

University “Politehnica” of Bucharest
Faculty of Electronics, Telecommunications and Information Technology

Musical chord recognition using neural networks

Diploma Thesis

submitted in partial fulfillment of the requirements for the Degree of
Engineer in the domain *Electronics and Telecommunications*, study
program *Technologies and Communication Systems*

Thesis Advisors

As. drd. ing. Victor POPA

Prof. dr. ing. Cristian NEGRESCU

Student

Alexandra-Cristina PLATON

2020

DIPLOMA THESIS
of student **PLATON A. Alexandra-Cristina** , 442G-TST.

1. Thesis title: Musical chord recognition using neural networks

2. The student's original contribution will consist of (not including the documentation part) and design specifications:

A series of algorithms will be developed for identifying chords from musical pieces. Initially, the musical signals will be processed in order to extract the chromagram evolution in time. The chromagram results will be filtered to improve the quality of recognition. Based on the chromagram, a neural network that recognizes chords will be built. The neural network will be trained using sets of chords from databases known in literature. To improve the detection, it will be taken into account the possibility of identifying the beginning of a chord (on-set detection). The testing of the neural network will be performed both with signals from databases and with own recorded signals, evaluating the performance of recognition.

3. Pre-existent materials and resources used for the project's development:

Python, Matlab

4. The project is based on knowledge mainly from the following 3-4 courses:

Signals and Systems, Analysis and Synthesis of Circuits, Audio Engineering, Digital Signal Processin

5. The Intellectual Property upon the project belongs to: U.P.B.

6. Thesis registration date: 2019-12-03 08:59:31

Thesis advisor(s),

As. Drd. Ing. Victor POPA

Prof. dr. ing. Cristian NEGRESCU

Department director,

Conf. dr. ing. Eduard POPOVICI

Student,



Dean,

Prof. dr. ing. Mihnea UDREA

Validation code: **8bf257580f**

Copyright © 2020, *Platon Alexandra-Cristina*

All rights reserved.

The author hereby grants to UPB permission to reproduce and to distribute publicly paper and electronic copies of this thesis document in whole or in part.

Statement of Academic Honesty

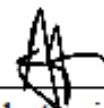
I hereby declare that the thesis “ *Musical chord recognition using neural networks*”, submitted to the Faculty of Electronics, Telecommunications and Information Technology in partial fulfillment of the requirements for the degree of *Engineer* in the the domain *Electronics and Telecommunications*, study program *Technologies and Communication Systems*, is written by myself and was never before submitted to any other faculty or higher learning institution in Romania or any other country.

I declare that all information sources sources I used, including the ones I found on the Internet, are properly cited in the thesis as bibliographical references. Text fragments cited “as is” or translated from other languages are written between quotes and are referenced to the source. Reformulation using different words of a certain text is also properly referenced. I understand plagiarism constitutes an offence punishable by law.

I declare that all the results I present as coming from simulations or measurements I performed, together with the procedures used to obtain them, are real and indeed come from the respective simulations or measurements. I understand that data faking is an offence punishable according to the University regulations.

Bucharest, 24.06.2020

Alexandra-Cristina PLATON



(student's signature)

Contents

List of figures	11
List of tables.....	13
List of abbreviations	15
INTRODUCTION	17
CHAPTER 1 Artificial intelligence	19
1.1 Deep learning	19
1.2 Biological neuron vs. Artificial Neuron	20
1.3 Basic structure of a Neural Network	21
1.1 The Perceptron	22
1.2 Feed-Forward Neural Networks	22
1.3 Recurrent Neural Networks.....	24
1.4 Convolutional Neural Networks	24
CHAPTER 2 Music Perception and Representation	27
2.1 Perception of sound	27
2.2 Sound Level.....	28
2.3 Note and Chord Structure	28
2.4 Types of Instruments	30
CHAPTER 3 Chord recognition system.....	33
3.1 Chromagram Computation:.....	33
3.1.1 Constant Q Transform	33
3.1.2 Implementation of the CQT.....	35
3.1.3 Post-Filtering the Constant-Q Transform	37
3.1.4 Representation of the chromagram	38
3.2 Post-filtering the chromagram	39
3.2.1 Moving-average filter	40
3.2.2 Geometric mean.....	41
3.3 Onset detection	42
CHAPTER 4 Artificial neural network.....	45
4.1 Network structure	45
4.2 Database of chords.....	48
4.2.1 Training dataset.....	48
4.2.2 Test dataset	49

4.3 Neural Network training.....	49
CHAPTER 5 Experimental Results	53
5.1 Experiment 1	54
5.2 Experiment 2	55
5.3 Experiment 3	56
CONCLUSIONS	59
Bibliography.....	61
Annexes	65

List of figures

Figure 1.1 Structure of a neural network [3].....	19
Figure 1.2 Network layers and connections [5]	20
Figure 1.3 Biological Neuron vs. Artificial Neuron [6].....	21
Figure 1.4 Parameters of the neural network [7]	21
Figure 1.5 The Perceptron model [8].....	22
Figure 1.6 Sigmoid function	23
Figure 1.7 Neural network architecture with two inputs	23
Figure 1.8 ReLU activation function [9].....	24
Figure 1.9 Convolutional Neural Network Structure [10]	25
Figure 2.1 Decibels and sound sources [13]	28
Figure 2.2 Piano keyboard: notes and octave representation [15]	29
Figure 2.3 C major and C minor chord [17]	29
Figure 2.4 Main components of an electric guitar [19].....	31
Figure 2.5 Guitar strings and chords [21]	31
Figure 3.1 Representation of Hann and Hamming windows [25]	35
Figure 3.2 Constant-Q Transform.....	36
Figure 3.3 Constant-Q Spectrogram (left) and zoomed plot (right)	37
Figure 3.4 Chromagram of a 12 seconds guitar audio sample.....	39
Figure 3.5 Note C3 played on the piano (left) and on the guitar (right) [29]	39
Figure 3.6 Chromagram before filtering	40
Figure 3.7 Chromagram passed through a moving-average filter	41
Figure 3.8 Chromagram passed through a geometric-mean filter	42
Figure 3.9 Onset, attack, transient and decay in the ideal case [33]	43
Figure 3.10 Onset detection function in Matlab (right)	44
Figure 4.1 Structure of <i>AlexNet Network</i> (modified).....	46
Figure 4.2 Training and validation data distribution.....	47
Figure 4.3 Analysed network structure	48
Figure 4.4 Training results	50
Figure 4.5 Confusion matrix for Test Data	50
Figure 4.6 Confusion matrix for Validation Data	51
Figure 5.1 Flowchart of the test script	54
Figure 5.2 Confusion matrix for piano chords	56
Figure 5.3 Confusion matrix for violin chords	56

List of tables

Table 1.1 Parameter evolution in a CNN [1].....	26
Table 2.1 Sound: physical vs. perception.....	27
Table 2.2 Classification of instruments [18].....	30
Table 3.1 Comparison of variables: Discrete Fourier Transform and CQT [2].....	34
Table 5.1 Experimental results.....	55
Table 5.2 Results of finding the main chord from an audio file.....	55

List of abbreviations

AI = Artificial Intelligence

CNN = Convolutional Neural Network

CQT = Constant Q Transform

CPU = Central Processing Unit

FNN = Feedforward Neural Network

ML = Machine Learning

RNN = Recurrent Neural Network

SPL = Sound Pressure Level / Sound Power Level

STFT = Short – Time Fourier Transform

INTRODUCTION

In present times, many computer-based programs have been developed in order to facilitate human activities. The convenient example to this is the smartphone, that incorporates the face recognition feature for unlocking it or speech recognition commands that speed up the human-computer interaction. The main field that is vital for these types of application is digital signal processing.

Digital processing has been accepted with enthusiasm in the world because it is fast, accessible and of major help to humans. From the debut of the computer until nowadays, digital signal processing is a main component of various day-to-day activities, because it is a main component of the digital products that people use everyday. From the personal computer(PC), laptop or mobile phone to the ultrasound scanner and electrocardiogram machine in medical sphere or the digital camera in photography, each of them is based on signal processing.

A particular domain that is of high interest currently is the audio processing of signals. Modern times demand higher quality of sound, the ability to change fast and numerous derivations from the original sound, achieved through meticulous techniques. Of course, with high demands showed up advanced software programs and more powerful computation machines. But how far can these programs be implemented and how sophisticated can they become?

Artificial Intelligence (AI) is currently a debated subject in terms of morality in the world. However, it has the ability to make our lives effortless. Some examples of its application are self-driving cars, visual recognition, fraud prevention, speech-to-text translation, social media feeds and even humanoid robots. AI is a concept applied to machines that simulate the human intelligence, work and solve problems in the same manner as the human brain does.

Audio processing is a domain where AI can be profoundly applied. The aim of this project is to explore this application on audio signals, specifically on music from instruments. The preference for this subject is due to a personal passion for music, but also because it is a considerable need of complex software technologies nowadays.

Music represents the combination of various sounds generated by musical instruments combined in a pleasing way for the humans to listen to. But when it comes to process sounds, it must be taken into account the way in which humans perceive music. It is of great practice for a user to detect chords from a song and to perform it on its own or even to record a song and to extract the main chords from it. Furthermore, the software used in this project can be adjusted to the needs of a user and adapted accordingly.

This project creates an automatic system based on neural networks using *Matlab* that can detect chords from musical samples, with a restrained application on music played on guitar. The targets of the project are:

- to describe the concept of Artificial Intelligence and the structure of a neural network
- to present the main aspects of audio processing
- to develop a relevant method of feature extraction from a signal
- to develop a method of chord partition of an audio file
- to choose an optimal neural network architecture and to adjust it to the chord recognition system
- to train the neural network with a large database of guitar chords

- to test the software with a script that inputs audio files recorded on a guitar and to test the efficiency of the system

The first chapter of the project is a brief introduction to artificial intelligence, but more on deep learning. A basic structure of an artificial neural network is defined, followed by three most-common network types, with emphasis on the Convolutional Neural Networks.

Chapter 2 presents how is the music perceived, sound characterization and instruments classification with an accent on the electric guitar.

In *Chapter 3*, a system of chord recognition is built, by extracting the chromagram from an audio signal. The Constant-Q Transform analysis is presented that helps in constructing a chromagram. Furthermore, some filtering techniques are described that help in pre-processing the features before passing them as input to an artificial neural network.

Chapter 4 begins with the structure of implemented the neural network, the training and test databases of guitar chords and the process of training, along with some results based on the efficiency of the process.

A program is defined in *Chapter 5* that is able to take as input audio guitar files, compute the chromagram of each chord present in the audio file and recognize it using the pre-trained network from *Chapter 4*.

The project ends with some conclusions on the results and the efficiency of the program, personal contributions and ideas for expansion and improvement of the project.

The references from the chapters represent the documentation and software-related guidance that contributed to the development of the project. They can be found in the *Bibliography*.

Annex 1 incorporates the user-defined functions from the project, along with the code for defining an artificial neural network in *Matlab*.

CHAPTER 1 Artificial intelligence

Machine learning is a field of computer science that is expanding nowadays and it is used in many applications from different domains. It implements algorithms based on pattern recognition that can be trained using large datasets of information. The aim of these algorithms is to create a pattern based on the input data such that they learn progressively about the received information and finally can make predictions and decisions based on it.

Artificial intelligence is the broader concept that embodies machine learning. AI rather refers to the behaviour of a machine to solve human-like tasks, while machine learning deals with the algorithms that perform these tasks.

1.1 Deep learning

Deep learning is a subset of ML that works with artificial neural networks. As their name suggests, they try to resemble the processes done by the human brain. When a human receives an information, the brain compares it to similar elements known before and interprets it by labelling and categorizing this information. The difference from ML is that deep learning uses multi-layer artificial neural networks that are composed of three main layers (as it can be seen in Fig. 1.1): the input layer, the hidden layer that actually covers more layers and the output layer.

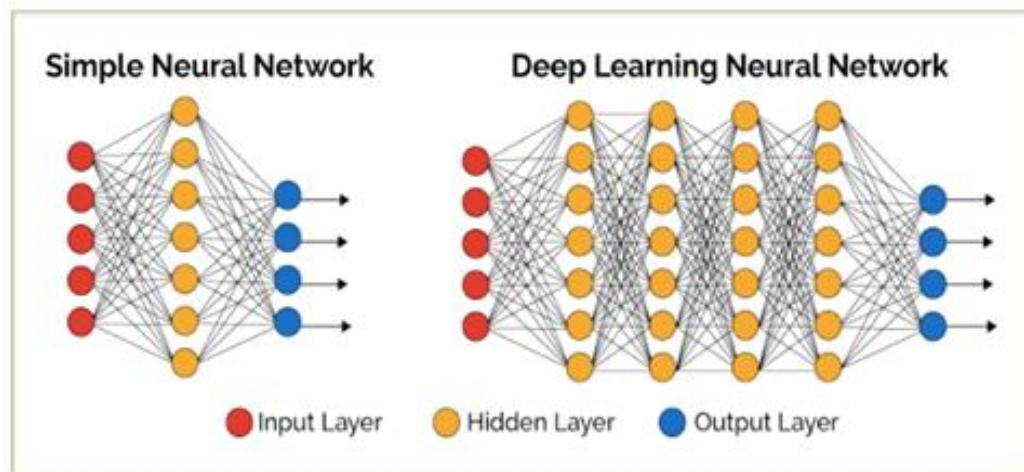


Figure 1.1 Structure of a neural network [3]

The input layer consists of nodes that represent the initial data for the network. No processing is done at this stage, they only pass the information to the hidden layers. Examples for this data can be pixel values in the domain of image processing or musical chords in audio processing. They essentially are various information from a specific field. Therefore, they are connected in a way directly to the outside world.

After the input nodes comes the hidden layer, which does not connect directly to the outside world: it serves as an activation function for computation and transfer of information from the input

to the output layers. The hidden layer can be formed of many other layers that collect hidden nodes. [4]

Finally, the output layer consists of output nodes that represent the results from the processed input information. In Fig. 1.2 there is a more detailed arrangement of a neural network.

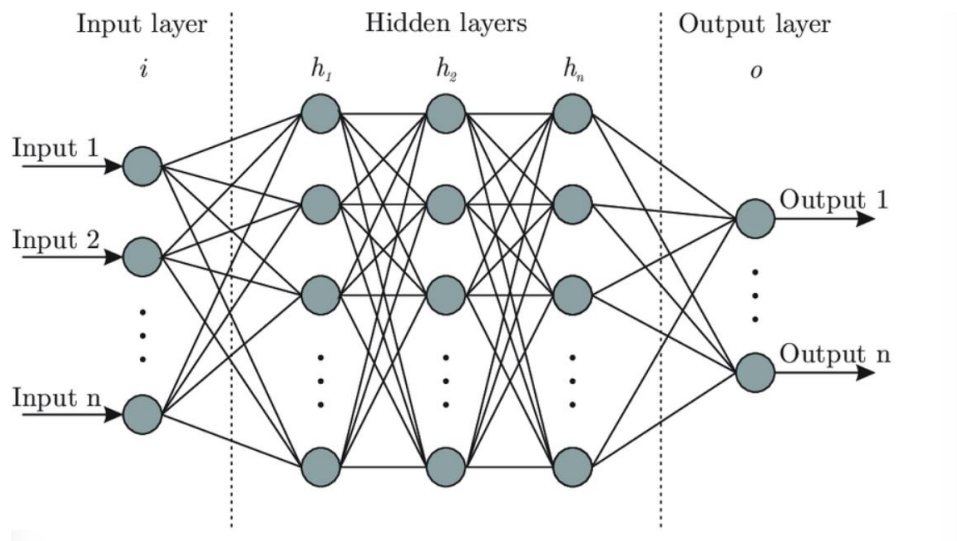


Figure 1.2 Network layers and connections [5]

To better understand the deep learning concept, one can observe Fig. 1.1. A simple neural network consists of only one hidden layer, while a deep neural network introduces more hidden layers. That means more computation power, more optimized information a better precision in identification and in output results. As mentioned before, these networks try to resemble the human brain processing of information. Thus, the layers described above behave like the biological neurons from the human brain: each of the nodes from a layer is connected to another one from the next layer. [3]

1.2 Biological neuron vs. Artificial Neuron

In order to see the resemblance and the grade of similarity between the two neurons, a comparison will be made in this subchapter.

The human brain has in its structure billions of interconnected neurons. Of course, it is almost impossible to reproduce this system in terms of a machine. However, the process of computation in terms of inputs and outputs is rather clear. A neuron has multiple dendrites, a nucleus and an axon. The input information is received through dendrites in the form of electrical pulses summed up and if the total electrical impulse power exceeds a threshold (the firing threshold for a neuron), the neuron transmits (outputs) another impulse along its axon. The axon, which has many terminals, distributes the output to numerous neurons connected to them.

Further, the similarities in structure of an artificial neuron with a biological one will be introduced. For a better visualisation, the neurons are presented in Fig. 1.3.

