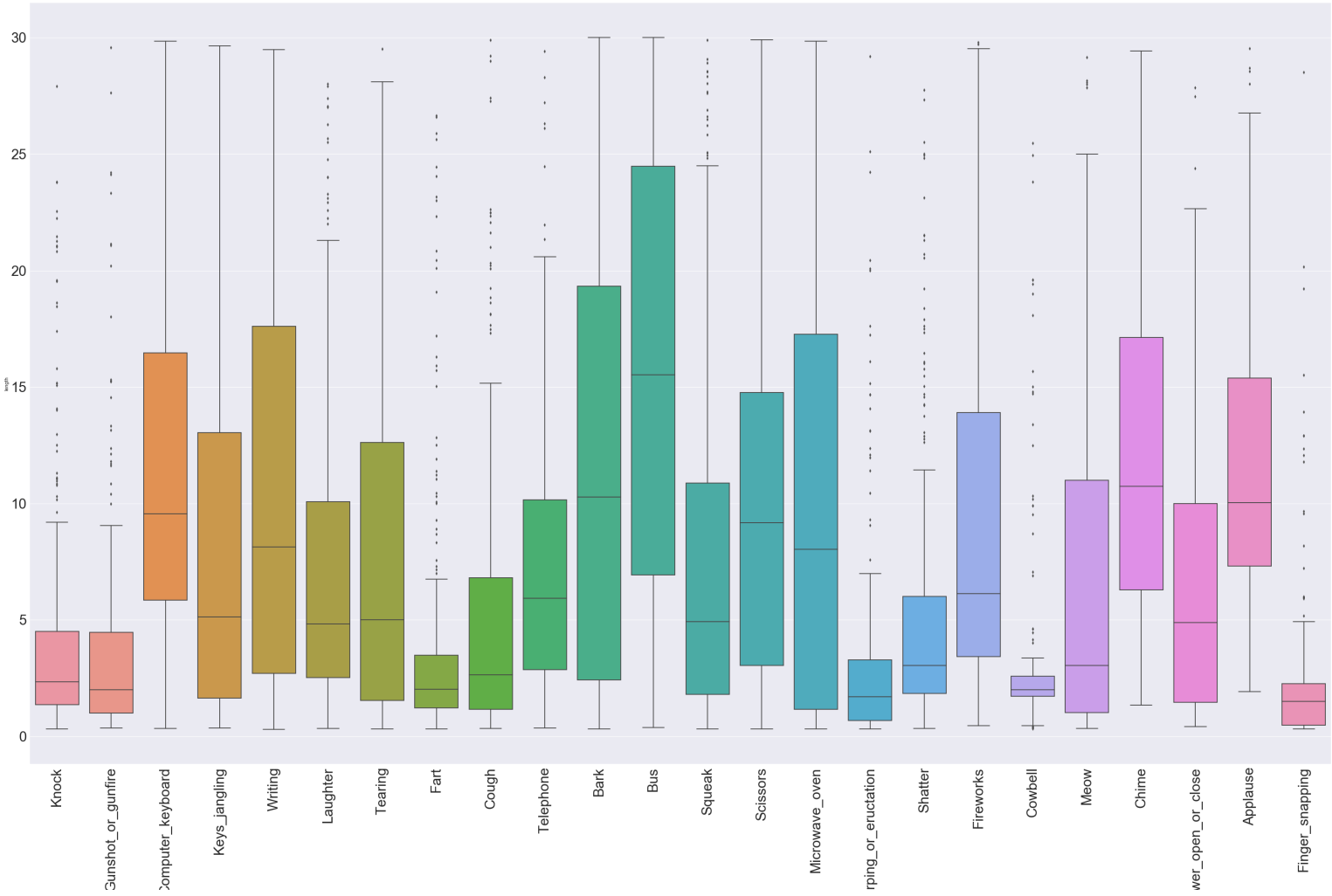


Figure 3.2: visualization of Environmental sounds length difference

3.4 Features extraction

After preparing the audio files with the same length, we proceeded with extracting the MFCC features from every single audio. With extraction of MFCC features was done through Python_Speech_Features library using base.mfcc methode with the following parameters:

The window length is : 0.023 seconds.

