Cybernetics and Robotics

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Measurement

# Software for the detection and classification of acoustic impulse events

Bc. Michel Jabali

Supervisor: Ing. Jakub Svatoš, Ph.D.
May 2023

## Acknowledgements

Soli Deo Gloria - To God alone be glory. I would love to thank my Savior and Lord Jesus Christ, before whom I walked, for He has dealt bountifully with me. The God who has been my Shepherd all my life long to this day. The Messenger who has redeemed me from all evil, for He have delivered my soul from death, my eyes from tears, my feet from stumbling. Therefore I could walk before the Lord in the land of the living.

I also want to thank my supervisor, Assistant professor Jakub Svatoš for his valuable guidance and precious counsel all the time of my work with him. And mainly for his long patience when things were not working well as expected on my side.

Last but not least I got to thank my parents for providing every necessary help with support throughout all the years of my studies, so one day I could successfully finish.

## Declaration

I declare that this work is all my own work and I have cited all sources I have used in the bibliography.

Prague, May 26, 2023

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 26. května 2023

## Abstract

This thesis deals with detection of an acoustic event, its training and the following analysis to recognize the voice using an automatic acoustic software. Mel-frequency cepstral coefficients (MFCC) have been used in speaker recognition as well as in speech identification, mainly because they have been empirically checked and verified to work well for audio recognition in general due to their ability to capture important spectral information. These coefficients reckon on a mel-frequency logarithmic spacing of filter bank energies that imitate the non-linear frequency response of the human hearing.

The objective of this research is to develop an effective audio signal processing system that leverages MFCC or other cepstral coefficients for various applications, such as audio event detection.

The thesis evaluates the developed audio signal processing system using extensive experiments and comparative analysis. The system performance is assessed in terms of accuracy, robustness, and computational efficiency. The results show effectiveness of MFCC-based approaches in detecting, training, and recognizing audio signals for a range of applications.

Overall, this thesis contributes to the field of audio signal processing by providing insights into the application of MFCCs for audio event detection and recognition. The findings offer valuable guidance for the development of efficient and accurate audio processing systems and pave the way for further advancements in the field.

**Keywords:** Audio signal, feature, filter bank, Mel-scale, cepstral coefficients, classification, neural network, training.

## Abstrakt

Tato práce se zabývá detekci akustické události, její trénování a následnou analýzu k rozpoznávání zvuku pomocí automatického akustického softwaru. Mel-frekvenční kepstrální koeficienty (MFCC) byly použity při rozpoznávání mluvčích i při identifikování řeči, především proto, že byly empiricky zkontrolovány a ověřeny, aby dobře fungovaly pro rozpoznávání zvuku obecně kvůli jejich schopnosti zachytit důležité spektrální informace. Tyto koeficienty počítají s mel-frekvenčním logaritmickým rozestupem energií banky filtrů, které napodobují nelineární frekvenční odezvu lidského sluchu.

Cílem této práce je vyvinout efektivní systém zpracování audio signálu, který využívá MFCC či jiné spektrální koeficienty pro různé aplikace, jako je detekce zvukových událostí.

Nakonec práce hodnotí vyvinutý systém zpracování audio signálu pomocí rozsáhlých experimentů a srovnávací analýzy. Výkon systému je hodnocen z hlediska přesnosti, robustnosti a výpočetní účinnosti. Výsledky demonstrují efektivitu přístupů založených na MFCC při detekci, trénování a rozpoznávání audio signálů pro řadu aplikací.

Celkově tato práce přispívá do oblasti zpracování zvukových signálů tím, že poskytuje pohled na aplikaci MFCC pro detekci a rozpoznávání zvukových událostí. Zjištění nabízejí cenné vodítko pro vývoj účinných a přesných systémů pro zpracování zvuku a dláždí cestu pro další pokrok v této oblasti.

**Klíčová slova:** Audio signál, znaky, filtr bank, Mel-škálování, kepstrální koeficienty, klasifikace, neuronová síť, trénování.

# Contents

## Figures

## Tables

# Introduction

With the growth of technology, the interest of checking it through a lot of means or controlling it in many ways for the simplification of work. Today we have numerous methods - for example, by sending sound signal through microphones, or by moving a part of our body in front of a device using cameras, and so forth.

As part of the research, many projects are freely available in different programming languages, which are already abundant used in various applications. Due to the popularity of such applications among users, as well as a large one military use, I decided to devote myself to a similar topic with a hope of developing an application for voice events detection and classification, to improve the process of monitoring using microphones. Due to the wide availability of them in many devices today, the option is offering the work in a lot of directions.

Due to the fact that we as people have five senses by which we perceive the surrounding world, but only three of them, precisely sight and hearing and touch, we have enough means of technology to work with. Regarding taste and smell science is not that far yet in developing enough methods to process and manage with them. Given the fact that in my bachelor thesis I worked with image processing, and already back then I did notice, how limited image processing is, because by it we can process and treat only the areas we are aimed at with our camera system. For example, capturing a public building from all different angles we cannot know about more details happening outside of our capturing perception. That is the reason that back in those days which made me thinking of other body sense to work with in my future survey either in school or after graduating.

Because of dangerous and risky moments people might face in life, especially in the last three decades regarding shooting in public areas, any kind of systems that could detect, localize, classify, recognize or anything else to catch the source of danger is appreciated to protect lives. And as mentioned above, visual monitoring systems in general are not enough to capture dangers, particularly fast gun-shooting events from angles not caught by cameras, because of their placing in corners or at ceilings in rooms. Such placement, albeit good to capture as much as possible, are known to human eyes, and to criminals who previously study and measure the area in order to not be caught. And so other techniques are demanded and called for, one of which could be acoustic system based on standalone units, which continuously monitor its surrounding, and in case of continuing threat, to track its source. In comparison with video monitoring systems, acoustic surveillance systems have the advantage to track the source, especially in crowded places or on rugged terrain.

Monitoring using cameras on one hand and using microphones on the other are two distinct approaches that provide different types of information and serve special purposes, and the choice between using a camera or a microphone to capture an environment and gather data depends on the specific goals, context, and type of information we are interested in. Having many similarities, like event detection, how both monitoring methods can be used to detect and identify specific behaviors of interest (events). Yet cameras can only detect visual events like motion, object presence, or changes in the scene, whereas microphones can detect solely audio events such as specific sounds, false alarms, or abnormal noise patterns. With the detection method come the differences. For example, sensory input, for cameras primarily capture visual information, providing an eye's perceptible representation of the monitored area. They are effective in detecting and observing spatial relationships between objects, and physical movements with changes. On the contrary, microphones capture auditory characteristics, enabling the monitoring of sound patterns, environmental noise levels, and speech pattern changes.

Inseparably to sensory input is data processing. Cameras monitor primarily image frames, which require visual processing techniques, such as Image Filtering, Image Segmentation, Edge Detection and many other methods to extract appropriate image information. While microphones demand analyzing audio signals, which call for signal processing, spectral analysis, or pattern recognition to obtain relevant insights about the audio signal.

As the significant advantage for microphones compared to cameras is regarding the environmental itself and its factors. While cameras rely heavily on lighting conditions and visual obstructions, including weather or objects in their way, which can affect the quality and accuracy of the captured visual data. Microphones on the other hand, are generally less affected by environmental factors and can capture audio data in a not-fitting weather or in a low light workspace.

In recent decades, the field of audio and speech processing has witnessed significant advancements in developing effective techniques for analyzing and understanding various aspects of audio. Among the widely used methods, Mel Frequency Cepstral Coefficients (MFCC) and other cepstral coefficients, like Inverse Mel Frequency Cepstral Coefficients (IMFCC), Linear Frequency

Cepstral Coefficients (LFCC), Gammatone Cepstral Coefficients (GTCC) have emerged as powerful tools for extracting meaningful features from speech and audio signals. These coefficients capture the spectral characteristics of the signals in a compact and efficient manner, making work applications easier. Among such applications are speech recognition, speaker identification, and sometimes even music genre recognition and classification.

The goal of my thesis is to explore the usefulness and capability of MFCC and other cepstral coefficients, along with spectral entropy and spectral centroid in audio analysis. By examining and exploring the fundamentals of cepstral analysis, its computational techniques, and applications of its coefficients, I strive for gaining a deeper understanding of their variations, restrictions, specific advantages and disadvantages for farther research.

The thesis begins with a basic overview of the fundamental concepts in speech and audio processing, including the basic principles of signal representation, Fourier analysis, and spectral characteristics of audio signals. This foundational knowledge will lay the basic for understanding the motivation behind the development of MFCC and other cepstral coefficients as feature extraction procedures.

Next, the thesis continues with an overview of MFCC computation, exploring the individual steps involved in the process, including steps such as framing, windowing, Fast Fourier Transform, Mel filter bank application, logarithmic compression, and Discrete Cosine Transform, which further shape the spectral information and generate the final cepstral coefficients. The implementation details and parameter choices for each step are thoroughly explored, considering their impact on the quality and discriminative power of the MFCC representation.

The thesis will then examine and explore the of cepstral coefficients effectiveness in speech recognition identification systems. Through a large experiments and performance evaluations, an assessment of the performance is included and robustness of these coefficients across different possibilities.