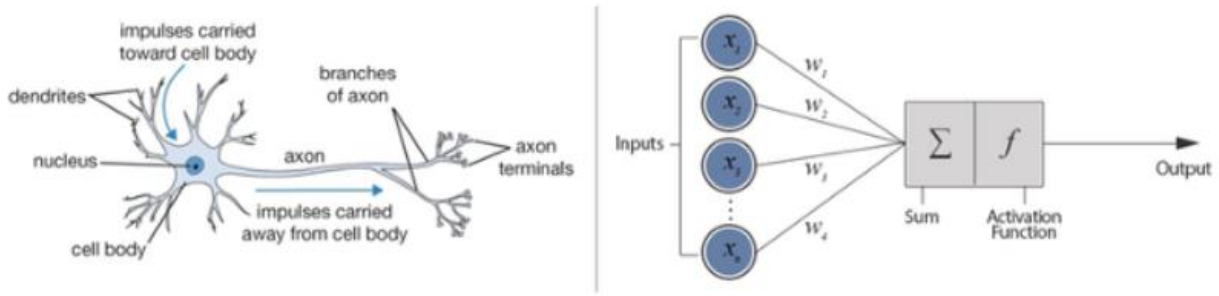


Figure 1.3 Biological Neuron vs. Artificial Neuron [6]

The analogy is as follows: the inputs from an artificial network are the dendrites, the sum and activation function represent the process inside the cell body and the output is the axon from the biological neuron.

1.3 Basic structure of a Neural Network

The artificial neural networks have a complex form and introduce terms like weight or bias. The weights can be described as parameters, usually found in the hidden layers of the network, that change the input in such a way that influences the output. Information is sent through the input nodes and after being passed to the hidden nodes it is multiplied by a weight, a numerical value and the resulting output can either be observed or passed to the next layer from the neural network. A low weight value will have little change on the input, while a larger value will have a considerable influence on the output results.

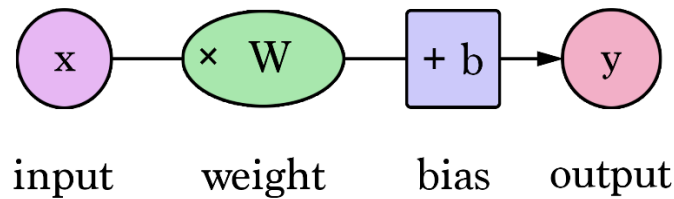


Figure 1.4 Parameters of the neural network [7]

In Fig. 1.4 it is revealed another parameter of the neural network: the bias. The weight illustrates how much influence has the input on the output, whereas the bias makes up the variation between the function's output and the intended output. [7] The output can be expressed in a simple equation, where N is the number of inputs and b is the bias:

$$Y = f(x) = \sum_{i=1}^N (X_i \cdot w_i) + b , \quad (1.1)$$

1.1 The Perceptron

The perceptron is the first and the simplest neural network, invented by Frank Rosenblatt in 1958. It is a linear classifier for data that applies a weighted sum on a set of inputs and passes it to a threshold activation function. It consists of four parts (Fig. 1.5): the input data, the weights and biases, the weighted sum and the activation function.

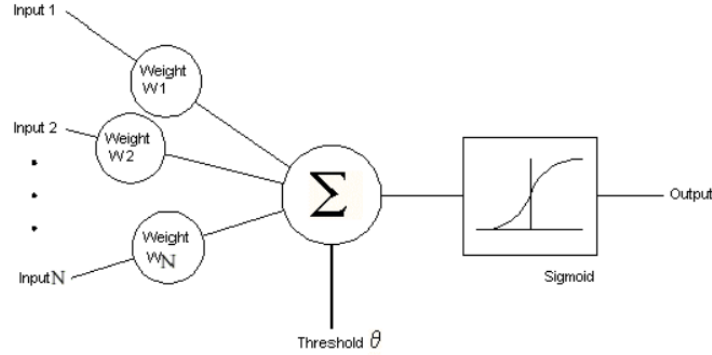


Figure 1.5 The Perceptron model [8]

The process is as follows: the inputs are multiplied by each weight, they are added and form a weighted sum that is evaluated by the activation function (a step function). This function labels the sum in two groups based on the threshold that is set and gives a boolean output. It is also known as ‘Linear Binary Classifier’. This network is the basis of the more complex structures developed until now and it is worth mentioning in Deep Learning methods.

There are three most commonly used neural networks in AI: feed-forward(FNN), recurrent(RNN) and convolutional neural networks(CNN).

1.2 Feed-Forward Neural Networks

Feed-forward neural networks, also known as multi-layered neural networks, are the first derived from the perceptron model. Their name comes from the fact that the information travels only forward in the network, from the input layer, through the hidden layer (that can consist of multiple layers) and finally reaching the output layer. It has a more sophisticated form than the perceptron and the complexity lays in the hidden layer that gives higher-order statistical information about the input data, not only boolean values. This becomes even more efficient when using the sigmoid function as an activation function.(Fig. 1.6)

$$\sigma(x) = \frac{1}{1 + e^{-x}} , \quad (1.2)$$

The curve of the function crosses 0.5 at an input of 0, for negative inputs the output is approximately 0 and for positive values it is approximately 1.

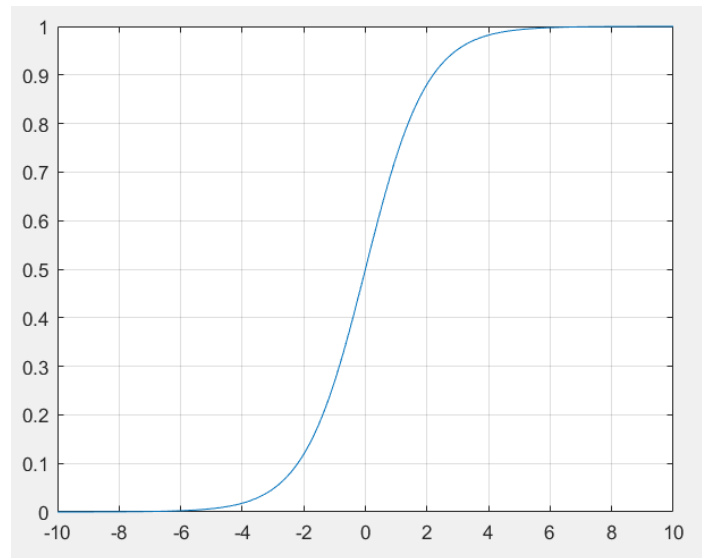


Figure 1.6 Sigmoid function

The sigmoid function introduces non-linearity in the network architecture, such that the output will not be a linear combination of the inputs, unlike the perceptron model. And each sub-layer from the hidden layer comes with a new representation and better visualization of the processed input. This makes it easier for a neural network to classify the patterns.

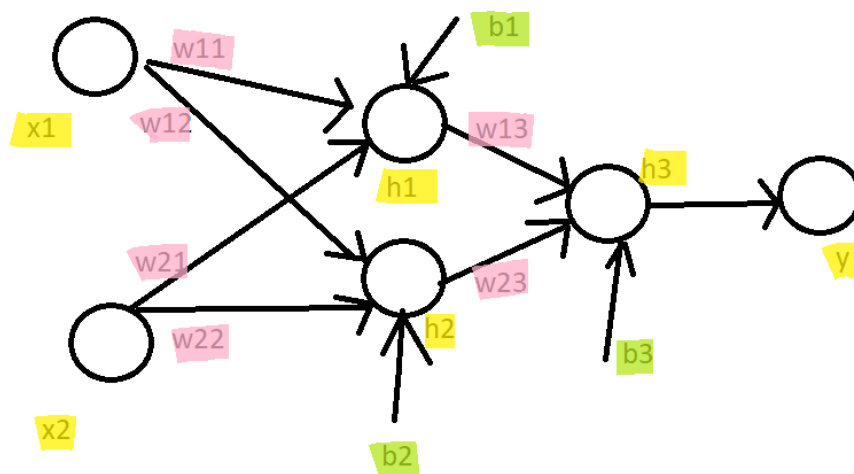


Figure 1.7 Neural network architecture with two inputs

In Fig. 1.7 it is presented a neural network with two inputs (x_1 , x_2) and two hidden layers consisting of three hidden nodes. In the first hidden layer we have the hidden nodes h_1 and h_2 . The inputs are connected to h_1 with the weights w_{11} and w_{21} and the biases b_1 and b_2 .

For each hidden neuron the inputs connected to it will be multiplied by the corresponding weights, summed and at the end the bias is added. Thus, some equations of the following form will be created:

- $h_1: z_1 = w_{11} \cdot x_1 + w_{21} \cdot x_2 + b_1$
- $h_2: z_2 = w_{12} \cdot x_1 + w_{22} \cdot x_2 + b_2$

The next step is to apply an activation function for these neurons. The choice of the Rectified Linear Unit function(ReLU) will be done. The function returns 0 for any negative input and the input itself in case of a positive value. It can be expressed as $f(x) = \max(0, x)$. Just as the sigmoid function, ReLU introduces non-linearity in the network structure.(Fig. 1.8)

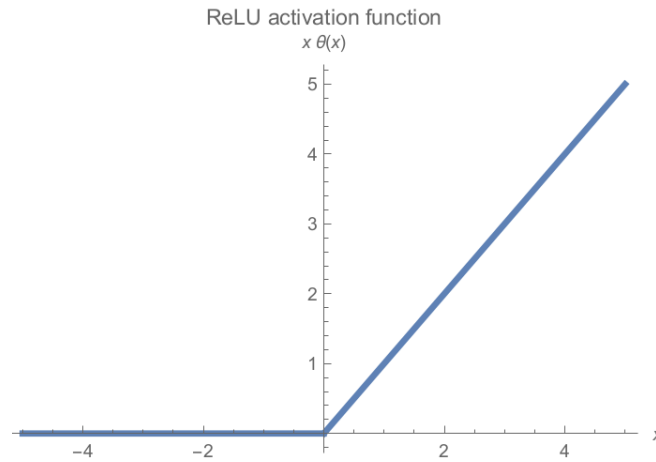


Figure 1.8 ReLU activation function [9]

Now the activation function is applied to the equations from the hidden layers and make a step to the next layer. This is a simple example on how neural networks work.

1.3 Recurrent Neural Networks

Recurrent neural networks differ from the FNN by having an extra connection, a feedback one, on the hidden layer nodes. They try to represent the recurrent connections from the brain and this can be achieved by introducing a temporal dependency. RNN can be described as dynamic systems that can make predictions based on the evolution in time of the neural network.

There are some advantages and disadvantages when using a RNN: they can have an input of any size and at each step in the network it is taken into account the history of the evolution, but the overall process is slower and the process of accessing information from the past is very complex.

1.4 Convolutional Neural Networks

Convolutional neural networks (CNN) are the most common structures used in deep learning, but mostly in image processing. They are generally composed of some main layers: convolution layers, pooling layers and fully connected layers. For a better understanding of this structure, an image classification network is illustrated in Fig. 1.9.

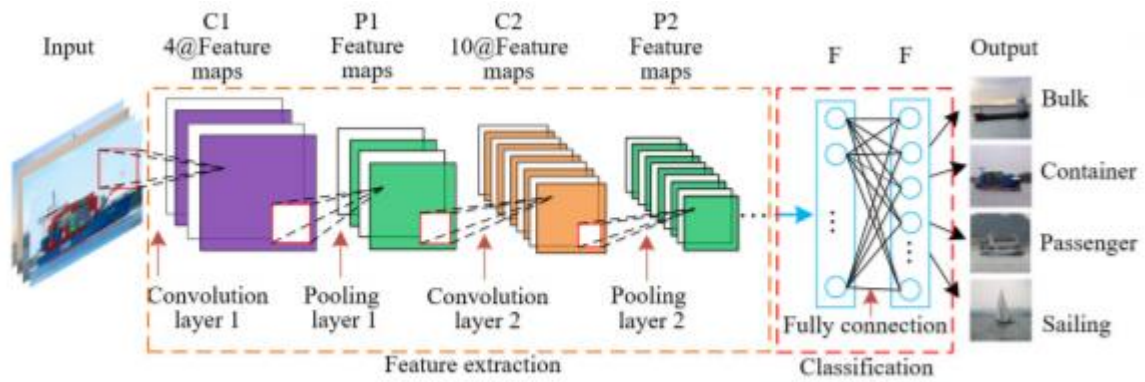


Figure 1.9 Convolutional Neural Network Structure [10]

A convolution layer is made of filters that perform sliding convolution while scanning the input in parallel. Its parameters are the size and the stride and they will be described in the following paragraphs. Each filter has a number of channels for which the parameters are changed. The pooling layer consists of a downsampling process that is usually applied after the convolution layers. Commonly max and average pooling are used in a CNN. Max pooling selects the maximum value from the output of the convolutional layer and average pooling makes an average of the values, finally each of them creating a feature map after the operations.

The fully connected layer is applied to a planed input where each part of the input is connected to the previous network structure. It represents the final stage of the CNN and it can be used for classification of the input.

A filter from the convolutional layer that is of size $F \times F$ is applied to the input I that has C channels. The resulting filter is a volume of $F \times F \times C$ that outputs a feature map of size $O \times O \times 1$. If the convolutional layer has K filters, the resulting output will be of size $O \times O \times K$.

The stride parameter S symbolizes the number of values by which the scanning operation is done. Zero-padding is a complementary parameter that represents the procedure of adding zeros to the margins of the input.

The CNN is a complicated model and the parameters vary with the input size. For a better understanding, the evolution of the parameters with the layers will be presented in Table 1.1. [1]

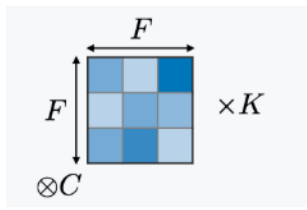
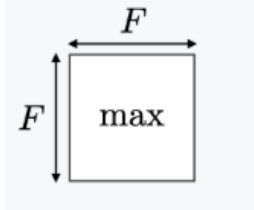
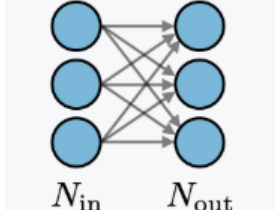
	CONVOLUTIONAL LAYER	POOLING LAYER	FULLY CONNECTED LAYER
Illustration			
Input size	$I \times I \times C$	$I \times I \times C$	N_{in}
Output size	$O \times O \times K$	$O \times O \times C$	N_{out}
Number of parameters	$(F \times F \times C + 1) \cdot K$	0	$(N_{in} + 1) \times N_{out}$
Remarks	<ul style="list-style-type: none"> • one bias / filter • $S < F$ usually • Commonly $K = 2C$ 	<ul style="list-style-type: none"> • pooling done per channel • usually $S = F$ 	<ul style="list-style-type: none"> • flattened input • one bias / neuron • any number of FC neurons (classes)

Table 1.1 Parameter evolution in a CNN [1]

The most commonly used activation function for all the outputs of the layers is the Rectified Linear Unit, presented before.

There is a need for a function that can determine the probability of something to happen or to exist. The softmax layer achieves this and it is a logistic function present at the end of the network. It takes an input vector of scores $x \in \mathbb{R}^n$ and has as output a probability vector passed through a softmax function. This is defined as: [1]

$$p = \begin{pmatrix} p_1 \\ \dots \\ p_n \end{pmatrix}, \text{ where } p_i = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}} \quad , \quad (1.3)$$

The Convolutional Neural Network structure was more described and highlighted in the previous subchapter from the other types because it will be implemented in this project.

CHAPTER 2 Music Perception and Representation

It is useful to have some knowledge about audio signals and audio processing in order to apply efficiently such a complex algorithm as an artificial neural network. Therefore, some main concepts will be detailed in this subchapter.

Audio processing is a subfield of signal processing that is focused on offering sound that can be described as agreeable for humans. To achieve this, a lot of algorithms have been developed over time that make use of digital processing and can reproduce songs at a high-quality level.

All of these processes are built in reference to the human hearing, that will be detailed in this chapter. As a matter of fact, human's perception of sound is the key to defining the audio signals. The three main elements that characterize a musical note (loudness, pitch and timbre) will also be presented here. [11] Since this project is tested with recorded guitar samples, the basic structure of a guitar will be detailed in this chapter.

2.1 Perception of sound

The common meaning of “sound” is the hearing sensation that a sound can produce. But there is another significance to this word, a physical one (sound wave): a variation of a physical quantity such as pressure, velocity in an environment with internal forces. This means that the humans don't perceive the sound in its essence, they sometimes don't perceive the sound at all at very low or very high frequencies. Besides, the perception can be different from one human to another, because the auditory system does not work in the same way for all.

There is a correspondence between the physical quantities of sound and the words used to describe what humans can hear.(Table 2.1)

Physical quantity	Perception
sound wave pressure/intensity	loudness
frequency	pitch
harmonic content	timbre

Table 2.1 Sound: physical vs. perception

The loudness is a measure of how strong or weak a sound is, pitch is defined in terms of high/low. The timbre is a more complex measure, independent of the pitch and loudness. It is determined by the harmonic content of a sound wave. [12] It is a good example to express the timbre in terms of speaking. The humans in most cases speak with the same fundamental frequency (different for women and men). But one can discern a familiar voice from a new one and the voice timbre makes the distinction here. This can be applied also for musical songs, where the style of an artist can be distinguished from another one's.