The window step is : 0.01 seconds

The number of cepstral coefficients is : 13

The number of mel filters is : 26

The FFT size is : 1024

In the following figures we can see wave plots and MFCC features of both acoustic guitar in figures 3.3 and 3.4 and writing sounds in figures 3.5 and 3.6.

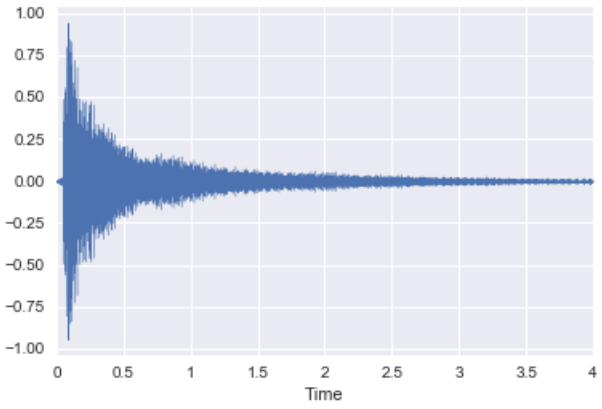


Figure 3.3: acoustic guitar wave form

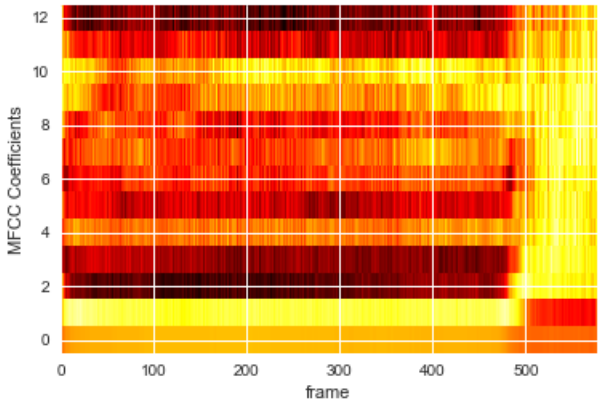


Figure 3.4: acoustic guitar MFCC

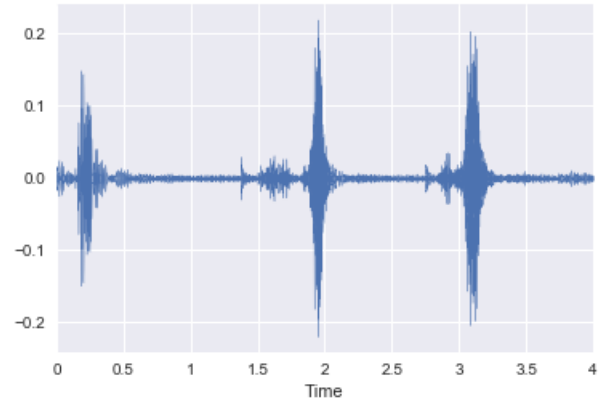


Figure 3.5: "writing" wave form

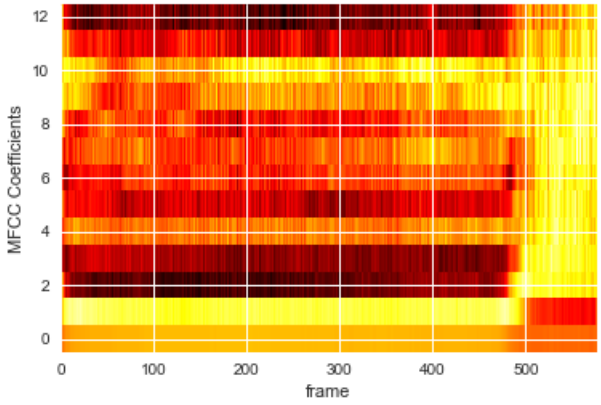


Figure 3.6: "writing" MFCC

3.5 ML or DL

Before going into introducing the CNN-based model we used, we will first explain why we chose Deep Learnig (DL) over Machine Learning (ML).

In both Machine Learning and Deep Learning, we need data to train our model and as for a classification problem they both use an algorithm to train the classifier, but what made the difference is how to process features. In Machine Learning, we need to work on the data engineering part (normalize, orthogonalize, etc) in order to make the classifier better (other than changing the model). However in DL, we can feed the model with row data and it will learn by itself which data features to pic. The figure 3.7 illustrates the difference very clearly.