To address the task of audio event detection, the thesis investigates techniques for identifying specific events or activities within audio recordings. Convolutional neural networks (CNN) as machine learning algorithms, has been involved to train classifiers using MFCC and other cepstral features. The thesis explores feature selection methods, classifier optimization techniques, and the challenges associated with real-world audio datasets.

The thesis also focuses on speaker identification, aiming to develop a system capable of recognizing individual false alarms and particular guns based on their gunshots' voices. So the thesis explores the impact of feature representation, classifier selection, and dataset characteristics on the kind/type/class classification performance. Speaker-specific MFCC models are trained using supervised learning techniques, and efficient matching algorithms, including dynamic time warping, are employed to perform speaker recognition. The thesis investigates the effects of different factors, including the number of MFCC coefficients, the size of the training dataset, and the choice of classifier, on the accuracy and robustness of speaker identification.

In summary, if an acoustic event is detected, an algorithm based on Mel Frequency Transformation and classification using Support Vector Machine is performed, because Mel coefficients could represent timbre quite accurately. With our capturing sensors we can not only accurately and immediately localize the source of the acoustic event, but also to track the source a identify its type after deep learning through all sorts of sound, because the voice of breaking a heavy pitcher is different than a glass of wine though both of them could be made of the same material, as much as knocking on the door is different than closing it, etc. If the gunshot is recognized, the detection system is designed to immediately calculate the coefficients needed for training and apply deep learning (to recognize and capture the source type.)

My hope is that through this thesis, I might contribute to the existing knowledge about cepstral coefficient and provide any amount of insight that might enhance the performance and applicability of cepstral coefficient-based methods for future researches. In anticipation of the thesis' conclusions to serve as a valuable resource for developers and research workers, in the field of speech and audio processing, helping them to enhance the using of MFCC and other cepstral coefficients for future analysis.

# Chapter 1

# Analysis

## 1.1 Audio signal basis information

Audio signals are pervasive in our daily lives, carrying information and serving as a fundamental means of communication, whether speech, music, or telecommunication. Studying to understand them and later analyze audio signals is essential for a wide range of applications, including speech recognition, speaker identification, noise reduction, audio synthesis, and in recent years even as music analysis. The study of audio signals involves exploring their underlying properties, processing techniques, and mathematical representations to extract meaningful information and gain insights into the underlying audio content.

To purely and simply define an audio signal we can say that it is a representation of sound waves, using a changing level of electrical voltage for analog signals, or using a changing series of binary numbers for digital ones. Audio signals are essentially time-varying representations of sound waves, which are generated by vibrating objects and transmitted through a medium (microphone, speaker). As a generating source human speech, animal roaring, natural event, such as a blowing wind, sea waves, raining drops touching a surface, or any other kind. Along the main features, like frequency, amplitude, and phase, by which we characterize sound waves, other parameters are used as well, such are the energy level in decibels, and the voltage level in volts, bandwidth (a hertz measured difference between the upper frequency and its lower counterpart on a continuous band.), and nominal level (the operating level at which an electronic signal processing device is designed to operate).

In our days, days of digital technology, is more available to work with audio in a digital form, which concludes to convert analog signal into discrete-time signals using an analog-to-digital converter (ADC). One key aspect of audio signal analysis is the representation of the sound wave in the digital domain, where the already converted signal into digital form is typically represented as a sequence of discrete samples, where each sample represents the amplitude of the signal at a specific point in time, and the rate at which these samples are taken is known as the sampling rate, measured in samples per second

(Hz). The accuracy of the digital representation is determined by the bit depth, which defines the number of bits used to represent each sample.

Audio signals manifest diverse characteristics, and as most crucial I believe are pitch (do not confuse with tone as sound's quality), timbre, and intensity. Pitch refers to the perceived frequency of a sound, determining its musical or tonal quality. Timbre describes the unique qualities of a sound that allow us to distinguish between different instruments or voices. Intensity represents the loudness or amplitude of the sound.

There are many methods in the field to process audio signals with various embedded techniques to analyze, manage, operate, and eventually extract meaningful information from audio signals. These techniques encompass a wide range of approaches, including frequency (or time) domain analysis, digital filtering, and machine learning, in order to reduce noise, extract feature, and recognize patterns. To name a few of the used techniques: Data compression (reduces the broadcast bandwidth and storage requirements of audio data), acoustic detection (means to measure the ability to differentiate between information-bearing patterns and random patterns that distract the information i.e. noise), filtering (frequency dependent circuits, working in the frequency range from Low-pass to High-pass), and many other methods.

This research aims to explore the fundamentals of audio signal analysis, focusing on specific aspects such as pattern recognition techniques, spectral analysis, and feature extraction. By understanding the underlying principles and methodologies, we can develop effective approaches for applications such as speech recognition, and speaker identification. The output of this thesis will contribute to the advancement of audio signal processing and facilitate the development of innovative solutions in audio-related fields.

## 1.2 Peak detection

Peak detection in audio files, usually *.wav* format, is a fundamental task in audio signal processing, in order to detect the significant points in acoustic events. Peaks represent significant changes or spikes in the amplitude of the audio signal and often correspond to important events or features in the audio data. The process of peak detection involves identifying the locations and magnitudes of the highest peaks in the audio signal. One common approach is to analyze the amplitude envelope of the signal, which represents the variation of signal amplitudes over time. The envelope can be obtained by applying an envelope extraction technique, such as rectification, low-pass filtering, or in some cases using the absolute value of the signal.

Once the amplitude envelope is obtained, the next step is to determine the threshold for peak detection. The threshold represents a minimum amplitude value above which a peak is considered significant. This threshold can be set manually based on the characteristics of the audio signal or determined automatically using statistical methods, such as estimating the background noise level or analyzing the distribution of signal amplitudes. After setting the threshold, peak detection algorithm scans through the amplitude envelope to identify regions where the amplitude exceeds the threshold. These regions are considered potential peaks. To accurately detect peaks, additional criteria can be applied, such as requiring the amplitude to exceed the threshold by a certain margin or checking for a minimum peak duration to avoid detecting transient noise spikes. Once potential peaks are identified, the algorithm selects the highest point within each peak region as the peak sample. The sample index and corresponding magnitude represent the location and intensity of the detected peak in the audio signal.

For my particular case I divided the event into 8 pieces (windows), where I found the mean value of each part. In view of the fact that the time course of all parts, i.e. graph shape, is hopping around the zero amplitude axis, all mean values will be very similar. It is because, in general, the more the waver is higher above zero, the more it goes under zero. Needless to say, the mean value of the window with a high peak and its following echos is greater than a silent window. So having the mean value of all windows, which was a little bit more than zero, I gave a trial based on empirical attempts and found a suitable value for the threshold which will always be above all echos. In Figure 1.1 we can see a recorded acoustic signal corresponding to a 556mm gunshot with its echos and threshold.

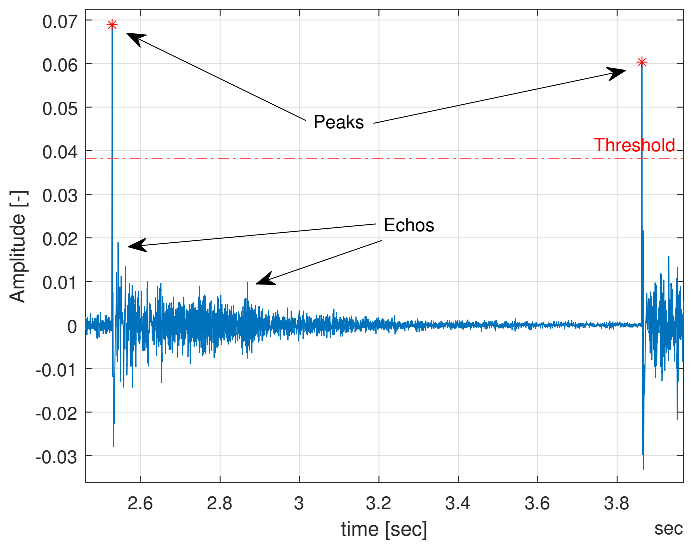


**Figure 1.1:** Peak detection in shooting by 556mm.

After finding individual peaks, one millisecond was reserved from before the peak location and nine milliseconds after it. The reason for that contrast is due to the audio course itself, since before any peak the sound wave carrying information does not have as important data as after, which my supervisor explained to me that run-ups bear less interesting data for future features extraction than the descending parts.

It is worth noting that peak detection algorithms can be sensitive to various factors, among which are the threshold value, noise levels, and signal characteristics. Therefore, careful consideration should be given to selecting appropriate techniques and parameters based on the specific requirements of the application and the characteristics of the audio data. For certainly, situations can happen where two close peaks might coincide a thus the seeking for data will overlap, see Figure 1.2. For such cases the software is designed to avoid those collapses.