By biological construction, the human ear can hear in a logarithmic manner, but a more detailed presentation can be found in [12]. In terms of frequencies, the human hearing range is between 20 Hz and 20 kHz, but there are major variations from one human to another. The higher a frequency is, the more high-pitched is the sound perception.

2.2 Sound Level

The amplitude of sound is measured in decibels (dB), a logarithmic scale working with powers of 10, where a small increase in dB equals with a large change in signal's energy. Therefore, if a sound is described as being 30 dB higher than another one, it means that it is 1000 times higher than that sound. The decibel is a dimensionless unit that calculates the ratio of (in this case) the measured sound pressure over the reference pressure ($p_{ref} = 20 \mu\text{P}$) that corresponds to 0 dB. It is called Sound Pressure Level and it can be defined as [12]:

$$L_p [\text{dB SPL}] = 20 \lg \frac{p}{p_{ref}} \quad , \quad (2.1)$$

,where p represents the effective sound pressure.

The use of this logarithmic scale is to better reproduce the human hearing. [13] In order to better understand the correlation between decibels and common sounds and how can this influence our hearing system, one can take a look at Fig. 2.1. It is important to know that sounds over the 120 dB limit can be harmful for a human ear.

Sound (dB)	Sound noise (with distance)
0 dB	Hearing threshold
10 dB	Distant rustling of leaves
20 dB	Whisper close up
30 dB	Quiet rural area
40 dB	Quiet library
50 dB	Conversation at home
60 dB	Conversation in a bar
70 dB	Vacuum cleaner at 3ft. (1m)
80 dB	Close alarm clock
90 dB	Operating a lawn mower
100 dB	Speaker in a club 3ft. (1m) away
110 dB	Vehicle horn 3ft. (1m) away
120 dB	Chain saw close up (discomfort)
130 dB	Jack hammer (pain threshold)
140 dB	Jet engine (pain threshold)
150+ dB	Eardrum rupture

Figure 2.1 Decibels and sound sources [13]

2.3 Note and Chord Structure

Music is represented through what are called musical notes, the symbols that help in reproducing and describing a song. Each note corresponds to a frequency, measured in Hertz (Hz). The equivalent of the musical notes, in the audio field, are the letters of the alphabet that compose

words, the fundament of verbal communication. The Western musical system is constructed by 7 natural notes (Do, Re, Mi, Fa, Sol, La, Si), which correspond to C, D, E, F, G, A, B , and another ten alterations of them: sharps (Do#, Re#, Fa#, Sol#, La#) and flats (Reb, Mib, Solb, Lab, Sib). The sharps raise a note by a semitone and the flats lower the note by a semitone. The next paragraph describes better the measures and ratios between notes.

An interval between two notes is defined as the ratio of the corresponding frequencies of the notes. It is called a tone and is composed of two semitones. Thus, the semitone is situated halfway between two notes and is the smallest step in Western classical music. An octave is a musical interval with a frequency ratio of 2:1.

The piano keyboard is the actual physical representation of the tones and semitones distribution of 12 notes in an octave. (Fig. 2.2) This is the most common distribution of notes, where all semitones are equal as intervals, called the Equal Tempered scale. Therefore, the interval between two notes distanced by N semitones is $2^{\frac{N}{12}}$ and for each value of N , from 1 to 12, there is a notation given for the interval (more about intervals on [14]). [12]

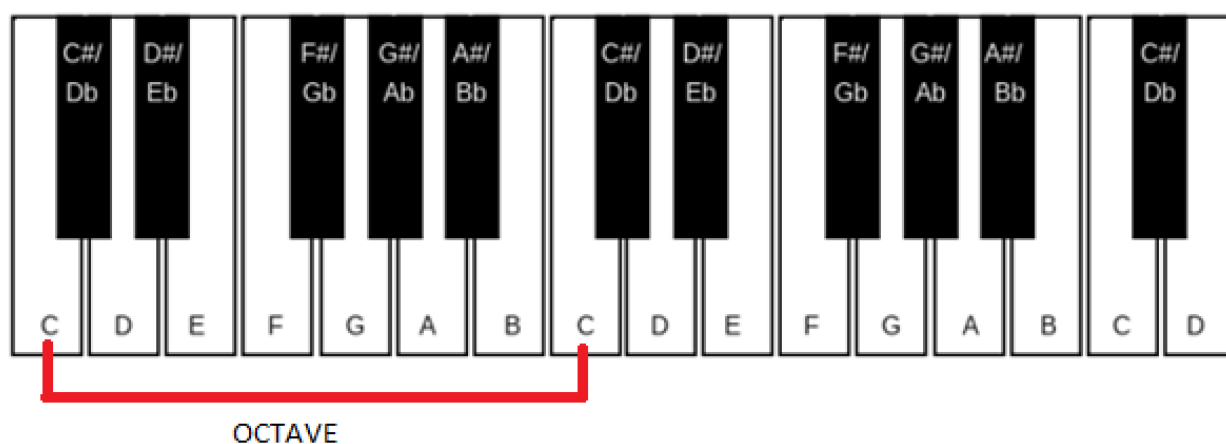


Figure 2.2 Piano keyboard: notes and octave representation [15]

On the other hand, music can be expressed in terms of chords. A chord is a set of several tones played simultaneously. In Western music, the most frequent chord is the triad, which is composed of three distinct notes: a root, the third and the fifth above it. There are many variations of them, but the most common are the minor and major triads. [16]. An example is shown in Fig. 2.3 , where C major and minor chords are represented on a piano keyboard.

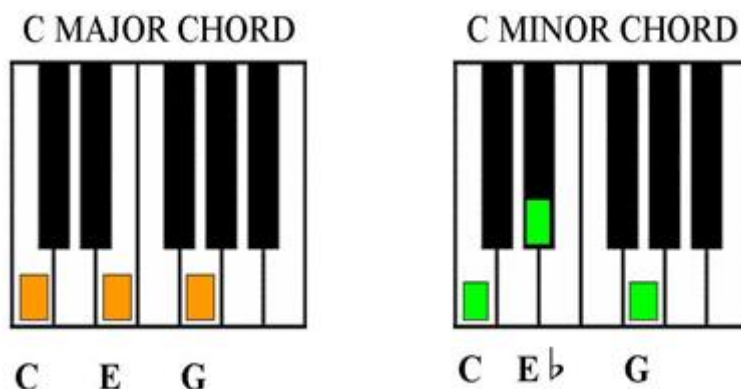


Figure 2.3 C major and C minor chord [17]

These two types of chords will be further used in the project for the artificial neural network training.

2.4 Types of Instruments

Music can be reproduced either by voice, or by musical instruments. As the project is implemented for chords produced by instruments, a classification of them will be summarized, with an accent on the guitar structure in the end.

The “*Sachs-Hornbostel*” is a system of musical instruments classification, formulated by C. Sachs and E. Moritz von Hornbostel and published in 1914. It classifies the musical instruments in four categories, depending on the fundamental vibrating material that produces the sound. Subsequently, another category was added for the electronic instruments. [12] The categories are presented in Table 2.2:

Category name	Source of sound production	Examples
Idiophones	vibrating solid material itself	<ul style="list-style-type: none"> • cymbals • xylophones • maracas • bell
Membranophones	vibrating stretched membranes or skin	<ul style="list-style-type: none"> • timpani • cuica • tambourine • bass drum
Chordophones	stretched vibrating string	<ul style="list-style-type: none"> • grand piano • guitar • sitar • violin • harp
Aerophones	vibrating a column of air	<ul style="list-style-type: none"> • accordion • harmonica • flute pipes of an organ • oboe • clarinet • saxophone
Electrophones	electronically	<ul style="list-style-type: none"> • electronic organ • synthesizers
	electronically amplified	<ul style="list-style-type: none"> • electric guitar • electric piano

Table 2.2 Classification of instruments [18]

Considering that the test database was constructed using recorded guitar audio samples, it is useful to understand the structure of a guitar and how is the sound delivered to the audio interface.

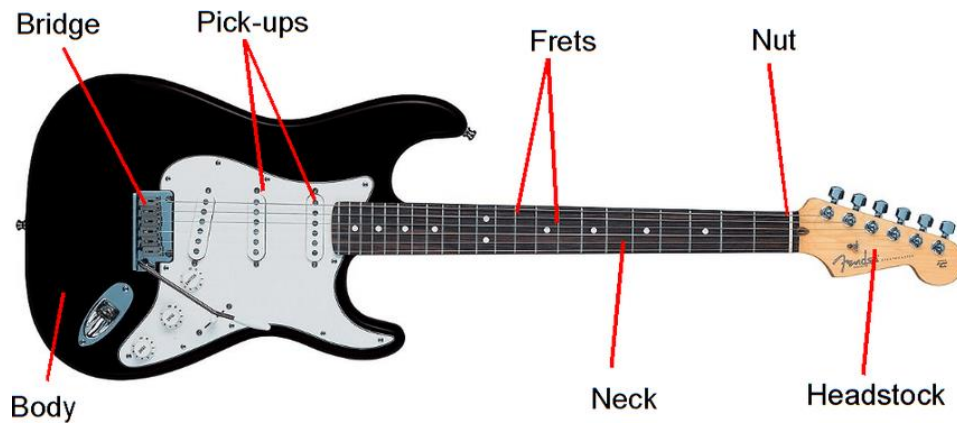


Figure 2.4 Main components of an electric guitar [19]

The head of the guitar (Fig. 2.4) is the end part of the neck that holds the tuners. The tension on the string is sustained by the string retainers. A fretboard (or fingerboard) is the top part of the neck where frets and fret markers are positioned. Frets are used for raising the pitch of a note and fret markers for knowing the position on the fretboard.

Usually a guitar has six strings and how each of them is picked denotes a musical note. [20] The main parts that describes this instrument as an “electric” one are the pickups, that can be found on the body of the guitar. They are electronic components that convert the string vibration into small electric signals that are afterwards passed through the output jack to an amplifier.

	OPEN E	F	F \sharp /G \flat	G	G \sharp /A \flat	A	A \sharp /B \flat	B	C	C \sharp /D \flat	D	D \sharp /E \flat	E
	OPEN B	C	C \sharp /D \flat	D	D \sharp /E \flat	E	F	F \sharp /G \flat	G	G \sharp /A \flat	A	A \sharp /B \flat	B
	OPEN G	G \sharp /A \flat	A	A \sharp /B \flat	B	C	C \sharp /D \flat	D	D \sharp /E \flat	E	F	F \sharp /G \flat	G
	OPEN D	D \sharp /E \flat	E	F	F \sharp /G \flat	G	G \sharp /A \flat	A	A \sharp /B \flat	B	C	C \sharp /D \flat	D
	OPEN A	A \sharp /B \flat	B	C	C \sharp /D \flat	D	D \sharp /E \flat	E	F	F \sharp /G \flat	G	G \sharp /A \flat	A
	OPEN E	F	F \sharp /G \flat	G	G \sharp /A \flat	A	A \sharp /B \flat	B	C	C \sharp /D \flat	D	D \sharp /E \flat	E
						V	VII	IX	XII				

Figure 2.5 Guitar strings and chords [21]

This is the main principle of how sound is produced through an electric guitar. The output electric signal from the guitar can also be passed to an audio interface directly, to eliminate noise during a recording.

CHAPTER 3 Chord recognition system

In order to recognize chords from musical pieces, a list of algorithms can be applied so that the chords can be easily identified. This would be the first step. The second one following is to extract the chroma feature or, simply, the chromagram. The chromagram represents the energy distribution of a signal's frequency content across the 12 pitch classes of the equal-tempered scale. [22]

3.1 Chromagram Computation:

The most common approach for chroma feature calculation is to apply the constant Q transform. It is well known that humans hear an approximate range of frequencies from 20 Hz to 20 kHz and their perception of sound has a logarithmic response to the pressure level. Therefore, an exponential representation of this spectrum is needed with the purpose of reproducing effectively the human auditory range. Considering this fundamental aspect, the constant Q transform (CQT) will be described in detail.

3.1.1 Constant Q Transform

CQT is a variation from the Short Time Fourier Transform (STFT), a linear time-frequency analysis implemented on sections of a signal using time windows. Each evaluated window provides frequency and phase content individually. The mathematical way to describe the STFT is:

$$X[m, \omega] = \sum_{n=-\infty}^{+\infty} x[n]W[n-m]e^{-j\omega n} , \quad (3.1)$$

, where $W[n-m]$ is the window function centred at moment $n=m$ that establishes the frequency bin width and position. In discrete time we have:

$$X[k] = \sum_{n=0}^{N-1} x[n]W[n]e^{-j\frac{2\pi kn}{N}} , \quad (3.2)$$

STFT can be used, but there appears a problem to this: the bin width is the same for each type of audio signal. Thus, the musical signal will be interpreted in the same way, whether is the sound from a piano, a bass guitar or a drum. Moreover, the standard STFT has equally spaced frequencies because the exponent increases linearly with 'k'.

As mentioned before, human ear responds to sound in a logarithmic manner. Therefore, using the same bin width for different frequencies will give uneven resolutions, causing a loss in important information and reduced quality of the audio signal. Fortunately, there exists a process that can handle all frequencies in the same way: the Constant-Q Transform, an algorithm that distributes the frequencies with a logarithmic step over the spectrum. [23]

Quarter tone spacing was used for this CQT evaluation, a method that divides an octave into 24 equal steps. This is unusual for Western music which is based on 12 notes. Quarter tone music can be found in Oriental culture. That is why this kind of music can be sometimes identified as out-of-

tune for some people. Their ears have been trained with Western music and every other different note they hear seems odd.

It is difficult, but not impossible, to play quarter-tone music on a piano. While looking at the keys from an octave there are physically 12 keys that represent the notes from the scale, with a half step distance between two notes. But when we think about a violin or a flute, there are no such constraints on choosing from 12 notes and other notes can be added between these 12. For example, the strings from a violin can produce intermediate frequencies besides the standard 12. This is how quarter tone music can be produced. [24]

The choice of quarter tone spacing was made because it gives better resolution taking into account all instruments and their behaviour. For this reason, the resolution will be $\delta f = (2^{\frac{1}{24}} - 1)f = 0.029f$ and a constant ratio of frequency to resolution, expressed through the Q parameter (hence the presence of Q in the name of the transformation): $Q = f/0.029f = 34$. [2]

$$X[k] = \frac{1}{N(k)} \sum_{n=0}^{N(k)-1} x[n] W_k[n] e^{-j \frac{2\pi Q n}{N(k)}}, \quad (3.3)$$

The expression from above is for the k th component with the frequency $f_k = (2^{\frac{k}{24}})f_{min}$, where f_{min} represents the minimum frequency from an audio signal that will be processed. It was chosen to be 32.7 Hz, corresponding to the musical note C1.

A comparison between the variables from the traditional Fourier transforms and CQT will be presented in the table below:

	CQT	STFT
Frequency	$(2^{1/24})^k \cdot f_{min}$ exponential in k	$k \cdot \Delta f$ linear in k
Window	variable = $N[k] = \frac{SR \cdot Q}{f_k}$	constant = N
Resolution Δf	variable = $\frac{f_k}{Q}$	constant = $\frac{SR}{N}$
$\frac{f_k}{\Delta f_k}$	constant = Q	variable = k
Cycles in window	constant = Q	variable = k

Table 3.1 Comparison of variables: Discrete Fourier Transform and CQT [2]

Having the sampling rate f_s , the length of the window for each frequency f_k is $N(k) = (f_s/f_k)Q$. The Q parameter can be expressed in Eq. 3.4 as the ratio between the central frequency of the window f and the resolution between frequencies δf . $W_k[n]$ represents the window function and has the same shape for each k component, but the size of it varies, depending on the frequency f_k . The equation is normalized by dividing the sum by $N(k)$, since the number of terms varies with k . The parameter can be expressed as:

$$Q = \frac{f}{\delta f} = \frac{1}{\frac{1}{2^{24}} - 1}, \quad (3.4)$$

A Hamming window was used, of the form:

$$W[k, n] = \alpha + (1 - \alpha) \cos\left(\frac{2\pi n}{N[k]}\right), \quad (3.5)$$

, where $\alpha = 25/46$ and $0 \leq n \leq N[k] - 1$. The window is an optimization of the Hann window and when α is adjusted to this value, the first sidelobe from the Hann window is cancelled. Fig. 3.1 shows the differences between the two windows.