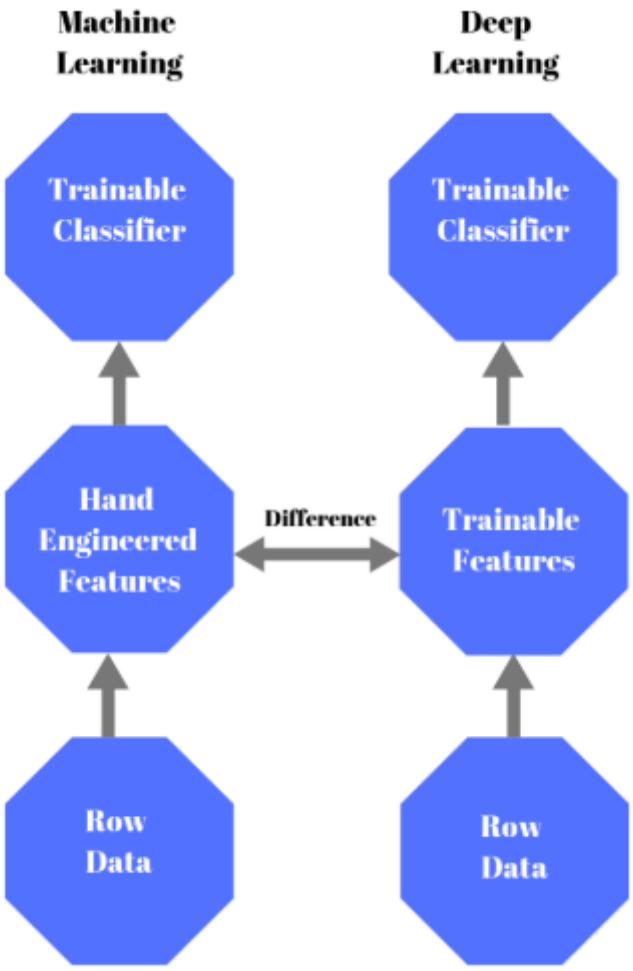


Figure 3.7: Difference between Machine Learning and Deep Learning

As an example we chose the SVM model to show how it is a bad classifier for our case. The figure 3.8 shows the corresponding confusion matrix. (You can compare the ML and DL predictions with the confusion matrix in figure 3.10)