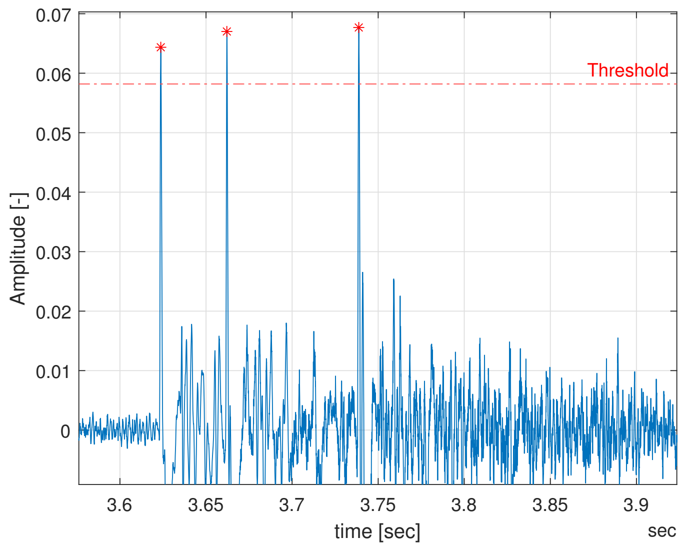


**Figure 1.2:** Close peak detection in shooting by 9mm.

For example, I have met softwares that allow Peaks' Merge error, when if two peaks are very close to each other in terms of their sample indices, a peak detection algorithm considers them as a single peak. And so in that case, the algorithm selects only the higher peak within the combined peak region as the representative peak to the entire area. Unfortunately, this technique results in the loss of information about lower individual peaks, since it prioritizes one peak over others close to it. This prejudice may certainly impact the representation of the detected peaks and will lead to inaccurate results or interpretations.

Peak detection process involves identifying significant changes in amplitude, which correspond to peaks in the audio signal. By analyzing the amplitude envelope and applying thresholding techniques, peaks can be accurately detected, enabling further analysis, feature extraction, or event segmentation in audio processing applications.

It is important to note that the specific behavior and outcome may vary depending on the implementation details of the peak detection algorithm and the characteristics of the audio signal being analyzed. Different approaches and parameters can be employed to address the challenges posed by closely spaced peaks, such as adjusting the detection threshold, employing advanced peak separation algorithms, or using adaptive techniques to account for variable peak distances.

Though Matlab allows the size of an array to grow greater and does not force users to define it along array's declaration, compared to other programming languages, the way data are collected in the program is by nested structures, because an ordinary array has to contain all the elements of same data type, and to solve our problem numerous data types are required to be accumulated at once.

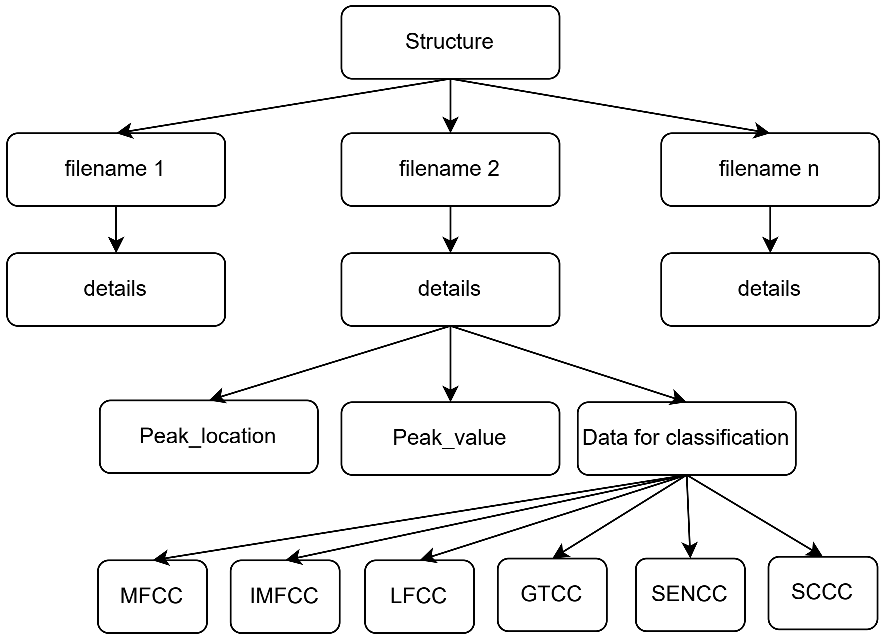


**Figure 1.3:** The initial nested structure used in the program.

Note: The software is designed in way that many *.wav* files could be read without restarting it (starting loading variables from the beginning) and their necessary information gathered in one structure.

## 1.3 Mel Frequency Cepstral Coefficient (MFCC)

Mel Frequency Cepstral Coefficients are a set of features that are widely used in speech and audio signal processing for assignments such as speech recognition with speaker identification, and if need be the classification of his voice. This type of cepstral coefficient being in the best recognition accuracy relative to other acoustic analysis, because they are derived from a type of cepstral representation of real or, precisely, nonlinear audio waves. Using this frequency bending one is allow for better representation of sound.

Those characteristics are attributed to their good representation of the human substantial aspects of perception of short-terms speech spectrum. MFCC rely on a Mel-frequency spacing of filter bank energies, which imitate the frequency response of the human auditory system's reaction to different frequencies. The main difference between common cepstrum and the mel-frequency one is the frequency bands, which are equally spaced on the mel scale that approximates the human auditory system's response more closely than the linearly-spaced frequency bands used in the common spectrum.

MFCC's are derived from the power spectrum of a signal and provide a compact representation of the spectral envelope of the signal. The operating procedure is the following:

- Framing the signal into several short frames.
- Power spectrum for all frames.
- Mel filter bank to the power spectra, sum the energy in each filter.
- Logarithm of all filter bank energies and DCT of the log filter bank energies. [SH23] [Lyo09] [Mat22]

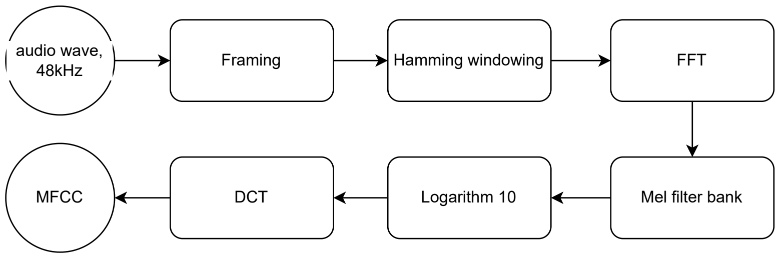


**Figure 1.4:** MFCC extraction in block diagram.

The resulting MFCCs can be understood of as a compact representation of the spectral envelope of the audio signal. Those spectral characteristics of the audio signal record crucial aspects of the signal's frequency and are less sensitive to variations in the signal's amplitude than usual spectral data. MFCCs are frequently used as input features for speech and speaker recognition, machine learning algorithms in various audio-related applications due to their effectiveness in representing the signal's acoustic characteristics.

### 1.3.1 Window frame

Any audio signal is constantly changing, more or less depending on the circumstances of its origin. To simplify the issue without ignoring frequent changes, one could safely assume that on short time scales the audio signal doesn't change much, precisely that frequencies in a signal are stationary over a short period of time.

The window size can significantly affect the achievement of MFCC features, because the framing window determines the size and the duration of frames into which the signal is divided before computing the coefficients. Though not often the rectangular window function is used to frame all windows in the signal, because of its generality due to peaks' position. And for my particular solution where peaks are always located in the *10-th%* in the window rectangular windowing was the best choice. If the original decision would have been to capture windows, in which peaks be placed right in the middle, then Hamming window function would have been a better resolution, since it represents a type of narrowing window that successively reduces the amplitude of the signal towards the edges of the frame, see figure 1.5 frequency domain, compared to figure 1.6 where the tapered window sustain its magnitude.

According to characteristics of the application output, the specific choice of framing window size is made. If in the output we require longer span features to be extracted from the original signal, then we must apply a larger window size as more appropriate, whereas smaller window size is more applicable for signals with shorter span features.

Usually framing signals take place into $20-40ms$ frames, because the speech waveform is generally regarded stable in this interval. For my particular case I chose *25ms* because it was the best fit to divide the features gained by Peak detection with, as a good balance between capturing satisfactory amount of information about the signal and minimizing the effects of spectral leakage/outflow. In case of the frame shorter than $20ms$ we don't have enough samples to get a reliable spectral estimate. On the other hand if it is longer, then signals change too much within the frame. Besides, any speech signal that is finite in time, and thus, any processing is only possible on limited number of sampled windows.

The amount of overlap between frames needs to be taken into consideration because it has an impact on the performance of any coefficient feature extraction. The advantage of overlapping frames is to capture more information about the signal, however they might as well increase the computational cost of the process which is a disadvantage. Similar to the choice of frame window due to the specific requirements of the output, the overlapping should be based on allowed for. For my particular case I tried and tested various overlaps with framing window size 25ms, where I found out that the most fitting overlap is no overlap at all, simply to keep clear all the data I work with inside each frame separate without mingling.

Since the solution needs to be universal for any input audio of which its length is unknown, not just framing is recommended to be in a specific interval, but also the frequency contours of the signal over time would get lost if the Fourier transform is applied across the entire signal. Therefore, by doing a Fourier transform over $25ms$ time frame, we obtain a good approximation of the frequency contours of the signal by associating the nearby frames. This windowing step is needful to reduce signal interruptions, and make the ends smooth enough to connect with the following beginnings.