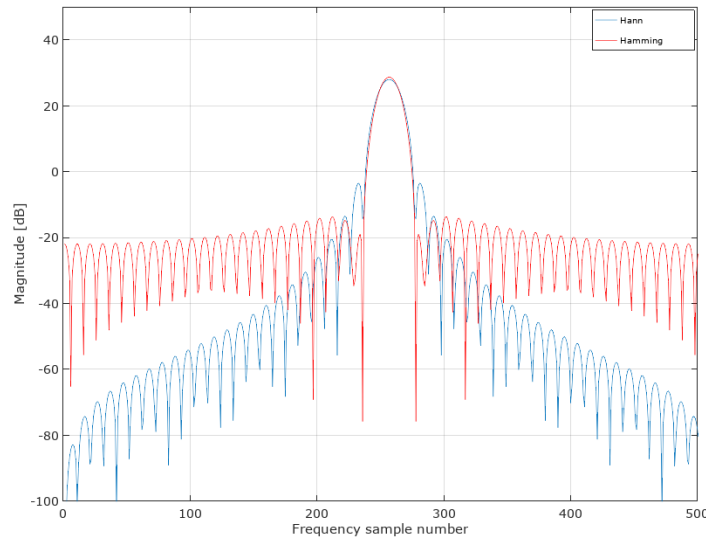


Figure 3.1 Representation of Hann and Hamming windows [25]

The Hamming window was chosen because it has a better performance in the process of recognizing the main frequencies from a signal and also reduces the spectral leakage phenomenon that may appear during processing.

3.1.2 Implementation of the CQT

A function was developed in *Matlab* for the Hamming window [Annex 1(a)]. The Constant-Q Transform was implemented on the same audio input signal using the function from Annex1(b). A representation of it is shown in Figure 3.2, where the main frequencies are plotted on a logarithmic scale. These frequencies are 36.7 Hz (D1 note), 110 Hz (A2), 220 Hz (A3), 7039 Hz (A8). From these information it can be affirmed that some main notes in the audio signal are D and A. The higher frequencies at which the A note is represented can be harmonics of the main A1 note. This can be further analysed by dividing the signal in frames and performing CQT not on the entire signal, but on individual frames in order to see the power of each note.

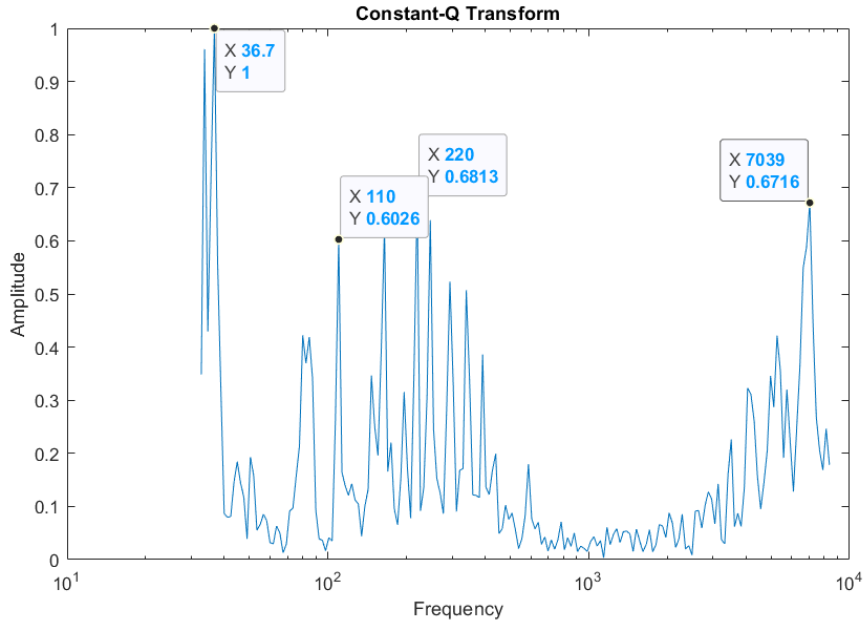


Figure 3.2 Constant-Q Transform

The CQT was performed on a number of 8 octaves, divided in 24 bins starting with a minimum frequency of 32.7 Hz, corresponding to C1 note. Therefore, a number of 193 frequencies are analysed. The signal was divided in frames of 20 ms and processed individually for each of these 193 frequencies, having as effect multiple frames with the distribution of frequencies over time. More than this, the choice of the window size for the CQT was made based on the frequency that is processed.

Below there is a fragment of the CQT function from Annex1(b), with the parameters *NoOctave* and *BinsPerOctave* which can be modified according to different evaluations. These lines of code compute Q , the ratio of frequency to resolution, the frequencies in the spectrum f that depend on the number octaves and of bins in which an octave is divided and, finally, N that is composed of the window length for each frequency from f . As a consequence, the Q parameter will be 34, f and N will have 193 values each.

```

k = 0:NoOctave*BinsPerOctave;
Q = floor(1/(2^(1/BinsPerOctave)-1));
f = 2.^(k/BinsPerOctave)*fmin;
N = round(fs*Q./f);

```

The choice of these parameters was made because a wide range of frequencies is needed in order to have a better resolution when determining chords from audio signals. After some experiments with 12 and 36 bins per octave, 24 bins have shown to give a reasonable accuracy using this algorithm.

The algorithm was implemented using the function from Annex1(c) and the results were represented in Fig. 3.3 (left) for a period of several seconds. The step size for the transformations is 20ms (corresponding to 882 samples) and the number of frames depends on the length of the input signal.

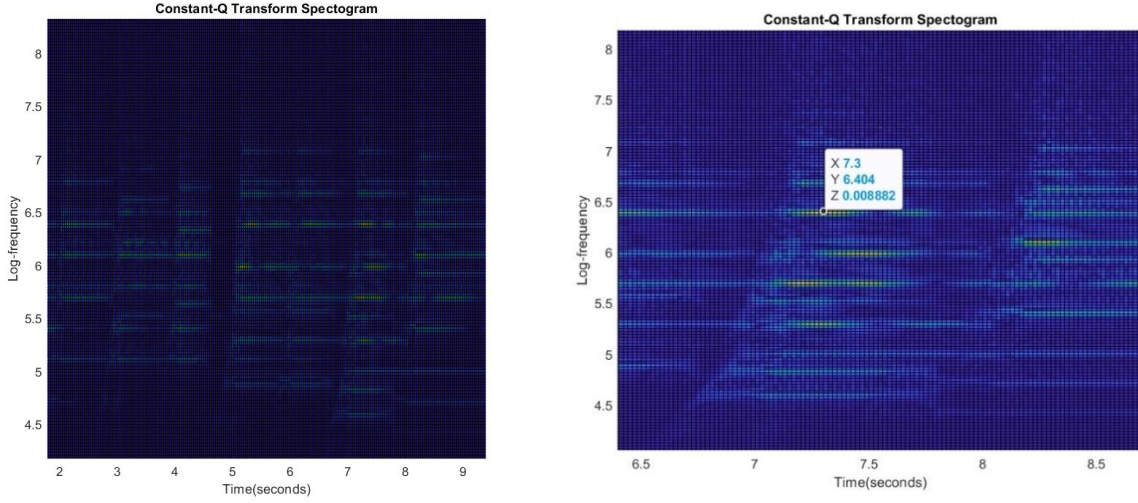


Figure 3.3 Constant-Q Spectrogram (left) and zoomed plot (right)

A more detailed focused plan in on the right side of the figure, when it can be clearly noticed that each frequency (each note) comes with other harmonics. At 7.3 seconds, the pale blue lines that appear on the spectrum illustrate the harmonics and overtones, while the orange-yellow portions depict the actual note that is played.

3.1.3 Post-Filtering the Constant-Q Transform

As mentioned before, the human ear does not perceive the sounds in a linear manner. A pure tone of under 200 Hz (a fundamental frequency that corresponds to the vocal signal of a woman) is considered dull and particularly new for the human auditory system. This happens because in everyday life there are no signals with the fundamental frequency as described before. Actually, they exist, but they come up with many harmonics that indeed compose the timbre of a sound. Consequently, in signal processing techniques this factor should be taken into account, because the physical sound does not correspond to the perceived sound by humans. The human ear is more sensitive to the medium frequencies in the auditory range (3000 – 5000 Hz). [12] Due to this fact, low and high frequencies need more amplitude power in order to level up with the mid-frequencies.

A study done by H. Fletcher and W. A. Munson in 1933 [26] first described the term “loudness” and introduced the *Sound Power Level*. This study has shown that the loudness perceived by humans is relatively linear proportional to the SPL, described by:

$$SPL = 10 \log_{10} \left(\frac{N}{N_{ref}} \right) , \quad (3.6)$$

, with N being the sound power (W) and $N_{ref} = 10^{-12}$ Watts, which is the reference sound power. This particular value comes from the fact that it is the lowest power of a sound that an excellent human auditory system can discern. As an analogy with examples from the real world, a sound power of 10^{-11} W corresponds to the sound of the human breath. [27]

A method was adopted from [28], namely the A-weighting, that takes into account the loudness perceived by the human ear.[Annex1(d)] It computes weights for each of the frequencies in the chosen range, in this case starting from $f_{min} = 32.7$ Hz, on 8 octaves with 24 bins per octave. The weight for the frequency f_i in the range is computed as:

$$R_A(f_i) = \frac{12200^2 \cdot f_i^4}{(f_i^2 + 20.6^2) \cdot \sqrt{(f_i^2 + 107.7^2)(f_i^2 + 737.9^2)} \cdot (f_i^2 + 12200^2)}, \quad (3.7)$$

, $i = 1, \dots, k$ where k represents the index of the last frequency analysed in the spectrum. In this case, we have 193 frequencies analysed. The values from the equation above are standard, taken from the Fletcher and Munson curves. [26] A value is added to each weight as follows:

$$A(i) = 2.0 + 20 \log(R_A(f_i)), \quad (3.8)$$

The final step is to apply the A-weight algorithm to the log-spectrum values pre-computed with the CQT:

$$X_A(i, n) = 10 \log X_{cqt}(i, n) - 10 \log R_p + A(i), \quad (3.9)$$

, for $i = 1, \dots, k$ where k is the number of frequency bins per octave, $n = 1, \dots, N$ with N = number of frames and R_p is the standard reference power level, equal to 10^{-12} .

This method was applied after having the values for the CQT spectrogram. However, there was no major difference observed on the A-weighted CQT spectrogram. Thus, having no considerable impact on the entire algorithm it was excluded.

3.1.4 Representation of the chromagram

As the constant-Q spectrogram was computed, the next step towards a chromagram representation is to add values for the same frequency (note) from each octave. This can be computed using the equation:

$$C_{cr}(b, n) = \sum_{m=0}^M |X(b + m\beta, n)| \quad (3.10)$$

, where M represents the total number of octaves on which the analysis was performed, β is the number of bins per octave and $b \in [1, \beta]$. The resulting values are in a form of a matrix with 24 rows, the numbers of bins per octave, and N frames of 20 ms in which the time is divided. Each row describes the intensity of a frequency in time.

The dimension of the matrix will be reduced to 12 rows, corresponding to the 12 pitch classes in standard Western music (C, C#, D, D#, E, F, F#, G, G#, A, A#, B), by choosing the maximum value of two consecutive bins. A pitch class is characterized by all pitches from the octaves that share the same chroma feature. The outcome is a chromagram that represents the evolution in time for each pitch class and how their energy is distributed along the signal. A function was implemented in Annex1(e) and the outcome is represented in Fig. 3.4.

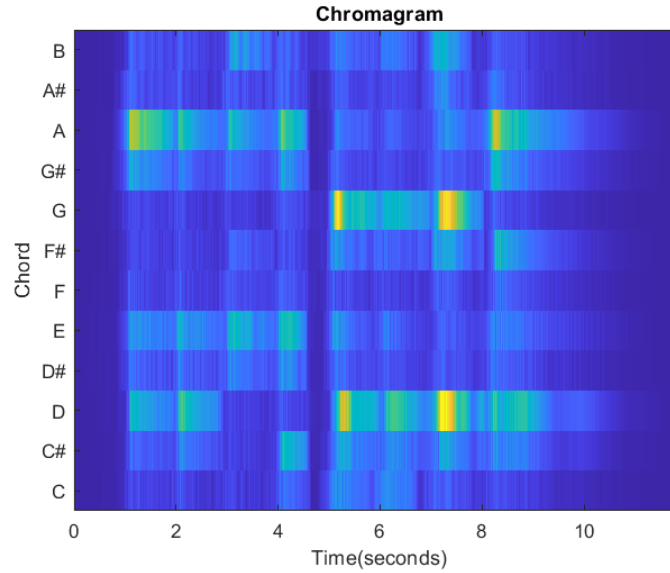


Figure 3.4 Chromagram of a 12 seconds guitar audio sample

3.2 Post-filtering the chromagram

A chromagram comes with many noisy parts and the main justification is that an audio signal is not a sequence of ideal frequencies. The truth is that each note played on an instrument comes along with other frequencies, named harmonics and overtones. But they are not as loud as the fundamental one or, in other words, they do not have a strong pitch. An overtone is used to describe any resonant frequency above the fundamental frequency and it may or may not be a harmonic. The harmonics refer only to integer multiples of it. These overtones actually make the instruments to sound different from each other and the characteristic that makes them distinct is called timbre. This attribute allows the human ear to distinguish between sounds that have the same fundamental frequency. A proper example to this statement is pictured in Fig. 3.5, where it is clearly shown that the spectrum differs for the same note played at a piano and a guitar.

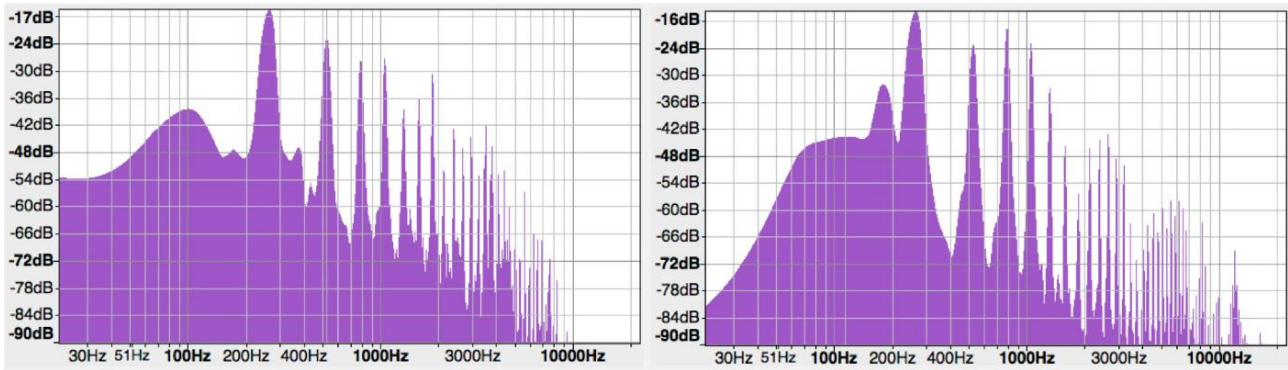


Figure 3.5 Note C3 played on the piano (left) and on the guitar (right) [29]

For better precision and identification, ideally a chroma feature should be composed of only the fundamental frequencies from the analysed audio signal. However, there are also some methods to emphasize these fundamental frequencies and reduce the other spectral components.

In this paper the aim is to identify the chords from an audio signal. Considering this, some filtering techniques should be applied in order to have an optimal chord recognition system.

3.2.1 Moving-average filter

In order to get a continuous passing from one frame to another in the chromagram, a filter should be used that smoothens the edges of the chords. This helps for a better identification and representation of chords over time.

A moving average filter is implemented, that can be calculated as:

$$\hat{c}(n) = \frac{1}{N} \sum_{\tau=0}^{N-1} c(n + \tau) \quad (3.11)$$

, where $c(n)$ is a frame from the chroma feature computed before, $\hat{c}(n)$ is a frame of the filtered chroma and n is the frame index. This filter smoothens the transition from one chord to another, making it easier to detect them. [30] N represents the number of points for the average.

The function in Annex1(f) takes as input the N number of points and the chroma matrix, and applies a moving average filter on the number of points for each row of the matrix. Initially the number of points was chosen to be $N = 10$ and after more experiments, the value of $N = 20$ gave a better result, without modifying relevant parameters of the feature. The algorithm was applied on a 23 seconds audio input signal. The chromagram before filtering is presented in Fig. 3.6.