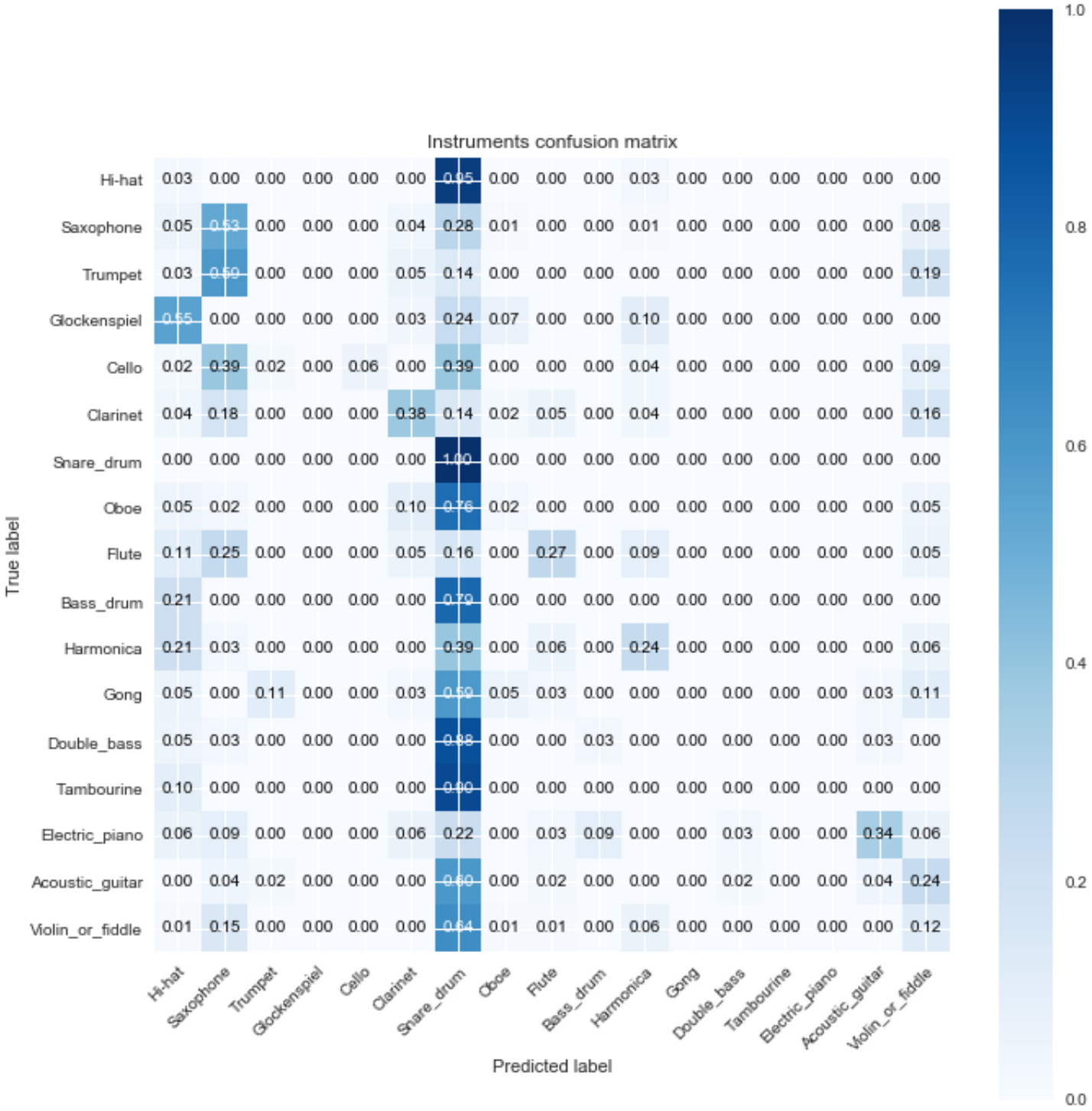


Figure 3.8: SVM confusion matrix for instruments database

From figure 3.8 we can notice that the SVM model is classifying almost all the sounds as snare drums. As a conclusion, the SVM is not a good model to perform instruments classification task regarding the huge amount of features and the big number of classes.

3.6 DL model

The construction of the models was the same for the two categories, since they are similar in audio files lengths.