So by dividing the input signal into considerably small no-overlapping frames and applying a window function to each frame, the window function helps to reduce spectral leakage, where the energy of a signal at one frequency leaks into nearby frequencies. After cutting the signal up into separate frames, to each frame I originally applied the Hamming window function, one which smooths windows and has the following form:

$$w(n) = 0.54 - 0.46\, cos\left(2\pi\frac{n}{N},\right) \tag{1.1}$$

where w(n) is the value of the window at sample n, and $n \in (0, N)$, with $N + 1$ as the window's frame length. Applying the (1.1) equation for 128 points we obtain the following graphs of time and frequency domains.

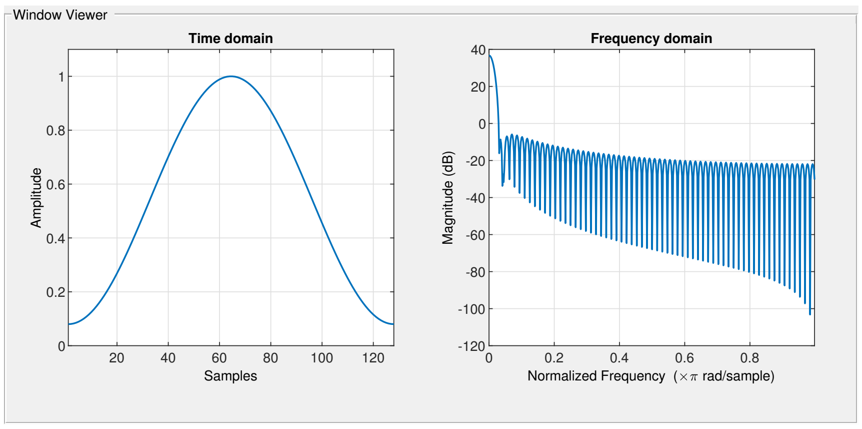


**Figure 1.5:** A symmetric Hamming window of 128 points.

The greater number of of points used, the more accurate and smoother the plot is in the time domain, and more inflated function in the frequency domain.

Flat top weighted windowing, Blackman windowing and other window functions could be used for cepstral coefficients extraction, where each window function has its own set of characteristics that impact the computation of the cepstral coefficients. For example, the Hamming window has a smoother transition from the frame center of to the edges, while Blackman is designed to minimize spectral leakage, enhance frequency resolution to provide a trade-off between main lobe width and side lobe levels, meanwhile Flat top weighted is designed for accurate amplitude measurements in the frequency domain, for it provides very precise representation of the true amplitudes of spectral elements, because it has a very low level of spectral leakage, is often used in applications that require accurate amplitude measurements, such as power spectral density estimation. Needless to say that Flat top weighted, Blackman and Hanning functions have similar shaping in time domain with different edges compared to hamming function.

Regardless of the advantages mentioned above about window functions, a different one not commonly used for spectral analysis was chosen in the end rectwin(L) i.e. Rectangular window which returns a rectangular window of length parameter L [1], because it forms a rectangular shape in the time domain. The reason for this choice is due to placing each peak in the frames. In 1.2 was mentioned how one millisecond before each peak and nine milliseconds after it were reserved for data processing, and on the grounds of that fact, non of the window functions mentioned above, though practically tested, could be used, for peaks' locations were no in the middle, but 40% more left.

On the basis of my supervisor's help, the rectangular window function was chosen as best fit due to peak's location. Mathematically, the rectangular window is expressed as:

$$w(n) = \begin{cases} 1, & n \in (0; N) \\ 0, & otherwise \end{cases} \tag{1.2}$$

where w(n) is the value of the window at sample n and N is the length of the window.

The signal is multiplied by a constant value of 1 in the window's duration and zero outside. From which we can conclude that the rectangular function does not taper the signal towards the edges of the window, and as a result, it provides equal weight to all samples within the window. Compared to other window functions mentioned above which offer smoother transitions and better control over spectral leakage, the rectangular's main issues is spectral leakage, where the sharp transitions at the edges of the window introduce unwanted frequency components in the Fourier transform. It should be noted that leakage factor in for hamming function is *0.03%*, whereas for the rectangular one is *9.14%*. (The greater the number of points is, the less is leakage factor.)

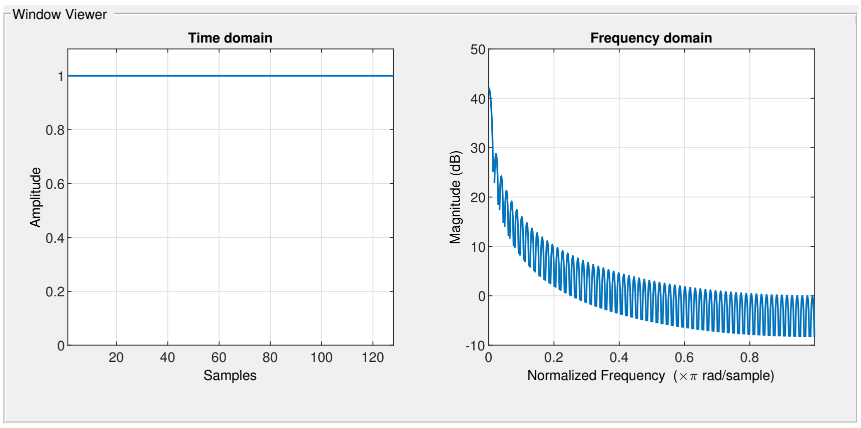


**Figure 1.6:** A rectangular window of 128 points.

[1]https://www.mathworks.com/help/signal/ref/rectwin.html

### 1.3.2 Fourier-Transform and Power Spectrum

Note: Sometimes in literature we might run across Discrete Fourier Transform, i.e. DFT instead of FFT. But Matlab uses primarily FFT because for computing the DFT with reduced execution time [2]. And because the FFT length of bins used in the window input must be greater than or equal to the number of elements in the Window using DFT. Occasionally, I found "Short-Time Fourier Transform", i.e. STFT, but mostly STFT is performed on a computer using the fast Fourier transform.

Fast Fourier Transform is a mathematical tool that decomposes a time-domain signal into its frequency components. It expresses the signal as a sum of complex sinusoidal functions of different frequencies. This type of transform allows users to analyze the frequency content of a signal and extract information about the amplitudes and phases of its frequencies. To extract the desired cepstral coefficients, FFT is applied to each frame of the audio signal to acquire the frequency spectrum which represents the distribution of signal energy across different frequencies in the frame.

Now the necessity to use Fast Fourier transform with 512 point-size on each frame to compute the frequency spectrum, and then calculate the power spectrum, i.e. periodogram using the following equation with the squared magnitude of each frequency bin in the Fast Fourier Transform:

$$P = \frac{1}{N}\left|FFT(x_i)\right|^2 \tag{1.3}$$

where $x_i$ is the $i^{th}$ frame in signal $x$. The final result is the square of the absolute value of the complex fourier transform. Though I generally performed a 512 point-size FFT, I kept only the first half coefficents +1, due to Hermitian symmetry that divide the FFT output into two halves, or the Nyquist frequency as the maximum frequency that can be represented in a discrete signal. More about that in the Implementation chapter.

The one dimensional DFT used, is defined as:

$$DFT = \sum_{i=0}^{N-1} x_i \, e^{-j2\pi ki/N} \tag{1.4}$$

where N is sample long analysis window, or the DFT is defined on the set of N samples.

The 1.3 operation gives a real-valued spectrum that represents the energy distribution across frequencies and provides a useful representation of the signal's spectral content for the cepstral coefficients extraction.

### 1.3.3 Mel Filter Banks

The Mel-scale is a perceptual scale that relates the frequency of a pure tone to the perceived pitch of that tone. The Mel scale is nonlinear and has a higher

[2] https://www.mathworks.com/help/signal/ug/discrete-fourier-transform.html

resolution at lower frequencies and a lower resolution at higher frequencies, aligning more closely to human hearing perception.

Precisely Mel-scale filter banks, is a set of 20 to 40 triangular filters used in audio signal processing that is applied to the periodogram power spectral estimate from the previous step to extract frequency bands, banks with fixed center and cut-off frequencies. They were designed to divide the frequency spectrum of an audio signal into a set of overlapping triangular filters, spaced according to the Mel scale. In other words, to simulate the non-linear frequency response of the human auditory system, see 1.7 and 1.8, where they help capture the perceptually relevant spectral information in a compact and efficient manner. The Mel-scale is a perceptual scale that approximates the way humans perceive differences in frequency. It is based on the observation that humans are more sensitive to changes in frequency at lower frequencies compared to higher ones. This scale is logarithmically spaced and covers a range of frequencies from low to high. It is important to define the filter center frequencies that are evenly spaced on the Mel scale. The lower and upper limits of the filter bank are typically set to the lowest and highest frequencies of interest in the signal.

The conversion between Mel-vector $m$ and the frequency one $h$ is based on the following two equations, where the spectral resolution becomes lower as the frequency increases:

$$m = 2595\ log_{10}\left(1 + \frac{h}{700}\right) \tag{1.5}$$

$$h = 700\ \left(10^{m/2595} - 1\right) \tag{1.6}$$

Sometimes in literature I run across a similar equation to 1.5, converting from Hertz to Mel scale using natural logarithm instead of Common logarithm (base 10). The equation is the following:

$$m = 1127\ ln\left(1 + \frac{h}{700}\right) \tag{1.7}$$

Later the Mel frequencies are converted back to Hertz scale using the inverse Mel-scale transformation, where the conversion maps each Mel frequency to its corresponding Hertz frequency.

$$h = 700\ e^{\left(\frac{m}{1125} - 1\right)} \tag{1.8}$$

My filter banks comes in the form of 26 vectors of length 24e3 Hertz based on the frequency vector. We see how each the shape od all triangular filters on the Mel-scale is defined by three points on the frequency axis: the lower and upper cutoff equals 0, the center frequency equals 1. More about it in Implementation chapter.