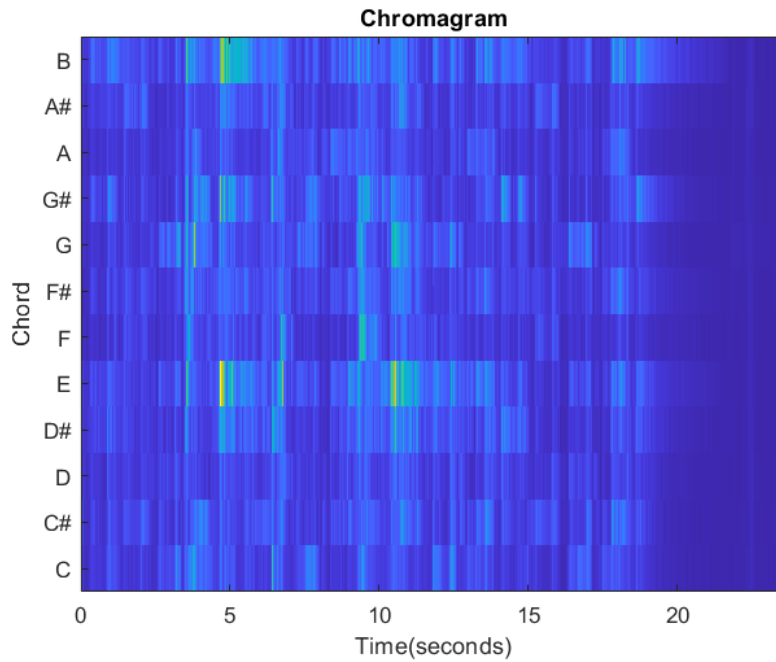


Figure 3.6 Chromagram before filtering

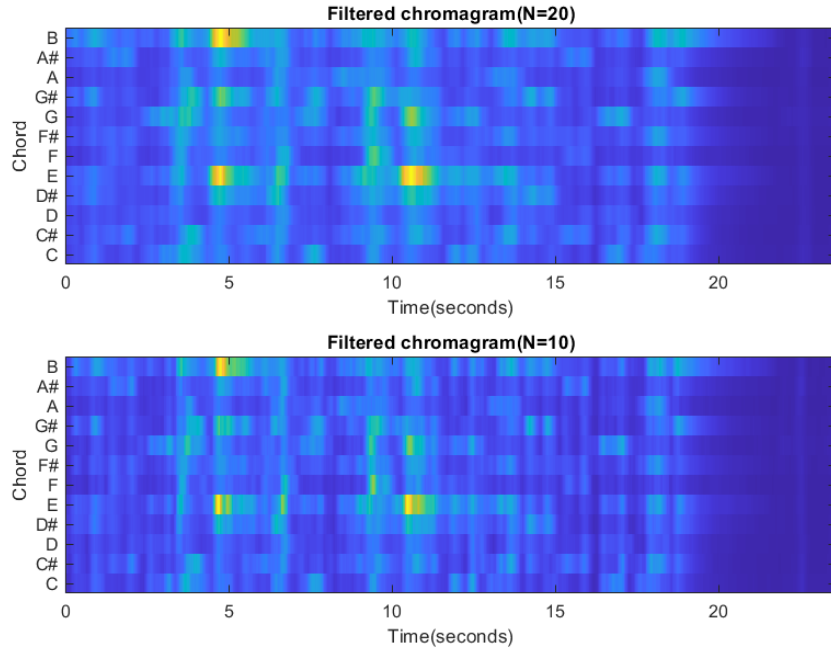


Figure 3.7 Chromagram passed through a moving-average filter

In Fig. 3.7 it is relevant to see that the chord detection is more efficient when using a 20-point moving-average filter.

3.2.2 Geometric mean

In digital signal processing there are many ways to filter and process data. Another filtering technique, besides the moving-average filter, that was experimented is the geometric mean. [31] , [32] While it is a common method in image processing for noise reduction, it can easily be applied on the chromagram. The filter is able to smooth the feature and improve the detection of chords. It can be calculated as:

$$C_g(b, n) = \left\{ \prod_{t=n-\frac{N-1}{2}}^{n+\frac{N-1}{2}} C_{cr}(b, t) \right\}^{\frac{1}{N}} , \quad (3.12)$$

, with b representing the bin index, n the frame index and N the points for filtering.

The developed function for the filter can be found in Annex1(g) and a comparison between two filtered chromagrams (for the same feature extracted from the audio signal in Fig. 3.4) with N=15 and N=10 is illustrated in Fig. 3.8.

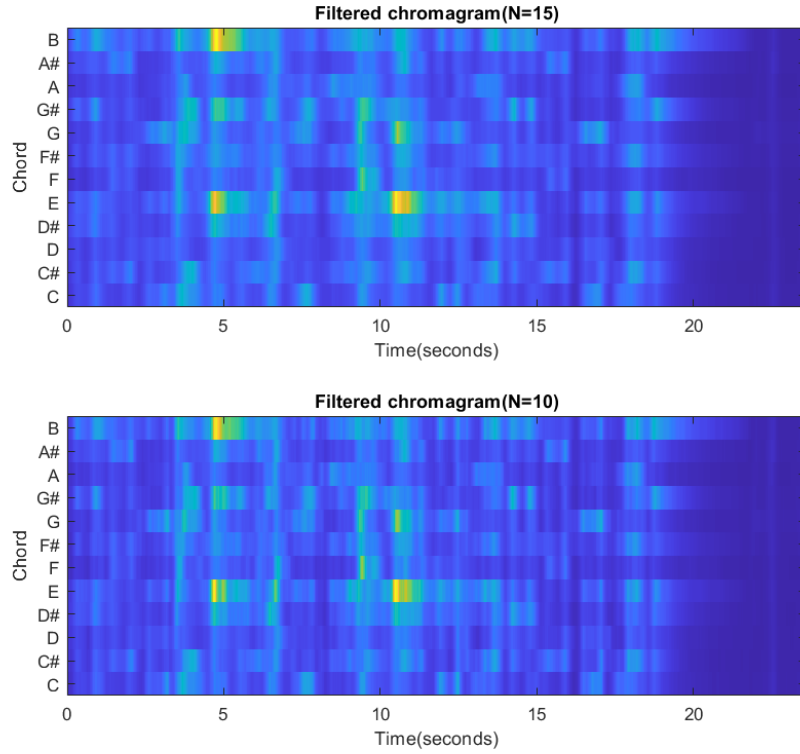


Figure 3.8 Chromagram passed through a geometric-mean filter

For a better approximation and preservation of the chord edges the choice of $N=15$ is more reasonable.

3.3 Onset detection

Onset detection represents the method of detecting the starting moment of notes from an audio signal. Until this point, a chroma feature was computed for an audio signal, but the signal is composed of many chords. In order to detect each chord individually, an onset detection function must be implemented.

As it can be observed in Fig. 3.2, an audio signal is a very complex signal, that illustrates the summation of many other ideal signals, with different frequency and amplitude values. When a note is played, the amplitude of the increases, the frequency changes and the signal is invariably fluctuating. This represents a physical change and characterization of the signal. But the human's perception is distinct: it is more associated to the variation of the energy, pitch and timbre in musical sounds.

There are three main concepts to be discussed concerning this topic that can actually characterize the presence of a note in a signal: attack, transient and onset. [33] Figure 3.9 presents the concepts on an ideal played note.

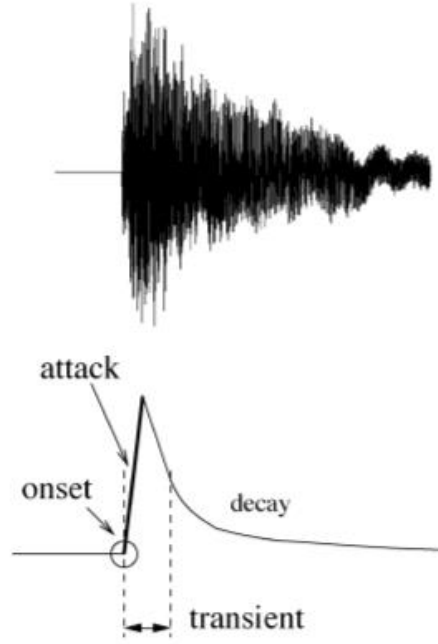


Figure 3.9 Onset, attack, transient and decay in the ideal case [33]

The onset may be defined as the beginning of a note or the instant transformation of the signal's state. Then the attack is the effect of a quick rise of the signal and it is represented as a slope (in the ideal case) in the figure above. In reality, it is a noisy part with different step values. The interval between when the note is played and the signal decays is called the transient. During this interval a considerable change in energy takes place. This is one main aspect that should be taken into consideration when implementing a function for onset detection.

There are some approaches for this implementation, but the energy-based method was chosen in this project. The first step is to construct an energy envelope of the signal. Before that, the signal is filtered using a Hamming window (with the use of the function from Annex1(a)). The equation that constructs the energies by smoothing and filtering:

$$E(n) = \frac{1}{N} \sum_{i=-\frac{N}{2}}^{\frac{N}{2}-1} [x(n+i)]^2 w(i) , \quad (3.13)$$

, where $w(i)$ is the N -point Hamming window function centered at $i=0$. [33] The value for the number of points in this function is 2000. The onset detection function from Annex1(h) takes as input the audio signal and sampling frequency and outputs the energy envelope, the points where the maximum values of the signal are situated and the time vector.

An additional called function [Annex1(i)] computes the energy envelope, based on the equation above. The number of points N can be adjusted. The energy signal is then filtered and normalized, in preparation for the peak identification.

Having the samples at which the peaks occur, the onset detection function can be further applied between the peaks, where intuitively a chord is played. A representation of the energy envelope along with the peaks (red lines) in in Fig. 3.10.

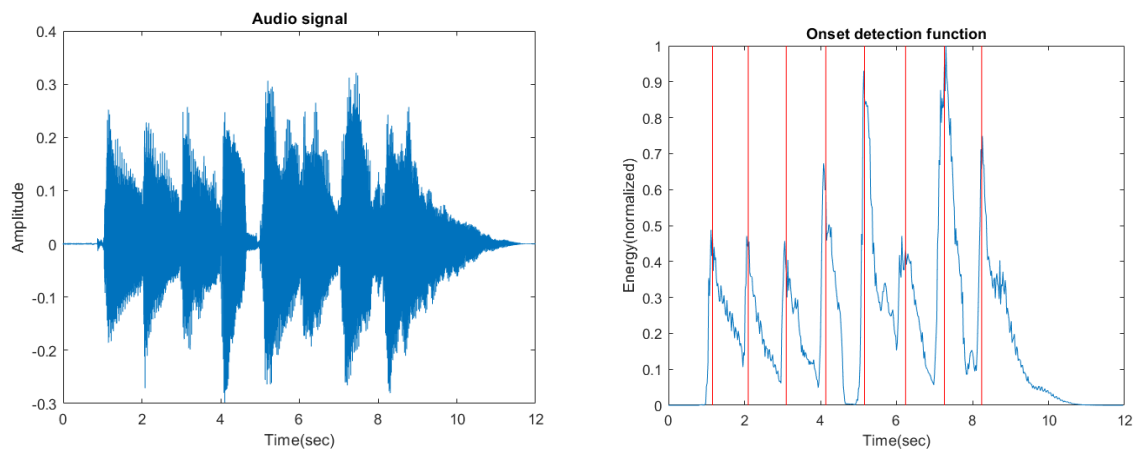


Figure 3.10 Onset detection function in Matlab (right)
and the original audio signal (left)

CHAPTER 4 Artificial neural network

4.1 Network structure

A convolutional neural network (CNN) was used in order to train features of audio signals. It was implemented before in Matlab's *Deep Network Designer Toolbox*, but some layers were modified in order to match better the audio extracted features. This network structure was chosen because it is a powerful classification network, pretrained with a large database of images and also due to limited memory and weak CPU(Central Processing Unit) performance. The process of training a neural network itself requires a lot of computing power, a large memory and based on variations of these two the process can take time from a few minutes to hours. Thus, the choice for a pre-trained network. It is described in [34] the concept of transfer learning: taking a pretrained network and tune it for learning new tasks. Transfer learning is the method implemented in this project, because it is expected to give better accuracy when using a smaller database and the learning process will be faster, as it passed once through the training procedure.

AlexNet Network was originally designed for image recognition and classification. It has been trained on a large database and it is able to classify images based on 1000 object categories. [35] For chord recognition, the following method was adopted: as input data to our CNN was chosen a database of chromagrams corresponding to different chords. These chromagrams were previously obtained by processing a large database of audio samples corresponding to 10 most frequent guitar chords, publicly available in "*Neural Networks for Musical Chords Recognition*" [36]. The chromagrams are processed through the layers and then at the end classified by their label(chord).

To store the chromagrams for training, the *imageDatastore* function was used. It takes as input the main folder of the chromagrams that are classified by chord-name labels in subfolders. It is a suitable function in this system, because it creates a collection of files in one variable, with each file labelled by the folder name. This preparation of input data is crucial for the training, as the final layers require these labels for classification. The function call is as follows:

```
imdsTrain = imageDatastore(folder_path, 'IncludeSubfolders', ...
                           true, 'LabelSource', 'foldernames');
```

The *IncludeSubfolders* and *LabelSource* arguments must be set accordingly for the arrangement described below.

After the input data is stored, the layer structure and layer parameters should be defined. The last three layers of the *AlexNet Network* were modified, according to the chord classes. The lines of code below describe this process, where 'AlexNet' is the imported network from Matlab: [37]

```
layersTransfer = AlexNet.Layers(1:end-3);
numClasses = numel(categories(imdsTrain.Labels));
layers = [
    layersTransfer
    fullyConnectedLayer(numClasses, 'WeightLearnRateFactor', ...
                        20, 'BiasLearnRateFactor', 20)
    softmaxLayer
    classificationLayer ];
```

The layers of the network are presented in a sequence in Fig. 4.1. There are 24 layers in this network, each of them with different properties, that contribute to the process of training.

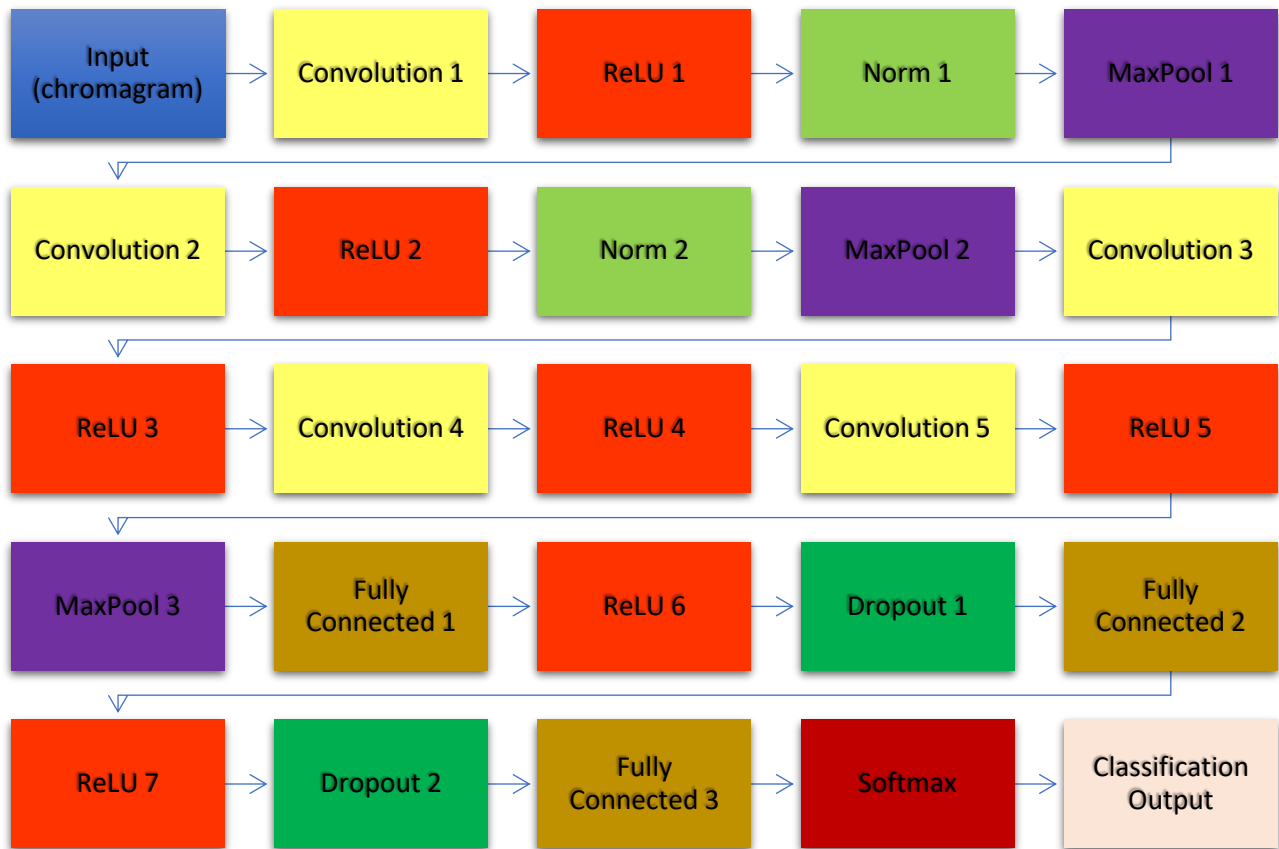


Figure 4.1 Structure of *AlexNet Network* (modified)

The input layer (*imageInputLayer*) stores the initial data for the network, taken from the datastore built in the workspace. This data is loaded and then split in two categories (using the *Deep Network Designer Tool*): 80% of the data is used for training and 20% for validation. The size of the images is 227x227x3. Therefore, the chromagrams should be redimensioned before passing them as input to the network. Fortunately, the images of the chromagrams are resized automatically by the tool. The only step to be done before is the construction of the image datastore in Matlab with labels that will help to classify the output. The distribution of the data can be visualised in the tool before the training (Fig. 4.2).