Each image obtained after extracting the MFCC feature has 13X399 dimension. For this reason the model needs to be as complex as possible to ensure the good accuracy. The model contains seven convolutional layers with LeakyRelu activation function, two maxpooling layers, a dropout layer of 50%, one fully connected layer with Relu activation function and ,eventually, an output layer with Softmax activation function. The figure 3.9 summarizes the model.

```
the input shape is(13, 399, 1)
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 13, 399, 16)       160
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 13, 399, 16)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 13, 200, 32)       4640
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 13, 200, 32)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 13, 100, 64)       18496
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 13, 100, 64)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 13, 50, 128)       73856
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 13, 50, 128)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 13, 25, 256)       295168
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 13, 25, 256)       0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 12, 256)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 4, 5, 512)         1180160
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 4, 5, 512)         0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 4, 5, 1024)        4719616
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 4, 5, 1024)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 2, 2, 1024)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 2, 2, 1024)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                262208
_________________________________________________________________
dense_2 (Dense)              (None, 17)                1105
=================================================================
Total params: 6,555,409
Trainable params: 6,555,409
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.9: DL model architecture

In order to train the models, it was primordial to run it with the Nvidia Graphics Processing Unit (GPU) rather than running it on CPU because of the huge number of parameters existing in both models. This number of parameters needs a performant

computational unit so for this reason we used the GPU in tensorflow by installing **CUDA toolkit**, which provides a development environment for creating high performance GPU-accelerated applications, **CuDNN** which is CUDA Deep Neural Network library and **Create a virtual environment** using Anaconda distribution.

3.7 Prediction results

In this section, we present and discuss the prediction results obtained with the developed DL models.

3.7.1 Data set "instruments"

Figure 3.10 shows the confusion matrix of the instruments' model predictions, where We can clearly notice that almost all the samples are correctly classified.

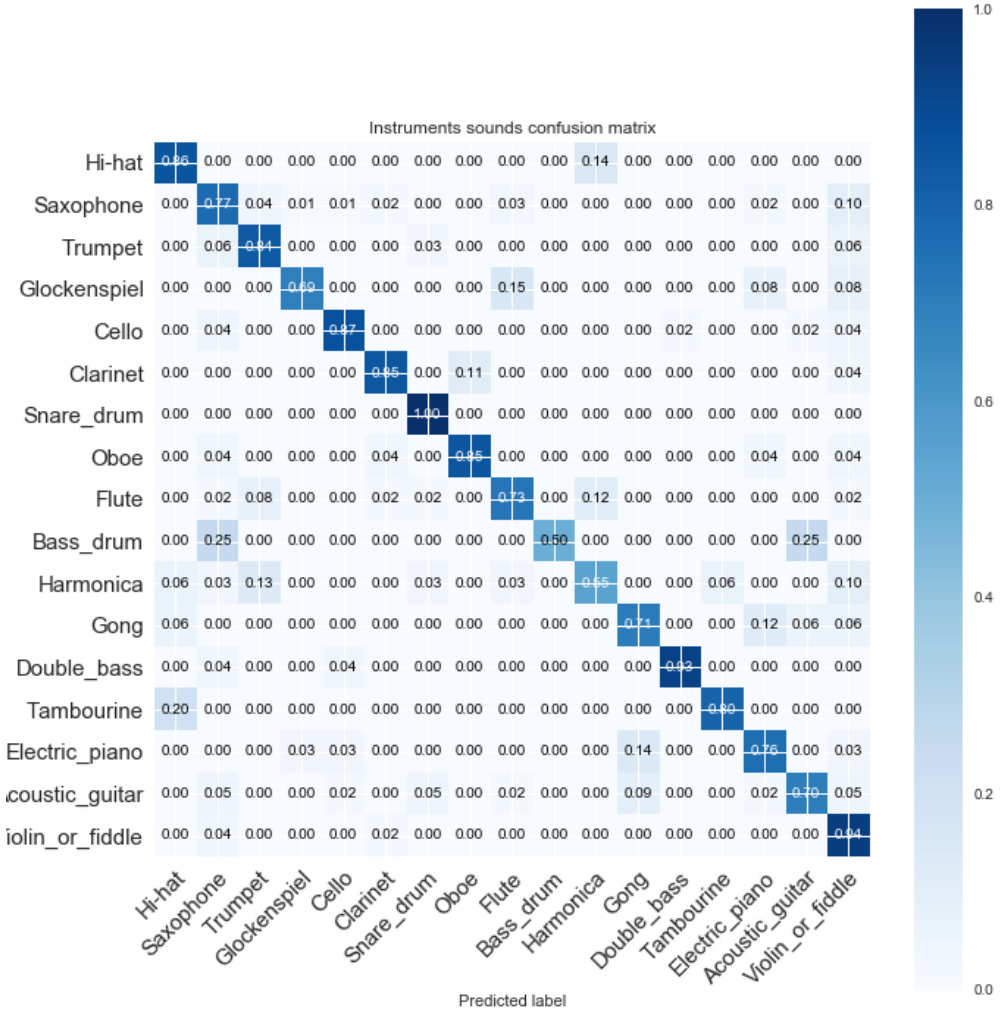


Figure 3.10: Instruments Confusion matrix

3.7.2 Data set "Environmental sounds"