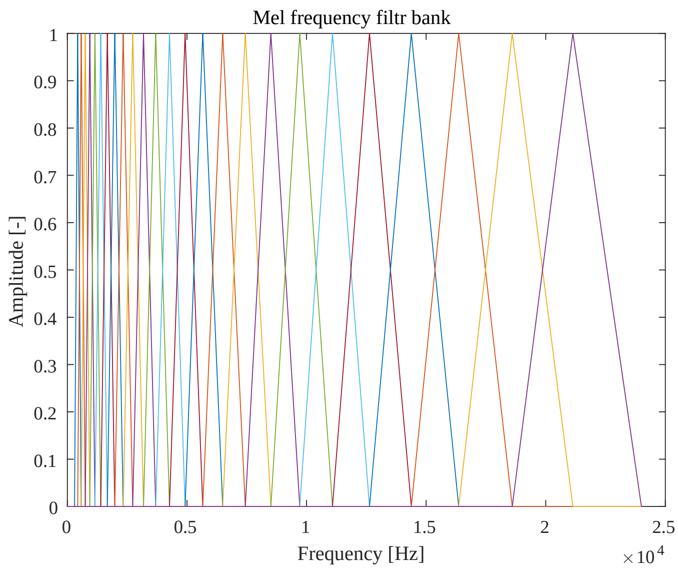


**Figure 1.7:** Filter bank on Mel-Scale.

The filters in the Mel filter bank are overlapped to ensure that the entire frequency spectrum is covered without any gaps or overlaps. The amount of overlap between adjacent filters is typically determined empirically and may vary depending on the application.

The reason why more filters are on the lower part of the frequency spectrum is due to similarity to the perception of human hearing, since human recognition of the frequency for any speech does not follow a linear scale, but the perceptive frequency is measured on the Mel-scale, based on actual aural experiments. By taking a closer look on the following figure 1.8 we see the non-linearity of the human audio perception, which is more sensitive to lower frequencies.

By computing the area under each filter in figure 1.7 is the contained energy of each band. In other words, the output of each filter represents the energy in the corresponding frequency band, which is going to be applied later for obtaining the spectral entropy. The wider filter is, the more energy it contains. Filters that are more closely spaced with narrower bandwidths at frequencies with higher speaker discriminative power, and more widely spaced with wider bandwidths at other frequencies. [LG09]

In the MFCC extraction process, the power spectrum is often further processed by applying a filterbank that emulates the human auditory system's response to different frequencies. The filterbank divides the spectrum into several frequency bands, and the energies within each band are summed or averaged to obtain the filterbank energies. These filterbank energies serve as the basis for further computations, such as the logarithmic transformation and the Discrete Cosine Transform (DCT), which ultimately yield the Mel frequency cepstral coefficients.

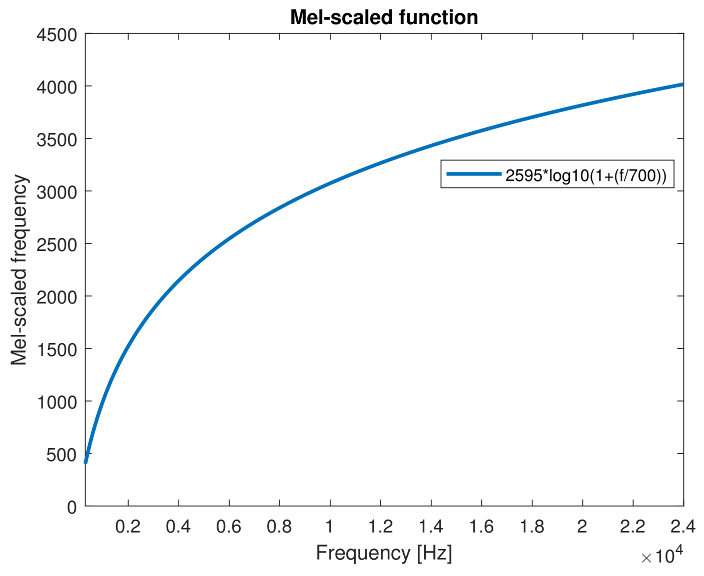


**Figure 1.8:** Mel-frequency scale depending on normal frequency.

The graph above shows how lower frequencies have a larger distance between them Mels, and the contrary, drawing closer its human like characteristic. Because the human cochlea vibrates at different locations in itself depending on the frequency of incoming sounds, where different nerves inform the brain that certain frequencies are at hand depending on the precise position in the cochlea that vibrates, as the Mel-scale aims to imitate the non-linear human ear perception of sound by more distinguishing lower frequencies and less the higher ones. In other words, a scale that is linear in low frequencies and logarithmic in high ones (approximately, linear frequency spacing less than 1 kHz and logarithmic spacing greater than 1 kHz) to represent this non-uniformity. That is why the 1.7 graph is very dense at lower parts of spectrum and almost ignoring higher frequencies, because it is not directed at them. The periodogram power spectral estimate performs a similar job, identifying which frequencies are present in the individual frames.

### 1.3.4 Mel Coefficient

Sometimes after all the computation mentioned above is still possible to obtain correlated Mel coefficients, which in some machine learning algorithms could be difficult to work with. Therefore is highly recommended to apply Discrete Cosine Transform (DCT) to the logarithmically transformed filter bank energies to de-correlate the coefficients and to get a compressed representation of the filter banks. [Fay16] And also the DCT reduces the redundancy in the filter bank energies and focuses on capturing the most significant spectral information. The Mel Coefficients are calculated by using logarithms ($log_i$) in the Discrete Cosine Transform $DCT$.

$$DCT = \sqrt{\frac{2}{N}} \sum_{i=1}^{N} (log_i) \cdot cos\left(\frac{j\pi}{N}(i - 0.5)\right) \tag{1.9}$$

where the N here is the number of filter bands. For my case the resulting cepstral coefficients 2-26 are retained and the rest are discarded because they represent fast changes in the filter bank coefficients and don't help the Automatic Speech Recognition. In the signal representation phase, Mel Coefficients throw away a lot of the information in the sound wave using a fixed filter bank.

The resulting DCT coefficients represent the MFCC, yet not all coefficients have usable information, because usually a subset of the coefficients is selected while discarding others. And that selection is often based on their applicability in capturing the spectral characteristics of the signal. The most common practice is to keep the lower order coefficients and discard the higher order ones, as they tend to contain less biased information/redundancy.

## 1.4 Inverse Mel Frequency Cepstral Coefficients

Inverse Mel Frequency Cepstral Coefficients (IMFCC) is an extension of MFCC that aims to preserve the phase information of the audio signal. It combines the magnitude spectrum from MFCC with the phase spectrum obtained from the inverse Fourier transform of the classical Mel coefficients, and by preserving the phase information, IMFCC allow for better reconstruction of the original audio signal. Yet, IMFCC may introduce additional computational complexity compared to MFCC due to the inclusion of phase information, and their effectiveness depends on the specific application and the importance of phase information in the analysis.

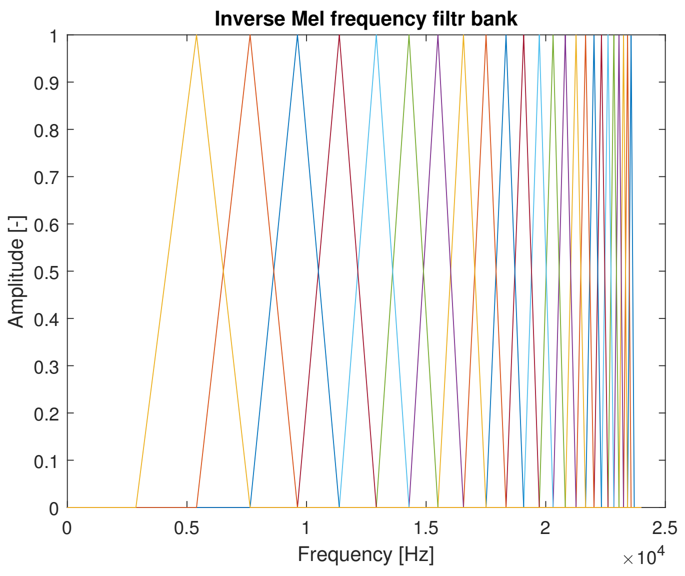


**Figure 1.9:** Filter bank on inverse Mel-Scale.

Note: for an unknown reason Matlab did not save the first filter in here, which is the last purple one in the 1.7, though in before saving the file the purple filter was clearly available.

As a consistent inference, which I found out later by working with inverse Mel coefficients, and should have concluded immediately from the label (inverse) itself, is the fact that IMFCC are able to provide a more precise representation of the audio signal, particularly for tasks involving audio synthesis and alteration.