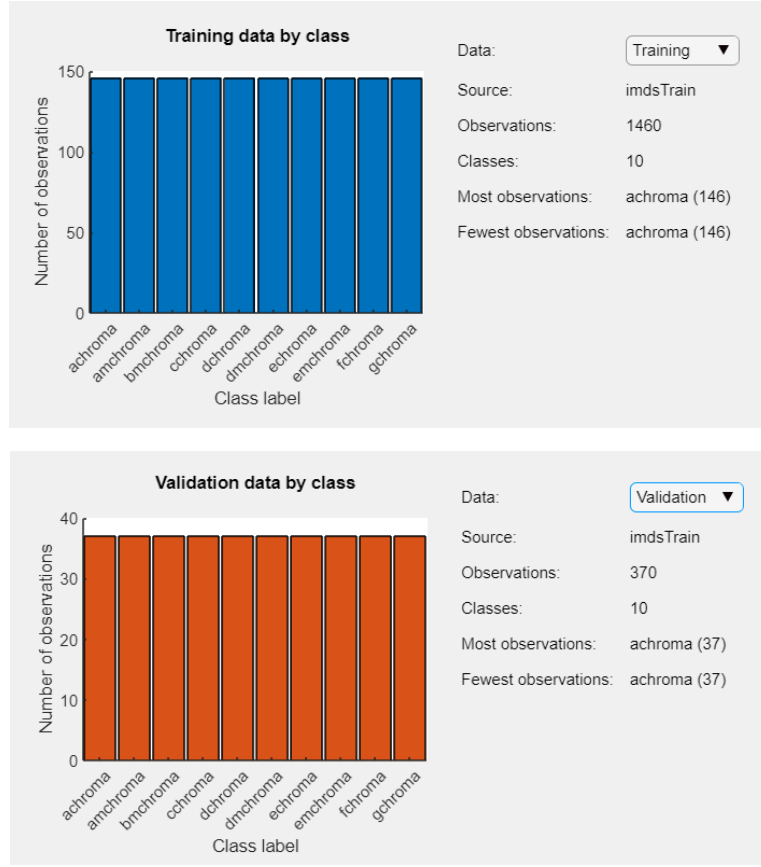


Figure 4.2 Training and validation data distribution

The convolutional layers (*convolution2dLayer*, *groupedConvolution2dLayer*) apply sliding convolution filters on the input image, both on vertical and horizontal dimensions. The grouped convolution layer does the same, but channel-wise. They also compute the product of the input with the weights and add the bias. The parameters of such a layer are the size of the filter (which in this project is 3-dimensional), the number of such filters to be applied, the stride and padding. The stride is the step for passing through the input, with two values for horizontal and vertical dimensions. Padding refers to the extra values to be added as rows and column to the input size so that the output matches the input size when it is the case.

Thresholding is achieved using a layer connected to the output of the computational layer, namely the rectified linear unit layer (*reluLayer*). It sets to zero any negative input and the value itself for 0 or positive values of the input.

A cross-channel normalization local response is created by the normalization layers (*crossChannelNormalizationLayer*) channel-wise that are situated after the ReLU layer. The process is done by replacing the each element from the input with a normalized value based on a number of neighbouring channels. The size of the normalization window is set through the function arguments. This replacement is represented by the equation:

$$x' = \frac{x}{\left(K + \frac{\alpha * ss}{windowChannelSize}\right)^\beta}, \quad (4.1)$$

, where K , α and β represent the hyperparameters in the normalization, ss - the sum of the squared elements from the window. The hyperparameters can also be adjusted when building such a layer. [38]

Down-sampling is achieved through the maxPool layer (*maxPooling2dLayer*) by dividing the input in rectangular areas, denoted by the *poolSize* argument, and computing the maximum from each

of these areas. It follows a convolutional layer, has similar parameters as it (stride, padding) and it has a major role is reducing the complexity of the network, by reducing the number of connections to the next layer, thus the number parameters that should be learned.

A fully-connected layer (*fullyConnectedLayer*) has the basic structure of a neural network: takes the inputs, multiplies them with individual weights adds the bias and forwards the outputs to the next layer.

At the end of the network, the *dropoutLayer* sets in a random way input parameters to zero, given as argument a probability of dropping out inputs. Like the maxPool layer, it helps the network from overfitting. [39]

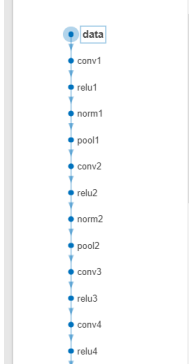
A softmax layer (*softmaxLayer*) applies the softmax function (described in detail in Chapter 1.4 – Convolutional Neural Networks) to the input. It is followed by the last layer, the classification one (*classificationLayer*). This is the fundamental structure (fully-connected - softmax - classification) of the ending network when having a classification problem. The final layer computes the cross-entropy loss of the classes that are used for the training.

The layers along with their parameters and definitions can be found at [40] . The structure was presented in summary in this chapter.

Deep Learning Network Analyzer tool has two important contributions: first of all, it checks for possible errors in the network structure and it highlights warnings, and secondly it shows a table with a detailed description of the layers and the evolution of the parameters, as in Fig 4.3.

Network from Deep Network Designer
Analysis date: 23-Jun-2020 11:22:29

25 layers 0 warnings 0 errors



	Name	Type	Activations	Learnables	Total Learnables
1	data 227x227x3 images with 'zerocenter' normalization	Image Input	227x227x3	-	0
2	conv1 96 11x11x3 convolutions with stride [4 4] and padding [0 0 0 0]	Convolution	55x55x96	Weights 11x11x3x96 Bias 1x1x96	34944
3	relu1 ReLU	ReLU	55x55x96	-	0
4	norm1 cross channel normalization with 5 channels per element	Cross Channel Normalization	55x55x96	-	0
5	pool1 3x3 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	27x27x96	-	0
6	conv2 2 groups of 128 5x5x48 convolutions with stride [1 1] and padding [2 2 2 2]	Grouped Convolution	27x27x256	Weights 5x5x48x128... Bias 1x1x128x2	307456
7	relu2 ReLU	ReLU	27x27x256	-	0
8	norm2 cross channel normalization with 5 channels per element	Cross Channel Normalization	27x27x256	-	0
9	pool2 3x3 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	13x13x256	-	0
10	conv3 384 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	13x13x384	Weights 3x3x256x384 Bias 1x1x384	885120
11	relu3 ReLU	ReLU	13x13x384	-	0

Figure 4.3 Analysed network structure

The full Matlab code for the architecture and the parameters initialization can be found in Annex1(j).

4.2 Database of chords

4.2.1 Training dataset

A significant part in training a neural network is to have a large database of inputs because a powerful network model of chord recognition requires many chord samples in order to have the best possible accuracy. For this model an available database was chosen from the internet, as mentioned at the beginning of the chapter. It gathers audio files recorded in WAV format, sampled at 44100 Hz. It is limited to the most frequent chords: A, Am, Bm, C, D, Dm, E, Em, F, G. This dataset gathers guitar chord samples recorded in two different environments: half of them were recorded in an anechoic chamber and the other half in a noisy environment. It is of great relevance that the noisy environment is taken into account, because not all recordings are made with professional equipment or in an anechoic chamber.

The training dataset gathers guitar chord samples, recorded using different types of guitars. For each chord 200 samples were recorded, half of them in the anechoic chamber and the other half in a noisy room. Therefore, the efficiency of the set increases as there are 2000 samples to be further trained. [36]

As the implemented neural network structure requires images as input data, on the samples from the training dataset the chromagram function was applied [Annex1(e)] to each sample and the resulting chroma features were saved with the “.jpg” format, creating the proper dataset for the network. A Matlab script was developed for this method that is based on files and folders manipulation. [41] [42]

4.2.2 Test dataset

Another dataset of chords was created with personal guitar sample recordings that will be used for testing the performance of the network. For ten audio files of approximately 10 seconds, the chords are already known and the performance of the program will be evaluated on them. The dataset also contains 48 short (less than one minute) audio recordings with ample guitar samples in terms of chord types.

A performing guitar artist helped with the composed guitar pieces. The recordings were made using a Fender American Performer Series Stratocaster HSS equipped with 2 Yosemite Stratocaster single coils (middle, neck positions) and 1 Double Tap humbucker (bridge) plugged in through a standard 2 x 6.3 mm jack cable to a Focusrite Scarlett 2i2 2nd Generation audio interface.

Ableton Live Lite was used as the digital audio workstation. Most of the audio files were recorded on the guitar’s humbucker pickup, but other pickup combinations were used as well. The tone and both volume knobs were set to 10. Thus, the guitar’s raw signal was captured with no additional effects.

4.3 Neural Network training

In order for the training process to take place, some training options must be defined in the *Deep Network Designer* tool. The layer weights were preserved from the existing trained structure. The initial learning rate was set to 0.0001, to have slow learning in the old layers and a fast learning in the new ones. This happens because in the modified fully-connected layer the weight and bias learning factor were increased. An epoch is a full training cycle on the training data. In this case, the number of epochs was limited to six, because the old layers were pretrained. [37] The code where the training options are set can be found in Annex1(j).

After this setup, the *trainNetwork* function is called with the training datastore, the layers structure and the training options as arguments. The training results are shown in Fig. 4.4, with a validation accuracy of 95.95% and a training time of 59 minutes.

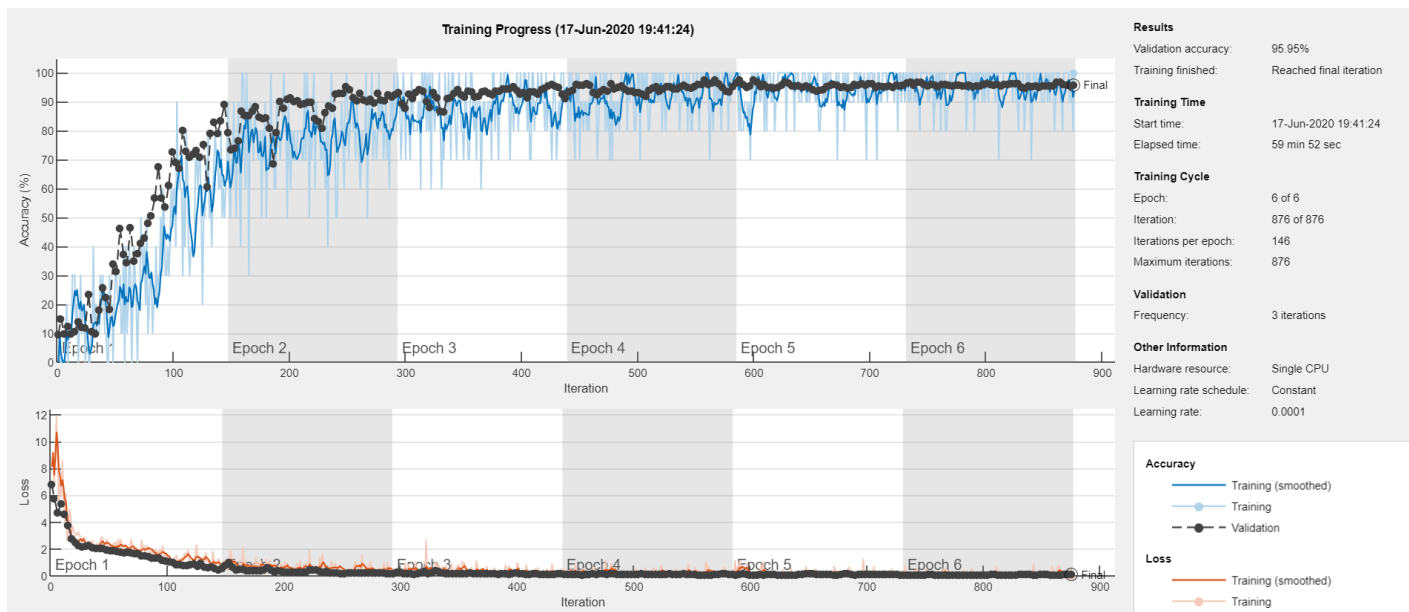


Figure 4.4 Training results

In Annex1(k) the trained network was loaded for seizing additional information related to the process of training and to the network itself. Therefore, using the “*whos('net')*” command, the network size was determined: *217.1428 MB* . The confusion matrix for test data (composed of additional guitar chord samples) and validation data are illustrated in Fig. 4.5 and Fig. 4.6 respectively.

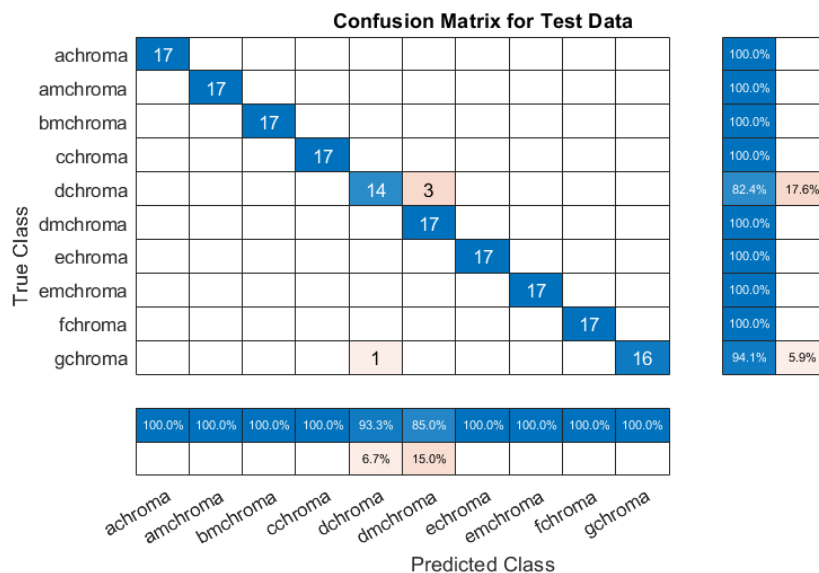


Figure 4.5 Confusion matrix for Test Data

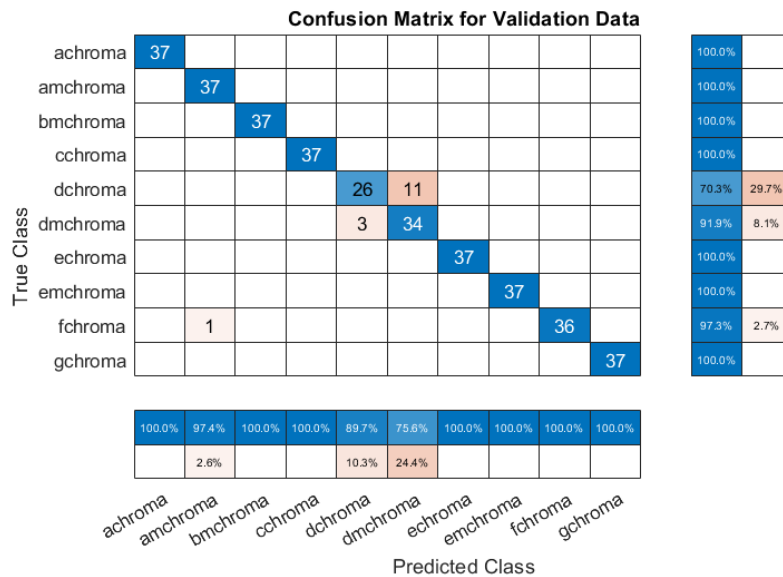


Figure 4.6 Confusion matrix for Validation Data

The confusion matrix describes the performance of the classification network, by telling which of the chords were identified as being from the true class and which were recognized wrong. In Fig. 4.5, a number of three *D* chords were recognized as *Dminor* and one *G* chord as *D*.

Finally, the overall error on the training and validation data through the classification network is:

- Training error: 2.6712%
- Validation error: 4.0541%

CHAPTER 5 Experimental Results

The main test script samples an input audio file with a sampling frequency of 44.1 kHz and stores the values in a vector. A flowchart of the test script is presented in Fig. 5.1. Some parameters can be modified at the beginning of the script. The minimum frequency from which the user wants to consider the start value for the range of evaluated frequencies can be changed by taking into consideration the instruments on which the song was recorded and whether the song has high and low pitches. If not, the frequency range can be minimized. This also applies to the number of octaves that can be changed by own preferences.

Another parameter that can be adjusted is the number of bins in one octave. Classically, the octave is divided in 12 bins. But for a better precision, the number of bins was set to 24. More information about this subject is in Chapter 3.

If the input audio signal is recorded on two channels, only the right one is chosen because only one channel is acceptable for the following steps in the program. As the neural network was trained before, it should only be loaded into the workspace. The actual design and process of training can be found in Annex1(j), where likewise the parameters can be changed.

Onset detection is performed and the peaks location is stored in a vector. A hop size is set for scanning the signal. The next step is to iterate on the chords (the signal from one peak to another) and to analyse each of them individually as follows:

- chroma feature computation
- filter the signal with an average filter or geometric-mean filter
- get a caption of the chromagram figure
- resize it to the dimensions of the input datat in the neural network ([227 227 3])
- classify the chromagram based on the network labels
- display the chromagram with the label title

The final step is to display the predicted chords from the audio file.

The full script in Matlab for this test can be found in Annex1(l).