As shown in figure 3.11, the prediction is much better for some classes than others. We conclude that we obtained these kind of results either because of lack of training samples or because of the model's complexity that was not enough to have a better prediction.

Figure 3.11: Environmental sounds confusion matrix

3.8 Conclusion

The results obtained with the CNN-based DL models were much better than those obtained with the SVM model. This explain how much the CNN-based DL model is powerful, as well as MFCC features that deserve the credits for these good results.

Inspite of the unbalanced and the unclean data set and the fact that we trained the model on a normal desktop, we had very good predictions.

Moreover, since the number of training data is so large the model needs more complicated layers and more parameters, we aim to use EC2 instance on Amazon Web Services (AWS) in order to achieve a better classification performance.

Conclusion and perspectives

Since the audio content has increased, it becomes a need to identify sounds. Therefore, systems were popped up to satisfy this need and perform certain tasks.

Although human can perform sound classification, computers still automatically perform poorer than humans. Over years, researchers were trying to improve machine learning to a level that can classify sounds better than humans. Indeed, with Deep Learning algorithms it becomes a reality.

The proposed project consists in analyzing recorded audio signals, removing background noise and discarding the silence from the audio as a preprocessing step. After cleaning the audio data, we need to extract prevalent features, as in our case we chose MFCC features. As a final step, we built a CNN model which we trained on audio features. The learning process made the model able to distinguish between the classes and that is because each class has specific features.

Before starting our project of Environmental sounds Classification, we had to learn and understand the prerequisite of signal processing, such as sampling, filtering and noise discarding, as well as understanding MFCC audio features and how to extract them from audio signals. And lastly and the most important part, we learned about Deep Learning from scratch, since there is no academic courses in our second year. Then we focused on CNN architectures since we are working on images.

The end of year project, gave us the opportunity to more develop our skills whether technical or soft. During the period of our project, we faced some difficulties because, as software engineers, we had no signal processing background. It also helped us to discover and enrich our knowledge in both signal processing and Deep Learning. We also learned how to manage time and control stress throughout the period of the project implementation.

As a perspective, we aim to enrich our knowledge in signal processing to make the model performance higher, without forgetting to enrich our knowledge in the theoretical part of Deep Learning algorithms. We are confident that this project will make us go even further and make us dig more on other problems like speech recognition problems.