## 1.5 Linear Frequency Cepstral Coefficients

Linear Frequency Cepstral Coefficients (LFCC) is another alternative to MFCC that uses linearly spaced filter banks instead of the Mel-scale, that is related to use linear frequency scale. Unlike MFCC which models the human auditory system's frequency perception, LFCC do not directly model the human hearing, for the human auditory system is non-linear frequency perception, where the perception of frequency intervals becomes wider at higher frequencies, see 1.8 graph. We may conclude that LFCC is be beneficial in applications where a linear representation of the audio signal is desired, such as music analysis, such as genre classification, or instrument recognition, LFCC could be used where linear frequency perception is more relevant, such as speaker recognition, or for capturing individual characteristics related to the fundamental frequency (F0) of the speaker's voice.

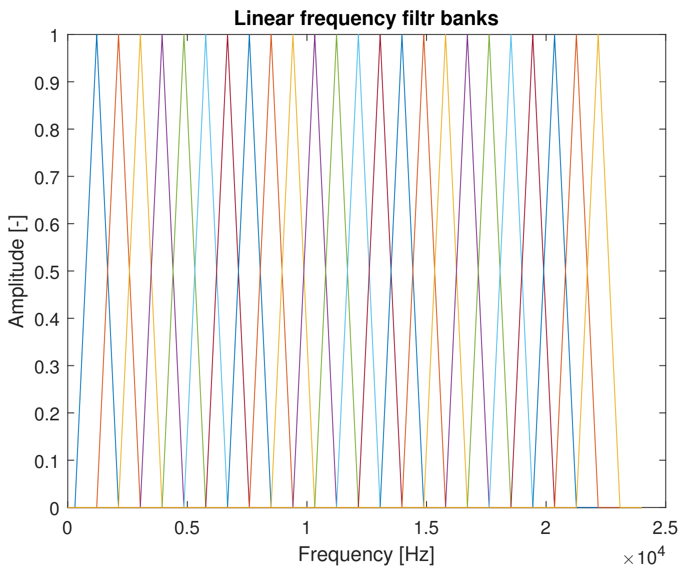


**Figure 1.10:** Filter bank on linear scale.

LFCC may capture fine-grained spectral details more accurately than MFCC, particularly in the higher frequency range, which could be useful in assignments that require precise frequency estimation. However, LFCC may be less robust to noise or variations in the audio signal compared to MFCC, since they do not incorporate the perceptual characteristics of the Mel scale.

## 1.6 Gammatone Filter Banks

Is another type of auditory filter bank used in audio signal processing, similarly to MFCC in modeling the human auditory system. It is designed to approximate the frequency selectivity and response characteristics of the cochlea which holds accountability for frequency analysis in the human ear. The Gammatone filters are effective in modeling the human auditory system's sensitivity to different frequency regions, and so it does provide a good representation of sound that aligns well with human auditory processing, making it fitting for a range of application.

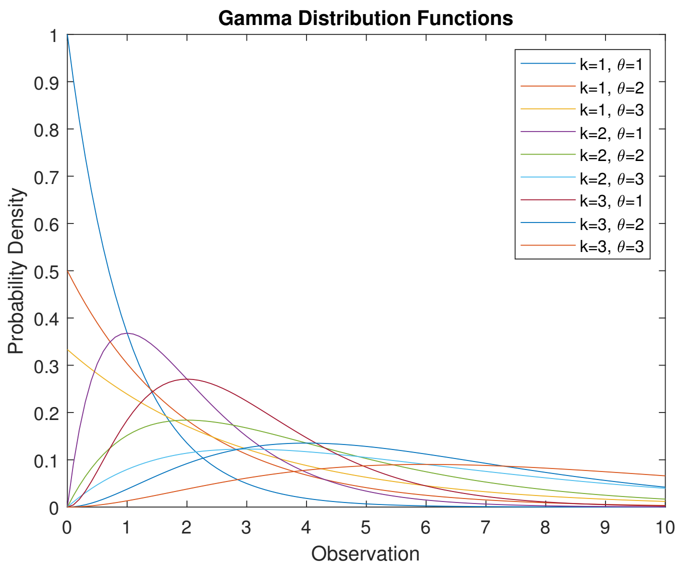


**Figure 1.11:** 9 Gamma Distribution Functions.

where every function depends on $k_i$ as a shape parameter, and $\theta_j$ as a scale parameter.

The issue with Gammatone filter bank is basically the same as with Mel filter bank, i.e. same frequency distribution, but different curving. So, instead of triangular filters based on Band Edges and logarithmically spaces, we use similar curving to the gamma distribution. The name Gammatone itself is derived from the gamma distribution, for it is the shape of each filter, which is characterized by a symmetric shape resembling a Gaussian-like function with a peak at its center frequency.

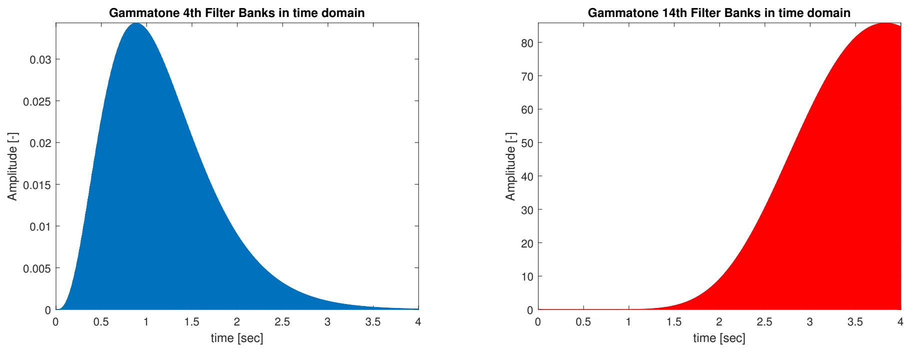
$$g(t) = at^{n-1}\, e^{-2\pi bt}\, cos(2\pi f_0 t + \Phi) \tag{1.10}$$

where $a$ represents the signal amplitude, $t$ is time, $n$ the individual filter order, $b$ is the bandwidth in Hz given the filter's center frequency $f_0$ in Hz, and $\Phi$ is the phase of the carrier in radians. The relation between the bandwidth $b$ and the carrier frequency $f_0$ is:

$$b(f_0) = 1.019 \cdot ERB(f_0) = 1.019 \cdot 24.7\ (1 + 4.37 f_0/1000) \tag{1.11}$$

The carrier/center frequency I would defined as the location of each filter's peak response and defines the specific frequency to which the filter is most sensitive, the frequency around which the filter's response is concentrated. In other words, it represents the primary frequency that the filter is designed to capture.

For verification of the fact, that the shape of each filter, which is characterized by a symmetric shape resembling a Gaussian-like function, I implemented them in time domain based on the 1.10 equation, see the following 2 subfigures.



**(a) :** the 4th Gammatone filter bank. **(b) :** the 14th Gammatone filter bank

**Figure 1.12:** 2 orders of Gammatone filter bank in time domain.

The center frequencies of the Gammatone filter banks are spaced logarithmically or according to the ERB (Equivalent Rectangular Bandwidth) scale, which approximates the bandwidths of the auditory filters in the cochlea, and helps capture the time-varying nature of sound perception. The bandwidths are controlled by the Q-factor, which determines the shape and selectivity of the filters - the higher Q-value, the narrower bandwidth and greater selectivity, meaning the filter will be more focused on a specific frequency region. The exact and final Q-values always vary depending on the desired trade-off between frequency resolution and computational complexity in the application.

$$Q = \frac{f_c}{b(f_0)} \tag{1.12}$$

where $f_c$ is the center frequency, and $bw$ represent the bandwidth. Each filter will have a different center frequency, different bandwidth, and therefore as many Q values as the number of filters in the span of the desired frequency range.

The steps to produce the gammatone filter bank are similar to the Mel spectrogram, yet gammatone consists of bandpass filters in the ERB scale and the shape is obtained as the multiplication of cosine and gamma functions, and in time domain, where after passing an input signal to a gammatone filter bank an impulse response of a particular filter has the 1.10 impulse response. Both Mel and Gammatone filters are computed using the same procedure based on the FFT whose frequency resolutions are set by the size of the Hamming or other windowing. And in the final analysis as well as logarithm and discrete cosine transform are applied. The output of each filter represents the energy in a specific frequency band. Unlike Mel filter, the Gammatone filter is defined in the frequency domain by its center frequency and its bandwidth. So, using a hyperbole language, not just 3 points are required as is the case with Mel filter, but every point.

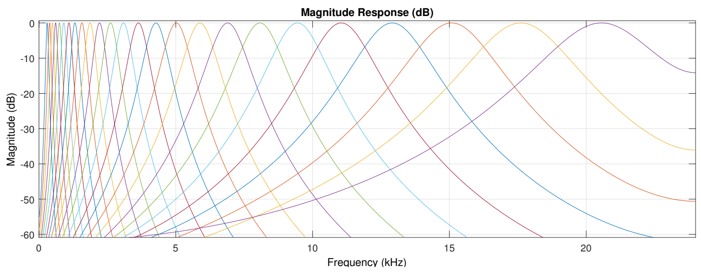


**Figure 1.13:** Filter bank on a Mel-Scale with Gamma curving in frequency domain.

Each filter in the Gammatone filter bank is a bandpass filter with a center frequency and a bandwidth that corresponds to the critical bandwidth of human hearing.