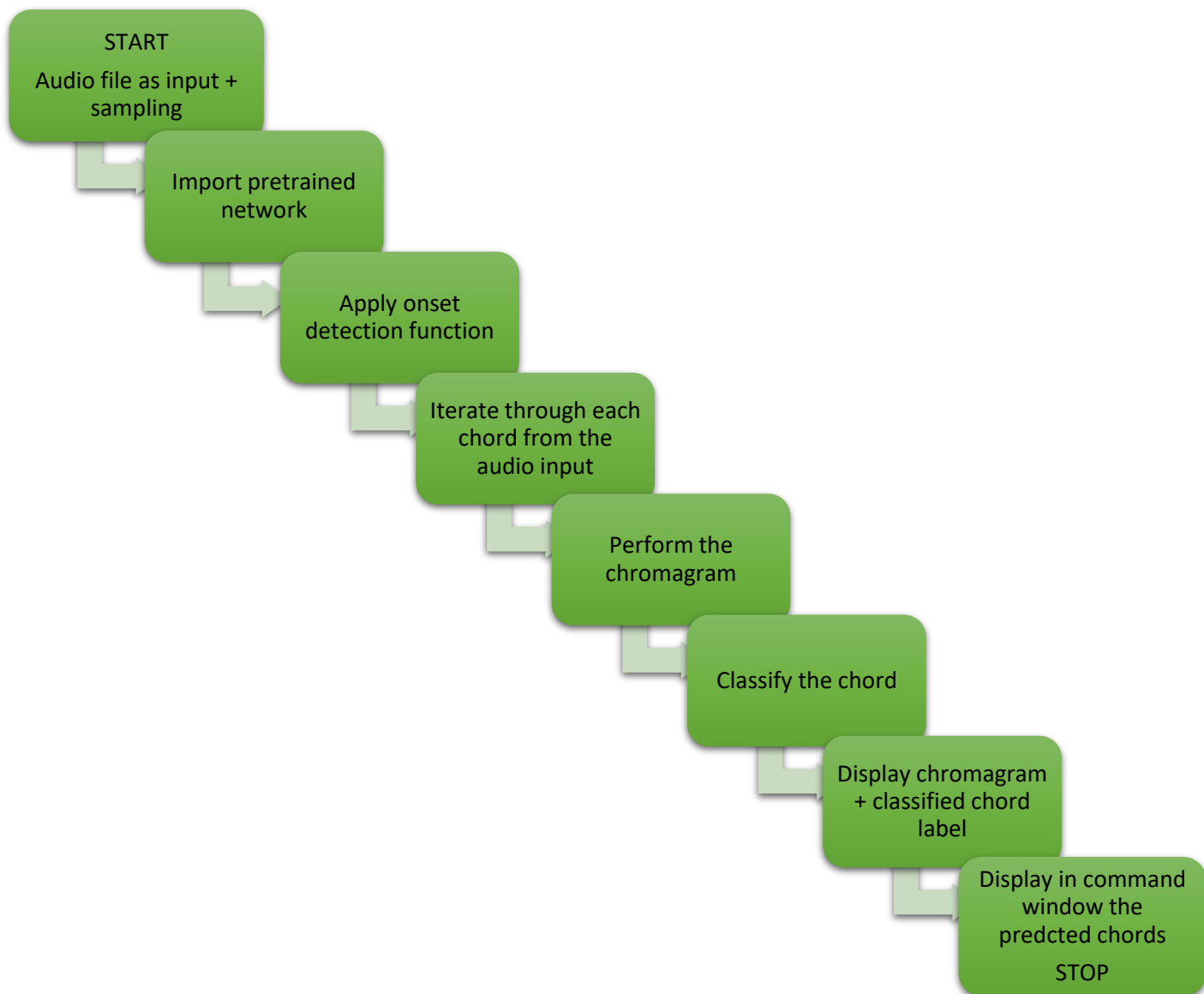


Figure 5.1 Flowchart of the test script

5.1 Experiment 1

The first experiment was made on a small part of the test dataset using the test script. First of all, the program was tested for 10 short guitar samples (from personal recordings) for which the played chord sequence was labelled. The predicted chords along with the accuracy are presented in Table 5.1. It was obtained an overall accuracy of 79.8% on the 10 test input data.

Name	Real chords	Predicted chords	Accuracy[%](*)
1.wav	Am – Dm – G – C – Em - Am	AM - DM - G - C - EM - AM	100
2.wav	B – D – B – E - F# - F#	BM - DM - BM - EM - F - F	50
3.wav	Em – A – D – G – C - F	EM - A - D - G - C - F	100
4.wav	G - C - Em - D	G - C - EM - D	100
5.wav	Dm - C - Bb - A - A7	DM - C - DM - A - A	80
6.wav	D – Bm - F#m - A	D - BM - F - A	75
7.wav	C - Am - Em - F	C - AM - EM - F	100
8.wav	A - Adim – Em - Dmaj7 - Cmaj7	A - F - EM - D - EM	60
9.wav	Bm – A – G – Em - Bm	BM - A - G - EM - BM	100
10.wav	C#m - Bm - E - A6 - B - G#	A - BM - E - F - E - A	33

Table 5.1 Experimental results

*taking into account only major and minor classification

5.2 Experiment 2

Another implementation of the project is to find the the most frequent chord from an audio file, that can be further optimized in making a decision on the tonality of a song. Another part of the recorded guitar files has been revealed and processed with an additional function at the end that can sort the identified chords and state which chord was played the most in the entire song.

The dataset is formed of 16 audio files with a duration of under one minute, among which some melodic lines from known songs can be found. A table was created to represent the results, that in *Matlab* are displayed as [Annex1(l)]:

*The chords from this audio file are: G D F EM G F EM EM EM F
G F F EM G D DM D C F F EM
The most frequent chord from this song is: F*

Audio file	Most frequent chord
rec1.wav	F
rec2.wav	F
rec3.wav	G
rec4.wav	Em
rec5.wav	G
rec6.wav	G
rec7.wav	F
rec8.wav	Dm
rec9.wav	Bm
rec10.wav	D
rec11.wav	A
rec12.wav	D
rec13.wav	D
rec14.wav	A
rec15.wav	D
rec16.wav	D

Table 5.2 Results of finding the main chord from an audio file

5.3 Experiment 3

The implementations until now limit the program only to guitar chord recognition. To see if the neural network can classify piano and violin chords, a smaller public database [36] was accessed with ten audio samples from ten chords played at the piano\violin.

An chromagram was computed for each of the audio file for a chord and stored in a separate labelled folder. The trained neural network was loaded and the chromagrams were classified by the network with the results given by the confusion matrix in Fig. 5.2 for the piano chords and Fig. 5.3 for the violin chords.

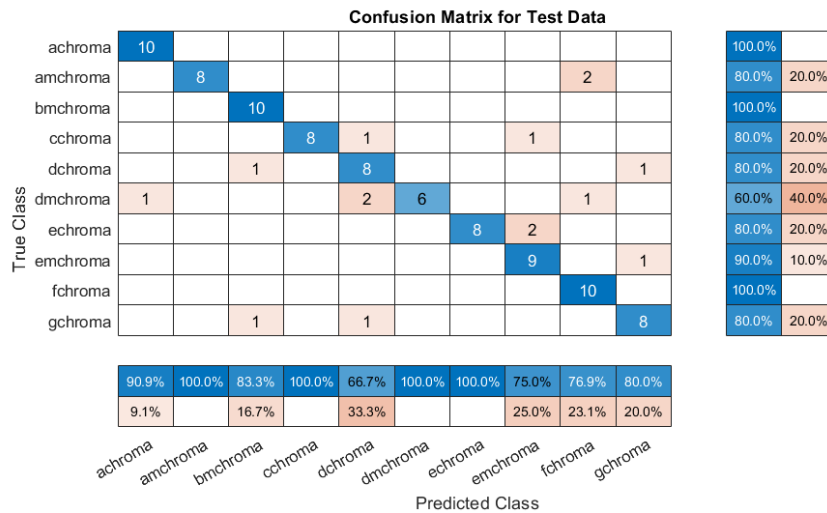


Figure 5.2 Confusion matrix for piano chords

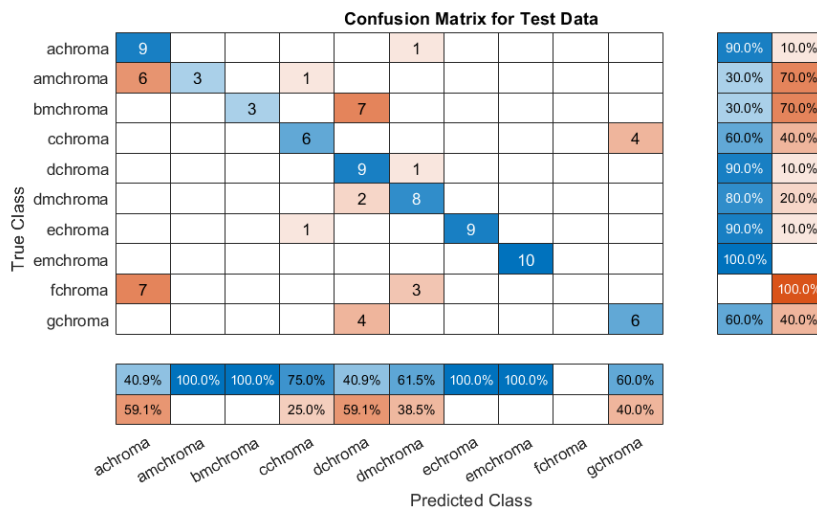


Figure 5.3 Confusion matrix for violin chords

When using guitar chords as test data (Fig. 4.5, Chapter 4) the classification gave almost 100% overall accurate results. That is because the network was trained only with guitar chords. With the piano set, the classification provided good results, whilst the confusion matrix for the violin set gave poorly results for some chords. This variations in accuracy occurs due to one particular characteristic of the sound played from the instruments: the timbre. The harmonic structure of a piano is reasonably similar to one of a guitar and this can be visualized in Fig. 3.5, Chapter 3. On the other

hand, a violin has a much more distinct harmonic structure than the other two instruments, hence the poor classification of the chords.

CONCLUSIONS

The implemented project required a considerable research in many domains: machine learning with its subfield of deep learning and neural networks, sound representation, signal processing and programming techniques in *Matlab*. The target was to extract features from an audio signal and based on them, after a training process of a neural network, to develop a script that can make use of the network and recognize the chords from recorded songs.

The first and second chapter comprise basic theoretical notions that are needed in order to develop the complex software that this project is based on.

The first step, in Chapter 3, was to establish a number of functions in Matlab that all together compose a chord recognition system. The Constant-Q Transformation provided satisfactory results and the option of choosing the range of frequencies for an analysed signal makes it an adjustable function. When extracting information of frequencies in time, CQT gives excellent results. Some other extraction methods exist and can be implemented as a comparison.

The following step was to represent the chromagram. The implementation of the function was done accordingly and two types of filters were applied to the feature, but the results were not as positive as expected. For optimization of the system, more performant filtering techniques can be adopted.

Further, an onset detection function based on energy computation helped in detecting the exact parts of an audio file where a note is played, hence the pitches. The energy-based method is the most accesible one, but there are other methods known in literature that can be tested and evaluated. The peaks were identified along with the time value where they are sampled.

An existing neural network from Matlab's Deep Network Designer Toolbox was adopted, primary because it was trained before on a large image database with good accuracy in the end, so it was appropriate for the chroma features from this project. Some adjustments were made to the input and output layers of the network, to be able in the end to classify chords by labels. A public database of chords was identified on the internet and used for the training process. Fortunately, the training lasted only one hour, but the options were set in such a way that the learning is emphasized on the new, untrained layers.

Another accomplishment was the conception of a dataset of various guitar samples, by recording with an audio interface.

The experiments were described in chapter 5 and they gave the expected results. The network was trained with guitar chords so it is predicted to have a better accuracy with guitar chords as test data. The experiments led to a disclosure, that the timbre plays a major role in describing the sound produced by an instrument.

Thus, to have a more optimized system, some adjustments should be made taking the harmonic structures into account, along with an efficient onset detection function for chords that have different sources of production. The training database for the network structure can be expanded to incorporate not only guitar chords, but also piano, violin and chords produced by other instruments. Also, some parameters of the artificial neural network can be remodelled, primarily for analysing the behaviour of distinct network types.

Original contributions

The substantial theoretical research that further played a major role in successfully finalizing the experiments, along with the experimental results establish the contributions of the author. These contributions are:

- Implementation of the Constant-Q Transformation through a function that represents the evolution of frequencies in time from an audio signal.

- Extracting a chroma feature, based on the CQT, that identifies the musical chords from an audio file.
- Research into the onset detection methods known in literature and development of an appropriate function.
- Choosing an appropriate artificial neural network structure and adjusting its parameters to the project requirements.
- Realizing the audio recordings, with the support of a performing guitar artist, through an audio interface and creating a personal database of audio samples.
- Developing of a test script that facilitated the achievements of the experiments and reaching to some conclusions regarding each step of the project, how can it be further improved and what may be some causes of errors.

Future development

This project can be a start point for other complex software-based implementations, that can contribute to the constant-expanding field of audio processing by use of deep learning. They may be related to:

- Developing more efficient methods of chromagram computation and pitch detection that take into account more complex songs of long duration. This a first essential step, because these developments represent the main support for an accurate chord recognition system.
- Adjustment the system for detecting chords from other types of instruments.
- Applying a fine tuning on the neural network architecture or building a higher-performace network in detail.
- Expanding the system to chord progression recognition using deep learning methods.

Bibliography

- [1] S. Amidi, "Deep Learning," [Online]. Available: <https://stanford.edu/~shervine/teaching/cs-230/>. [Accessed 14 June 2020].
- [2] J. C. Brown, "Calculation of a constant Q spectral transform," *Media Laboratory, Massachusetts Institute of Technolo*, 1990.
- [3] "Notes on Artificial Intelligence, Machine Learning and Deep Learning for curious people," [Online]. Available: <https://towardsdatascience.com/notes-on-artificial-intelligence-ai-machine-learning-ml-and-deep-learning-dl-for-56e51a2071c2>. [Accessed 10 06 2020].
- [4] "A Quick Introduction to Neural Networks," [Online]. Available: <https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/>. [Accessed 10 June 2020].
- [5] "How to Implement a Neural Network with only +1 and -1 ?," [Online]. Available: <https://mc.ai/how-to-implement-a-neural-network-with-only-1-and-1/>. [Accessed 10 June 2020].
- [6] J. Roell, "From Fiction to Reality: A Beginner's Guide to Artificial Neural Networks," [Online]. Available: <https://towardsdatascience.com/from-fiction-to-reality-a-beginners-guide-to-artificial-neural-networks-d0411777571b>. [Accessed 10 06 2020].
- [7] "Weight (Artificial Neural Network)," [Online]. Available: <https://deeptai.org/machine-learning-glossary-and-terms/weight-artificial-neural-network>. [Accessed 10 06 2020].
- [8] "Single-Layer Perceptrons," [Online]. Available: <http://homepages.gold.ac.uk/nikolaev/311perc.htm>. [Accessed 12 06 2020].
- [9] "On-line learning in neural networks with ReLU activations," [Online]. Available: <https://michielstraat.com/talk/mastertalk/>. [Accessed 12 06 2020].
- [10] J. Y. Q. Z. G. Y. Ren, "Multi-Feature Fusion with Convolutional Neural Network for Ship Classification in Optical Images," *MDPI*, 9 October 2019.
- [11] S. W. Smith, "Chapter 22: Audio Processing," in *The Scientist and Engineer's Guide to Digital Signal Processing*, California Technical Publishing, 1997.
- [12] C. Negrescu, "Audio Engineering," *Course notes, "Politehnica" University of Bucharest*, 2018.
- [13] "HUSHCORE™ Acoustical Products & Systems, Technical Discussion," [Online]. Available: <https://hushcore.net/>. [Accessed 10 06 2020].
- [14] "Intervals," [Online]. Available: <https://www.musictheory.net/lessons/31>. [Accessed 18 June 2020].
- [15] "Tones and Semitones," [Online]. Available: <https://www.mymusictheory.com/learn-music-theory/for-students/grade-1/grade-1-course/111-10-tones-a-semitones>. [Accessed 14 June 2020].
- [16] "Chords and chord inversions," [Online]. Available: <https://www.earmaster.com/wiki/music-memos/what-are-chords-in-music.html>. [Accessed 18 June 2020].
- [17] "Minor Chords," [Online]. Available: <http://www.choose-piano-lessons.com/minor-chords.html>. [Accessed 18 June 2020].
- [18] "Classification of Musical Instruments: the Sachs-Hornbostel System," [Online]. Available: <https://www.liveabout.com/classification-of-musical-instruments-2456710>. [Accessed 20 June 2020].
- [19] Y. Fujiso, "Vibroacoustical Study of a Solid-Body," CHALMERS UNIVERSITY OF TECHNOLOGY, Göteborg, Sweden , 2009.
- [20] "The PARTs of an Electric Guitar," [Online]. Available: <https://www.guitar-skill-builder.com/parts-of-an-electric-guitar.html>. [Accessed 16 06 2020].
- [21] "Guitar String Notes," [Online]. Available: <http://griffhamlinbluesguitarunleashed.com/tag/what-notes-are-the-guitar-strings/>. [Accessed 21 June 2020].