Bibliography

[1] https://www.mathworks.com/help/matlab/import_export/read-and-get-information-about-audio-files.html

[2] https://upload.wikimedia.org/wikipedia/commons/a/aa/Mel-Hz_plot.svg

[3] https://sites.google.com/site/stevedtran/course/intro-to-digital-signal-processing/notes2/windowing/type-of-windowing/hann-window

[4] https://community.plm.automation.siemens.com/t5/Testing-Knowledge-Base/Window-Correction-Factors/ta-p/431775

[5] http://www.ece.northwestern.edu/local-apps/matlabhelp/toolbox/signal/spectra8.html

[6] http://www.ece.northwestern.edu/local-apps/matlabhelp/toolbox/signal/spectra8.html

[7] https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html

[8] https://github.com/librosa

[9] https://github.com/tyiannak/pyAudioAnalysis

[10] https://essentia.upf.edu/documentation/essentia_python_tutorial.html

[11] Gurney. An Introduction to Neural Networks. 2014

[12] www.researchgate.net/figure/7947079_Analogy-between-artificial-neuron-and-biologi

[13] http://www.ecommerce-digest.com/neural-networks.html

[14] http://cs231n.github.io/neural-networks-3/

[15] http://neuralnetworksanddeeplearning.com/chap1.html

[16] https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html

[17] https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc

[18] https://gombru.github.io/2018/05/23/cross_entropy_loss/

[19] https://en.wikipedia.org/wiki/Stochastic_gradient_descent

[20] http://ruder.io/optimizing-gradient-descent/

[21] Sebastian Ruder. An overview of gradient descent optimization algorithms. 2016

[22] https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convoluti
html

[23] https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networ

[24] https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neura

[25] https://adeshpande3.github.io/A-Beginner27s-Guide-To-Understanding-Convolutional-

[26] https://blog.xrds.acm.org/2016/06/convolutional-neural-networks-cnns-illustrated-

[27] https://www.quora.com/What-is-max-pooling-in-convolutional-neural-networks

[28] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. 2014

[29] https://deeplearning4j.org/docs/latest/deeplearning4j-nn-early-stopping

[30] https://fr.wikipedia.org/wiki/TensorFlow

[31] https://keras.io/

[32] https://fr.wikipedia.org/wiki/Fichier:Pytorch_logo.png

[33] https://mxnet.apache.org/

[34] Charu C. Aggarwal. Neural Networks and Deep Learning. 2018

[35] https://www.kaggle.com/c/freesound-audio-tagging/data

[36] https://www.kaggle.com/mmoreaux/environmental-sound-classification-50

Appendix

.1 Activation functions

.1.1 Sigmoid or Logistic function

advantages :

-Smooth gradient: prevent the radical shift in the output values

-Output values normalized between 0 and 1.

-Enables clear predictions due to the smoothness of the function.

disadvantages :

-For very high and low value of X the gradient of sigmoid function is zero, causing a vanishing gradient problem. In this case the neural network either is so slow to learn or even stop learning further.