# Chapter 2
# Implementation

In the beginning is necessary to read the audio file, to have its data and the sample frequency which is the basis element for framing. The next step to find peaks with the mean values of each window and verify how the existence of peaks and their echos in a certain window raise the mean value of that window. Adding the peak locations along with their values in a structure or an array, we also add one millisecond of the signal before each peak, as much as nine milliseconds from after all peaks.

## 2.1 Mel Frequency Cepstral Coefficient (MFCC)

### 2.1.1 Window frame

According to the number of elements I gather in the structure, which makes ten milliseconds in total, I found that $25ms$ frames to be the best fit for both, the features dividing number, and to have enough samples with a reliable spectral estimate without signals changing too much within frames. The requirement is to frame until the end of the audio file is reached. In case the file does not divide into an even number of frames, is recommended to fill it with zeros so that it does.

After obtaining the exact number of elements in structures, then through dividing that particular number by the frame division ($25ms$) I found out and suggest to eliminate some lateral elements, so in the final analysis there is no need to pad any frame with zeros in order to make it dividable, see the snippet code below (part of a for loop that scan all peaks in an audio signal) to round any frame with odd number:

```
    % if speech file does not divide into an even number
    if mod(n_frame,2) == 1
        framed(n_frame+1,:) = zeros(size(framed(1,:)));
        n_frame=n_frame+1;
    end
```

where $n\_frame$ represents the number of frames obtained as the length of 10 milliseconds around each peak divided by the frame size, which is usually a 1200 samples achieved by the sample frequency times the framing window constant, i.e. $25ms$.

Rounding the frame number ensures that the frames align properly with the signal samples, which later allows for consistent analysis and avoids any limited/incomplete frames to overlapping. And later while working with many data is very efficient to operate with fixed-size frames to save most memory control, for rounding the frame number to an integer value simplifies memory allocation and processing.

In the next part of code we see a temporary variable *temp* that is increased with each round of loop, where *f_size* equals 1200 samples, and representing sampling frequency times frame division.

```
    temp = 0;
    for i = 1:n_frame % i ranges over the number of frames.
        temp = temp + f_size;
    end
```

The last step regarding window framing is applying smoothing filters to all frames, where originally the sampling was hamming symmetrical, then changed to Rectangular window, see the difference between figures 1.5 1.6 and in time domain, primarily.

```
    for j = 1:n_frame
%       sn(i,j) = framed(i,j)'.*hamming(length(framed(i,j)),'
    symmetric');
        sn(i,j) = framed(i,j)'.*rectwin(length(framed(i,j)));
    end
```

## 2.1.2 Fourier-Transform and Power Spectrum

This is a crucial part in the cepstral coefficients computation, by efficiently computing the frequency spectrum of audio signal frames, providing a representation of the signal's spectral content that is further compiled to extract the coefficient.

Now with results from the previous step we use Fast Fourier transform, of which we take the absolute value since it is a complex function, and square the result, but we only keep the first half coefficients +1 with regard to Hermitian symmetry that divide the FFT in two halves, where the first half lying within 0 to N/2 (both indexes included), contains the unique frequency components, while the other half from index N/2+1 to N-1, is a mirror image of the first half.

In some literature I run across the Nyquist frequency as the maximum frequency that can be represented in a discrete signal, which in the audio processing is half to the sampling frequency, the element at index N/2+1 in the FFT output corresponds to the Nyquist frequency. By taking just the first half of the FFT output, including the Nyquist frequency component as the latest element and 0 as first, we capture all the unique information about the frequency components of the real-valued signal while discarding the redundant information in the second half. This reduces computational complexity and ensures efficient use of memory.

```
    Y = fft(sn);
    P2= abs(Y/n_frame);
    P1= P2(1:n_frame/2+1);
    P1(2:end-1) = 2*P1(2:end-1);
```

The FFT output represents a complex-valued spectrum. However, for cepstral coefficient computation, typically only the magnitudes of the Fourier coefficients are used, as the phase information is less relevant for capturing the spectral characteristics of the signal. And to obtain the power spectrum, which represents the distribution of energy across different frequencies in the frame, the squared magnitudes of the FFT output are computed.

### 2.1.3 Spectral centroid

On the basis of this article about An Entropy-Based Architecture for Detection of Sepsis in New-2 born Cry Diagnostic Systems [ZKT22], section 2.3.3. Spectral Centroid Cepstral Coefficients (SCCC), I discoverd that Spectral Centroid (SC) is a measure of the shape of the spectrum of the signal and the position of the mass of the spectrum, where the mean value of SC was shown to be a discriminative feature that indicates where the major energy of the signal is concentrated.

The article has been a very helpful tool to know the number of Spectral Centroid Cepstral Coefficients (SCCC), though there is no fixed or standard number of coefficients for Spectral Centroid, which is not dependent on the number of Mel Frequency Cepstral Coefficients (MFCC), because each of these two types of cepstral coefficients capture different aspects of the audio signal. Since I run across various numbers in books or articles, where inherently no one specified the exacts number of coefficients for the spectral centroid, besides the used article above, where the authors define the number by writing: "SC denotes the center of the signal's gravity and is computed from taking the weighted mean of the frequency bins. The SC value, Ci of the i-th window is computed by

$$C_i = \frac{\sum_{k=1}^{W_{f_l}} kX_i(k)}{\sum_{k=1}^{W_{f_l}} X_i(k)} \tag{2.1}$$

Where $x_i(n)$ are the i-th window samples, and $X_i(k)$ are the DFT coefficients." From which I concluded that for each window could be only one spectral centroid cepstral coefficient, see the following figure 2.1, where five SCCC are depicted. So, the spectral centroid is a single scalar value representing the center of weight of the power spectrum, and their number is usually smaller than the number of MFCC or any other cepstral coefficients. Their selection is altogether independent on other cepstral coefficients and based on factors such as the desired level of accuracy, computational constraints, or the type of speech signal.

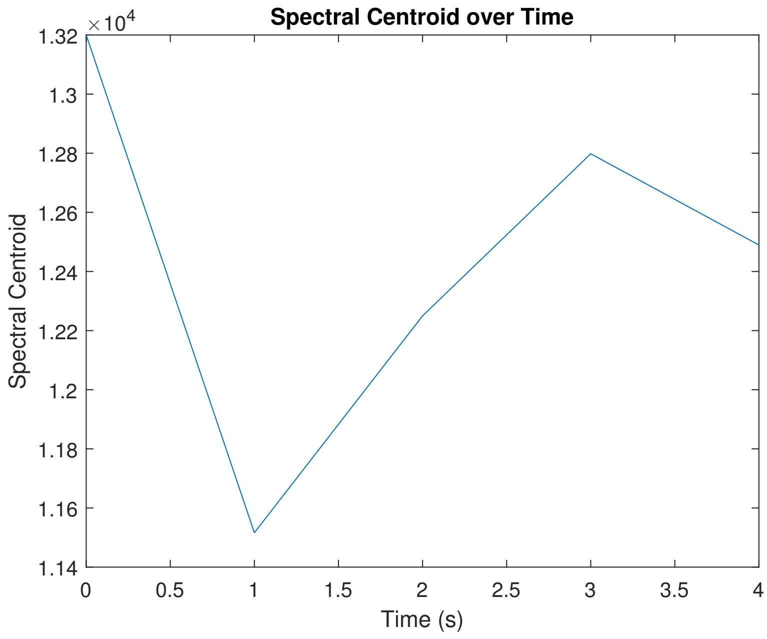


**Figure 2.1:** Spectral Centroid of 5 detected peaks in the signal.

Certainly, the spectral centroid values are within the same frequency range as the signal being analyzed. For example, since I analyzed speech signals within a range of 0 Hz to 48 kHz, then the spectral centroid values would fall within that range. The result, spectral centroid, should be within the frequency range of the signal and not exceed the Nyquist frequency, which is half the sampling rate.

In the following piece of code, the $freq_axis$ represents the frequency values corresponding to each bin of the power spectrum. The spectral centroid is then calculated by taking the weighted average of the frequency values, where the weights are the corresponding magnitudes from the power spectrum.

```
% Compute the frequency axis for the power spectrum
freq_axis = (0:(n_frame/2)) * (fs/n_frame);

% Compute the spectral centroid
spectral_centroid = sum(freq_axis * P1(:,j)) / sum(P1(:,j));
if spectral_centroid>0 && spectral_centroid<fs/2
    % add to structures
    str(n).detail(j).SCCC = spectral_centroid;
end
```

It is important to note that the calculation of the spectral centroid should be done on a frame-by-frame basis if we are working with frames of the signal. And so summing up the spectral centroid across all frames is not correct. To compute the spectral centroid for each frame, we would need to perform the above computation within a loop iterating over the frames. Also, we need to make sure that the dimensions of power spectrum P2 are appropriate for the calculation. That is the reason I added power spectrum P1 having the dimensions $(n\_frame/2 + 1)$ within the code snippet in 2.1.2 section. Spectral centroid could also represent the average frequency of the power spectrum, and its value is directly related to the frequency range of the signal.