- [22] J. P. B. Taemin Cho, "On the relative importance of individual components of chord recognition systems," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2014.
- [23] R. Ramaswamy, "Audio pitch-shifting & the Constant-Q transform," [Online]. Available: <https://www.edn.com/audio-pitch-shifting-the-constant-q-transform/>. [Accessed 16 June 2020].
- [24] "What is Quarter Tone Music?," [Online]. Available: <https://livingpianos.com/what-is-quarter-tone-music/>. [Accessed 16 June 2020].
- [25] "Signal Processing," [Online]. Available: <https://dsp.stackexchange.com/questions/40598/why-would-one-use-a-hann-or-bartlett-window>. [Accessed 18 06 2020].
- [26] W. A. M. H. Fletcher, "Loudness, Its Definition, Measurement and Calculation," *Bell Telephone Laboratories*, 1933.
- [27] "Sound Power," [Online]. Available: https://www.engineeringtoolbox.com/sound-power-level-d_58.html. [Accessed 19 June 2020].
- [28] M. M. R. S.-R. a. T. D. B. Yizhao Ni, "An end-to-end machine learning system for harmonic analysis of music. Audio, Speech, and Language Processing," *IEEE Transactions*, 2012.
- [29] J. Chapman, "Harmonics in Sound," 09 January 2019. [Online]. Available: <https://bayanaudio.com/blogs/article/harmonics-in-sound>. [Accessed 12 06 2020].
- [30] R. J. W. a. J. P. B. Termin Cho, "Exploring Common Variations in State of the Art Chord Recognition Systems," *Music and Audio Research Laboratory (MARL)*, 2010.
- [31] T. H. Park, in *Introduction to Digital Signal Processing: Computer Musically Speaking*, pp. 364-366.
- [32] A. Bonvini, "Automatic chord recognition using Deep Learning techniques," Master Graduation Thesis, Politecnico di Milano.
- [33] L. D. S. A. C. D. M. D. M. B. S. J. P. Bello, "A Tutorial on Onset Detection in Music Signals," *IEEE Transactions*, no. Speech and Audio Processing, 2005.
- [34] "Transfer Learning Using AlexNet," [Online]. Available: <https://www.mathworks.com/help/deeplearning/ug/transfer-learning-using-alexnet.html>. [Accessed 20 June 2020].
- [35] Matlab, "Deep Learning Toolbox Model for AlexNet Network," [Online]. Available: <https://www.mathworks.com/matlabcentral/fileexchange/59133-deep-learning-toolbox-model-for-alexnet-network>. [Accessed 20 06 2020].
- [36] J.-J. E. S. P. M. V. D. J. Osmalskyj, "Neural Networks for Musical Chords Recognition," *Journées d'informatique musicale*, pp. 39-46, 2012.
- [37] "DeepLearning AlexNet," [Online]. Available: <https://www.mathworks.com/help/deeplearning/ref/alexnet.html>. [Accessed 22 June 2020].
- [38] "MATHWorks- Batch Normalization Layer," [Online]. Available: <https://www.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.batchnormalizationlayer.html>. [Accessed 18 June 2020].
- [39] "MATHWorks- DropoutLayer," [Online]. Available: <https://www.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.dropoutlayer.html>. [Accessed 20 June 2020].
- [40] "Deep Learning Layers," [Online]. Available: <https://www.mathworks.com/help/deeplearning/ug/list-of-deep-learning-layers.html>. [Accessed 20 June 2020].
- [41] "MathWorks Import Export Sequence of Files," [Online]. Available: https://www.mathworks.com/help/matlab/import_export/process-a-sequence-of-files.html. [Accessed 15 June 2020].

- [42] “Mathworks Read files from a folder,” [Online]. Available: <https://www.mathworks.com/matlabcentral/answers/342202-read-all-files-from-a-folder-edit-them-and-save>. [Accessed 15 June 2020].
- [43] “MATHWORKS-Layers of CNN,” [Online]. Available: <https://www.mathworks.com/help/deeplearning/ug/layers-of-a-convolutional-neural-network.html>. [Accessed 18 June 2020].

Annexes

Annex 1

(a)

```
%Hamming window function
function y = hamm(x)
%y = output of windowed input x
    N = length(x);
    alpha = 25/46;
    W = [alpha + (1-alpha)*cos(2*pi*(0:N-1)/N)]';
    y = x.*W;
end
```

(b)

```
%CQT function
function [y, f, N] = cqt(x,fs, fmin, BinsPerOctave, NoOctave)
% fs - sampling frequency
% fmin - minimum frequency
% BinsPerOctave - nr. of values in an octave
% NoOctave - nr. of octaves
% N - size of each window
% y - output with N rows
% f - frequencies corresponding to each bin
    if size(x,1)== 1 % turn input in column
        x = x';
    end
    k = 0:NoOctave*BinsPerOctave;
    Q = floor(1/(2^(1/BinsPerOctave)-1));
    f = 2.^(k/BinsPerOctave)*fmin; % range of frequencies
    N = round(fs*Q./f); % sizes of windows
    % complete with zeros at the end if the size of x
    % is too small
    if length(x)<max(N)
        x = [x; zeros(max(N)-length(x),1)];
    end
    y = zeros(length(k),1);
    for i=1:length(k)
        exponent = exp(-1i*2*pi*Q*(0:N(i)-1)/N(i));
        y(i) = 1/N(i)* exponent * hamm(x(1:N(i)));
    end
end

end
```

(c)

```
%CQT spectrogram
function [y, t, f] = cqt_spectrogram(x,fs, fmin, BinsPerOctave, NoOctave, step)
% fs - sampling frequency
% fmin - minimum frequency
% BinsPerOctave - nr. of values in an octave
% NoOctave - ne. of octaves
% step - step size between constant-Q transforms
% N = BinsPerOctave*NoOctave
% y - output with N rows and M columns,
```

```

% depending on the evolution of the signal in time
if size(x,1)== 1 % turn input in column
    x = x';
end
% nr. of processes of CQT
index = 1:step:length(x)-step;
y = zeros(NoOctave*BinsPerOctave+1,length(index));
for i = 1:length(index)
    [y(:,i), f, ~] = cqt(x(index(i):end),fs, fmin,...
        BinsPerOctave, NoOctave);
end
%time vector
t = (index-1)/fs;

```

end

(d)

```

%A-weight function
function y_w = a_weight(f,y)
    y_w = zeros(size(y));
    Rp = 10^(-12);%reference power
    for i=1:length(f)
        r = (12200^2*f(i)^4)/((f(i)^2+20.6^2)*sqrt((f(i)^2+107.7^2)+...
            (f(i)^2+739.9^2)*(f(i)^2+12200^2)));
        A = 2 + 20*log(r);
        y_w(i,:) = 10*log(y(i,:)) - 10*log(Rp) + A;
    end
end

```

end

(e)

```

%chromagram computation
function [y, t, f] = chromagram(x,fs, fmin, BinsPerOctave, NoOctave, step)
    [x_cqt, t, f] = cqt_spectrogram(x,fs, fmin, BinsPerOctave, NoOctave, step);

    y = zeros(12,length(t));
    BinsPerNote = BinsPerOctave/12;
    for i = 1:length(t)
        aux = zeros(BinsPerOctave,1);
        for j=1:BinsPerOctave
            aux(j) = sum(abs(x_cqt(j:BinsPerOctave:end,i)));
        end
        % choose max of 2 consecutive bins
        for j=1:12
            y(j,i) = max(aux( (j-1)*BinsPerNote+1 : j*BinsPerNote ));
        end
    end
end

```

end

(f)

```

%moving average filter
function Y = avg_filt(N,in)
% N = number of points for averaging
Y = [];
size_in = size(in);
for j= 1:size_in(1)
    filt = [];
    for i=1:size_in(2)-(N-1)

```

```

        s = 1/N*sum(in(j,i:i+N-1));
        filt = [filt s];
    end
    Y = [Y;filt];
end
end

```

(g)

```

%geometric-mean
function Y = geometric_mean(N,in)
% N = number of points for averaging
Y = [];
size_in = size(in);
for j= 1:size_in(1)
    filt = [];
    for i=(N-1)/2+1:size_in(2)-(N-1)/2
        p = (prod(in(j,i-(N-1)/2:i+(N-1)/2))) ^ (1/N);
        filt = [filt p];
    end
    Y = [Y;filt];
end
end

```

(h)

```

%Onset detection function
function [e,locs,t_e] = odf(x,fs)
%energy of the signal
N = 2000;
[e,t_e] = energy_envelope(x,N,fs);
%normalize
e = e./(max(e));
%filter the energies
e_filtered = medfilt1(e,10);
%normalize
e_filtered = e_filtered/max(e_filtered);
%find peaks
[pks,locs] = findpeaks(e_filtered,'MinPeakProminence',0.15);
locs = [locs locs(length(locs))+50];
%plot
figure;
plot(t_e,e);
title('Onset detection function'),
ylabel('Energy(normalized)'),
xlabel('Time(sec)')
hold on
for i=1:length(locs)-1
    line([(locs(i)-1)*0.02,(locs(i)-1)*0.02],[0,1],'Color','r');
end
end

```

(i)

```

%energy computation
function [e,t_e] = energy_envelope(x,N,fs)
%energy method

```

```

L = length(x);%length of input signal
hop = 0.02*fs;
index = 1:hop:length(x)-hop;
t_e = index/fs;
%use a Hamming window and compute energies
for i=1:(L+1-N)/hop
    xx = hamm(x((i-1)*hop+1:hop*i+N-1));
    e(i) = sum(xx.^2);
end
e = [e zeros(length(index)-length(e),1)'];
end

```

(j)

```

%training code
%import images of chromagrams
imdsTrain = imageDatastore( training_data_folder_path ,...
    'IncludeSubfolders',true,'LabelSource','foldernames');
%no. of labels
T = countEachLabel(imdsTrain);

%resize training images
imdsTrain.ReadSize = numpartitions(imdsTrain);
imdsTrain.ReadFcn = @(x)imresize3(imread(x),[227 227 3]);

%imdsTrain => 1460 chromagrams
%imdsValidation => 370 chromagrams
[imdsTrain, imdsValidation] = splitEachLabel(imdsTrain,0.8);

%layers description
layers = [
    imageInputLayer([227 227 3],"Name","data")
    convolution2dLayer([11 11],96,"Name","conv1",...
        "BiasLearnRateFactor",2,"Stride",[4 4])
    reluLayer("Name","relu1")
    crossChannelNormalizationLayer(5,"Name","norm1","K",1)
    maxPooling2dLayer([3 3],"Name","pool1","Stride",[2 2])
    groupedConvolution2dLayer([5 5],128,2,"Name","conv2",...
        "BiasLearnRateFactor",2,"Padding",[2 2 2 2])
    reluLayer("Name","relu2")
    crossChannelNormalizationLayer(5,"Name","norm2","K",1)
    maxPooling2dLayer([3 3],"Name","pool2","Stride",[2 2])
    convolution2dLayer([3 3],384,"Name","conv3",...
        "BiasLearnRateFactor",2,"Padding",[1 1 1 1])
    reluLayer("Name","relu3")
    groupedConvolution2dLayer([3 3],192,2,"Name","conv4",...
        "BiasLearnRateFactor",2,"Padding",[1 1 1 1])
    reluLayer("Name","relu4")
    groupedConvolution2dLayer([3 3],128,2,"Name","conv5",...
        "BiasLearnRateFactor",2,"Padding",[1 1 1 1])
    reluLayer("Name","relu5")
    maxPooling2dLayer([3 3],"Name","pool5","Stride",[2 2])
    fullyConnectedLayer(4096,"Name","fc6","BiasLearnRateFactor",2)
    reluLayer("Name","relu6")
    dropoutLayer(0.5,"Name","drop6")
    fullyConnectedLayer(4096,"Name","fc7","BiasLearnRateFactor",2)

```

```

reluLayer("Name","relu7")
dropoutLayer(0.5,"Name","drop7")
fullyConnectedLayer(10,"Name","fc","BiasLearnRateFactor",20,...
"WeightLearnRateFactor",20)
softmaxLayer("Name","softmax")
classificationLayer("Name","classoutput"]];

%plot layers
figure,plot(layerGraph(layers));

%training options
opts = trainingOptions("sgdm",...
    "ExecutionEnvironment","auto",...
    "InitialLearnRate",0.0001,...
    "MaxEpochs",6,...
    "MiniBatchSize",10,...
    "Shuffle","every-epoch",...
    "ValidationFrequency",3,...
    "Plots","training-progress",...
    "ValidationData",imdsValidation);

%TRAIN NETWORK
[net, traininfo] = trainNetwork(imdsTrain, layers, opts);

(k)
%load training setup data
clear all;clc;
trainingSetup = load("Data_Setup_Training_Setup.mat");

%load the layers
load("layers_1.mat");

%plot layers
figure,plot(layerGraph(layers_1));

%import data
imdsTrain = trainingSetup.imdsTrain;

%resize data to match the network input layer
imdsTrain.ReadSize = numpartitions(imdsTrain);
imdsTrain.ReadFcn = @(x)imresize3(imread(x),[227 227 3]);

%imdsTrain => 1460 chromagrams
%imdsValidation => 370 chromagrams
[imdsTrain, imdsValidation] = splitEachLabel(imdsTrain,0.8);
load("trainedNetwork_1.mat");
net = trainedNetwork_1;

%network size
info = whos('net');
disp("Network size: " + info.bytes/(2^20) + " MB")

%%create database for test chromagrams
imdsTest = imageDatastore(test_data_path ,...
    'IncludeSubfolders',true,'LabelSource','foldernames');

```

```

T_test = countEachLabel(imdsTest);

%resize test images
imdsTest.ReadSize = numpartitions(imdsTest);
imdsTest.ReadFcn = @(x)imresize3(imread(x),[227 227 3]);

%confusion matrix for test data
preds_test= classify(net, imdsTest);
figure('Units','normalized','Position',[0.2 0.2 0.5 0.5]);
cm = confusionchart(imdsTest.Labels,preds_test,...
    'Title','Confusion Matrix for Test Data',...
    'RowSummary','row-normalized','ColumnSummary','column-normalized');

%classify validation data using the pre-trained network
preds_validation= classify(net, imdsValidation);
figure('Units','normalized','Position',[0.2 0.2 0.5 0.5]);
cm = confusionchart(imdsValidation.Labels,preds_validation,...
    'Title','Confusion Matrix for Validation Data',...
    'RowSummary','row-normalized','ColumnSummary','column-normalized');

%training and validation error
validationError = mean(preds_validation ~= imdsValidation.Labels);

preds_train = classify(net,imdsTrain);
trainError = mean(preds_train ~= imdsTrain.Labels);

disp("Training error: " + trainError*100 + "%")
disp("Validation error: " + validationError*100 + "%")

%test some data from the test datastore
for i=50:58
    I = imread(imdsTest.Files{i});
    I = imresize3(I,[227 227 3]);
    [YPredTest,scoresTest] = classify(net,I);
    figure
    imshow(I)
    label = YPredTest;
    title(string(label) + ", " + num2str(100*max(scoresTest),3) + "%");
end

(1)
close all, clear, clc;

fmin = 32.7;%minimum frequency analysed
bpo = 24;
noo = 8;

audio_file = audio_file_path;
[x,fs] = audioread(audio_file);

%choose one channel
if size(x,2) == 2
    x = x(:,2);

```

```

end

t=0:1/fs:length(x)/fs-1/fs;
%plot audio signal
figure,
plot(t,x),xlabel('Time(sec)'),ylabel('Amplitude'),
title('Audio signal')

%import the trained network
load("trainedNetwork_1.mat");
net = trainedNetwork_1;

%perform ODF
[e,locs,t_e] = odf(x,fs);

%set the step
step = 0.02*fs;
chords = [];

for i=1:length(locs)-1
    a = t_e(end)*locs(i)/length(e);
    if round((a+1)*fs) > length(x)
        sel = x(round(a*fs):length(x));
    else
        sel = x(round(a*fs):round((a+1)*fs));
    end
    %compute chromagram
    %for every chord detected
    [cr,t,f] = chromagram(sel,fs, fmin, bpo, noo, step);
    %filter with 5 points; avg filter or geometric mean
    % cr = avg_filt(5,cr);
    cr = geometric_mean(5,cr);
    % try the trained network
    h1 = figure('Visible','off');
    imagesc(cr);
    set(gca,'Position',[0 0 1 1],'XTick',[],'YTick', []);
    I = getframe;
    %resize chroma to fit the network
    I = imresize3(I.cdata,[227 227 3]);
    %classify each chord
    preds = classify(net,I);
    %display each chroma with label
    figure
    imshow(I)
    title(preds)
    chords = [chords preds];
end

%display chords from audio file
c = string(chords);
for i=1:length(c)
    c(i) = erase(c(i),'chroma');
    c(i) = upper(c(i));

```

```
end
fprintf('The chords from this audio file are: ')
fprintf(' %s ',c)
fprintf('\n')

%display most frequent chord
[s,~,j]=unique(c);
freq = s{mode(j)};
fprintf('The most frequent chord from this song is: ')
fprintf(' %s ',freq)
fprintf('\n')
```