-Its output isnt zero centered. It makes the gradient updates go too far in different directions. $0 <$ output < 1, and it makes optimization harder.

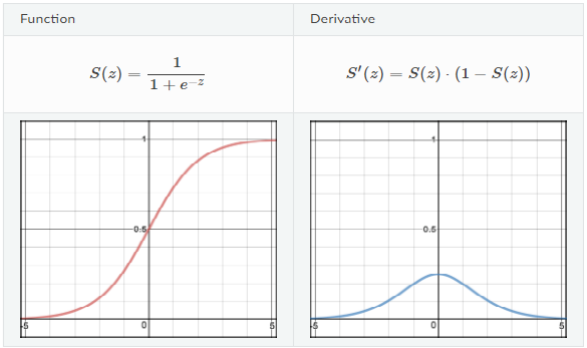


Figure 12: Sigmoid function [16]

.1.2 Tanh

advantages :

-Zero centred function

-The derivative is steeper than the Sigmoid function.

disadvantages :

-The problem of vanishing gradient.

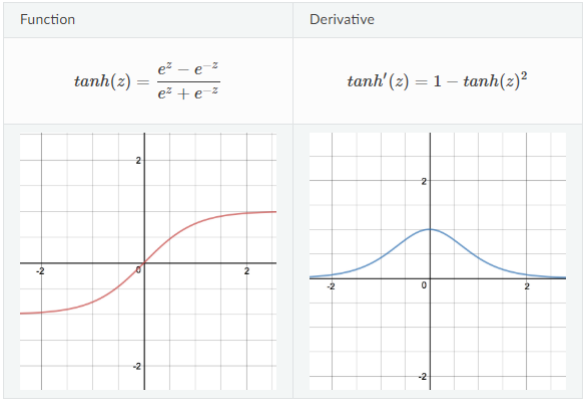


Figure 13: Tanh function [16]

.1.3 ELU(Exponential Linear Unit)

advantages :
-ELU is a strong alternative to ReLu.
-Produce negative output.
disadvantages :
-when x is very low the derivative becomes zero which stop the Network from learning.
-It can blow off when x is very high.

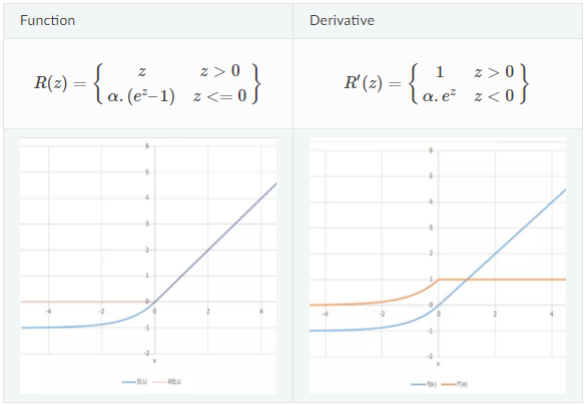
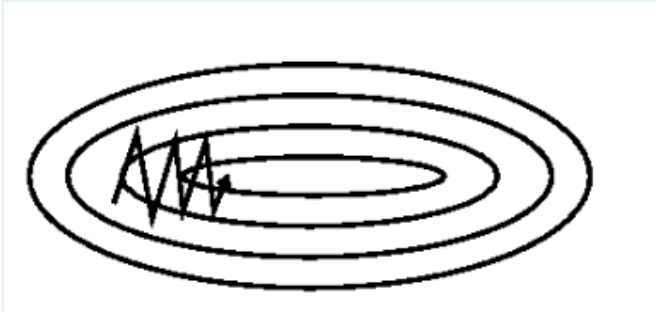


Figure 14: ELU function [16]

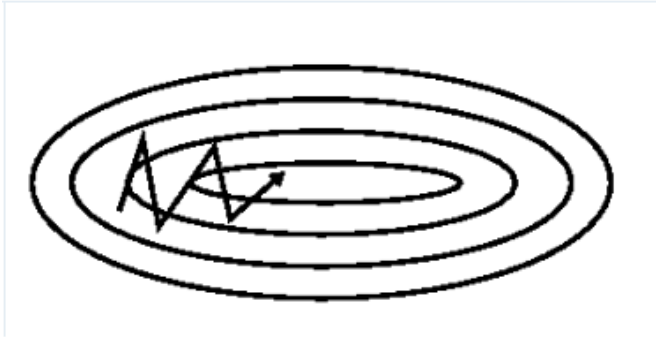
.2 Momentum effect on the gradient descent

.2.1 Momentum

For example SGD has trouble navigating to the local minimum, for this reason the momentum is used, it accelerates the SGD to go to the relevant position (local minimum) and reduces oscillations as shown in figure 15a and 15b . We can see also in figure 16 the effect of the momentum when we are at a certain position of the cost function.



(a) SGD without momentum



(b) SGD with momentum

Figure 15: Momentum effect on SGD
[20]

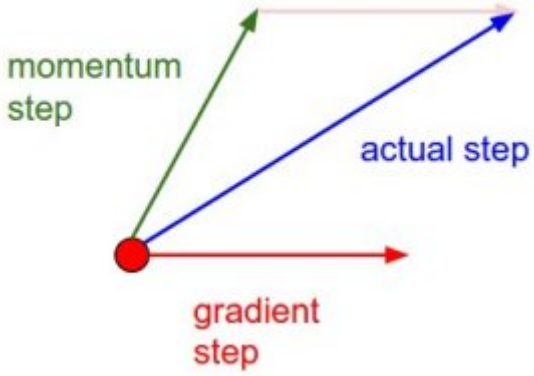


Figure 16: Momentum update [14]

.2.2 Nesterov accelerated gradient:

But it does not stop here. We can make the SGB performance even better through the Nesterov Momentum. The Nesterov Momentum is **slightly** different version of the momentum update that we had just mentioned. It enjoys stronger theoretical converge guarantees for convex functions and in practice it also consistently works slightly better than standard momentum.

Let's take the figure 17 to better understand the difference: In this version, instead of

evaluating the gradient at the current position (red circle), we evaluate the gradient from the tip of the green arrow mentioned in the figure as the ”lookahead” gradient step.

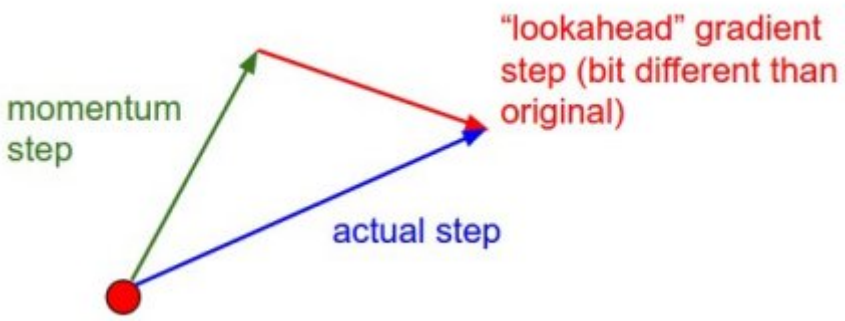


Figure 17: Nesterov Momentum update [14]

.3 Libraries

.3.1 Pytorch

Pytorch is a Python open source native library developed by Facebook to help power its services. Not only used by Facebook but also by several giant industries like Twitter and Salesforce. PyTorch is based on Torch Lua deep learning framework that offers a tensor computation with performant GPU acceleration, as well as dynamic deep learning style way simpler compared to Torch.

.3.2 MXNet

MXNet is a deep learning framework created by Apache and supported by multiple languages like Python, R, C++, and Julia. It has been adopted by AWS, Intel and other remarkable companies. Besides, It offers an API that enable the user to code in huge variety of programming languages. MXNet has a high performance imperative API, its simple like keras and dynamic like PyTorch which makes debugging much easier. Not to mention the fact that It was designed for the purpose of providing high efficiency, productivity, and flexibility.