### 2.1.4 Mel Filter Banks

MFCCs are computed from the mel-filterbank energies. The number of MFCC coefficients is typically determined by the output of the application. While both MFCC and SCCC can be used as features for audio analysis, they serve different purposes and are not inherently linked in terms of the number of coefficients. De facto, the number of coefficients is usually chosen based on the trade-off between computational complexity and the performance of the feature extraction algorithm. Filter banks made up of 26 vectors within the range (0.3-24)e3 Hertz based on the frequency vector. The lower value was chosen experimentally as a good value, whereas the upper is limited to half of the sample frequency. Now to convert both lower and upper frequencies to Mel scale from the linear frequency scale using the conversion equation in 1.5 or 1.7.

```
    lowf = 300;
    highf=fs/2;
    % computing band frequency to Mel scale
    mel_low =2595 *log10(1+(lowf /700));
    mel_high=2595 *log10(1+(highf/700));
```

Mel Filter Bank, involves applying a set of Mel filters to the power spectrum obtained from the FFT with the purpose to approximate the frequency response characteristics of the human hearing perception. According to the number of filters are created both Mel-scaled vector along with frequency vector, by which we verify whether start and end points of it are the required frequencies, i.e. lower and upper. To determine the number of Mel filters or filter bank channels to be used that depends on the required frequency resolution and the output's requirements. Each filter is defined by its center frequency in Mel scale and its bandwidth.

```
    nfilt=26;
    % creating the mel-scaled vector
    mVec = linspace(mel_low, mel_high, nfilt+1); %nfilt+1
    % computing frequencies of the Mel vector
    hVec = 700*(exp(mVec/1125)-1);
```

And later the Mel frequencies are converted back to Hertz scale using the inverse Mel-scale transformation, where the conversion maps each Mel frequency to its corresponding Hertz frequency using 1.8 equation, for $hVec$ is the frequency variable that is going to define the center of each filter.

Since there is no frequency resolution to put filters at the exact points calculated above, so is essential to round those frequencies to the nearest FFT bin. This process has no impact on the accuracy of features. Rounding the frequencies to the nearest FFT bin is necessary to align the filter bank channels with the power spectrum output and ensure consistency in the analysis. The FFT bins (or bins resolution) represent discrete frequency points in the spectrum, because the FFT operates on a discrete set of frequency bins determined by the sampling rate and the FFT size. The frequency resolution of each bin is determined by the sampling rate divided by the FFT size. And also the very Mel filters, which are designed based on the Mel scale, need to align with the FFT bins to accurately capture the energy distribution across different frequency regions. Rounding the filter center frequencies to the nearest FFT bin ensures that each filter aligns with the corresponding frequency bin in the FFT output, and avoiding any mismatch or misalignment that may occur due to fractional frequency values. This alignment is essential for accurate representation of the spectral energy distribution.

```matlab
nfft = 512; % FFT point-size.
% convert frequencies to nearest bins
for i=1:nfilt+1 %nfilt+1
    f(i) = floor((nfft+1)*hVec(i)/fs);
end
```

Rounding the frequencies helps maintain consistency and coherence in the analysis. It guarantees the filter bank energies correspond to the suitable frequency regions and allows for meaningful comparisons across different signals or frames, assuring accurate representation of the energy distribution across different frequency regions

To build the Mel filter bank by making the triangular-shaped filters, where each one is centered at a specific Mel frequency and extends to adjacent Mel frequencies. The shape of each triangular filter is defined by its center frequency and bandwidth. The step to get the desired filters is to use the following formula $h$ [Lyo09], from row 3 of the next piece of code for calculating:

$$h(m,k) = \begin{cases} 0 & k < f(m-1) \\ \dfrac{k - f(m-1)}{f(m) - f(m-1)} & f(m-1) \leq k \leq f(m) \\ \dfrac{k(m+1) - k}{f(m+1) - f(m)} & f(m) \leq k \leq f(m+1) \\ 0 & k > f(m+1) \end{cases} \tag{2.2}$$

where parameter $m$ represents filter number, $k$ the frequency on x-axes for future plotting, and $f$ is the precise frequency of each filter taken from 1.6. The first filter bank starts at the first point, reach its peak at the second point, then return to zero at the 3rd point. The second filter bank will start at the 2nd point, reach its max at the 3rd, then be zero at the 4th etc.

```
% filterbank has 26=m vectors of length fs/2=k
for m=2:nfilt
    for k=1:fs/2
        h(m-1,k) = formula(k,hVec,m);
    end
end
```

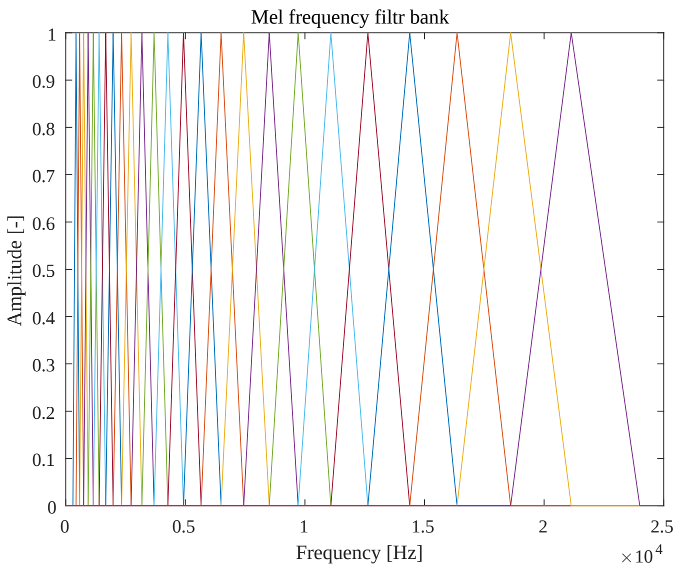


**Figure 2.2:** Filter bank on a Mel-Scale.

As has been said in subsection 1.3.3 how Mel-scale imitates the non-linear human hearing by more distinguishing lower frequencies and less the higher ones. The dimensions of filters is $26x24000$, where 24000 is the upper value on the frequency scale, and 26 represents the number of filters -1, because of the resulting cepstral coefficients for Automatic Speech Recognition, that start from number 2 instead of 1. Otherwise the $h-$formula will not work, for it is designed that particular way.

### 2.1.5 Mel Coefficients

The last step to acquire Mel Coefficients, we need to apply each Mel filter to the power spectrum obtained from the FFT. The application involves element-wise multiplication between the power spectrum and the filter's frequency response. This process calculates the energy within each filter's frequency band for later using the Discrete Cosine Transform (DCT).

```
1      % final cepstral coefficients.
2      for k = 1:length(str(n).detail)
3          for j = 1:n_frame/2+1
4              for i = 1:nfilt-1
5                  num(i,:) = P1(j,k).*h(i,:);
6              end
7              energy_log_M(j,i) = log(sum(num(i,:)));
8              dct_energy_M(j,:) = dct(energy_log_M(j,:));
9          end
10         % add to structures
11         str(n).detail(k).MFCC=dct_energy_M(1,:);
12     end
```

The logarithm operation at row 8 is used to approximate the logarithmic nature of human perception of sound (the natural logarithm is used instead of other bases). Taking the logarithm compresses the dynamic range of the filter bank energies, giving the Mel coefficients more perceptually meaningful, for Human hearing is more sensitive to relative changes in loudness than absolute values. The dynamic compression is needed, because the energy values obtained from the Mel triangular filter bank might vary over a wide range, and so the natural logarithm, the dynamic range of the filter bank energies is compressed. This reduces the impact of big energy variations and helps to normalize the representation across different signal conditions.

And also the natural logarithm in this part help weighting the low energy sections, for these sections of the frequency spectrum in many audio processing or speech recognition contain less relevant information. What the logarithm does, is emphasizing sections with higher energy and at the same time downplaying the importance of sections with lower energy. This can help improve the discriminative power of the MFCCs by focusing on the more perceptually salient spectral features. Besides that natural logarithm improves stability and numerical precision of most cepstral coefficients, because when dealing with very small energy values the logarithm helps to decrease all effects of numerical errors and enhance, which leads to better validity of the cepstral coefficient calculation.

As the last step to achieve any of the cepstral coefficients mention, whether MFCC, IMFCC, LFCC or GTCC the discrete cosine transform is crucial for de-correlating the coefficients, to get a compressed representation of the filter banks, to reduce the redundancy and captures the most prominent spectral features more effectively. Because the energies obtained by filter bank are often correlated for neighboring filters tend to capture energy from similar frequency areas around. And so DCT transforms the correlated coefficients into a set of de-correlated cepstral coefficients. What also happens along with the de-correlation is energy concentratio, when DCT concentrate the energy in a few lower order coefficients while spreading it out in higher order coefficients. The lower order DCT coefficients seize the outstanding spectral elements, while the higher order ones seize lighter spectral items. And because noise in audio signals affects primarily higher order coefficients, by discarding those DCT coefficients we reduce the influence of noise in the resulting cepstral coefficient significantly. Which means that DCT can help improve the robustness of the MFCC features to noise.

Besides de-correlation, energy concentration and features' robustness, the DCT is used for a purpose called Dimensionality Reduction, which reduces the dimensionality of the feature representation that happens in case the output of the Mel filter bank yields a large number of filter bank energies, which might contain redundant information. The DCT transforms that energy into a smaller set of coefficients that catch the most critical spectral information. All that retaining of the less discriminative information then cepstral coefficients representation becomes more efficient and compressed.

### 2.1.6 Spectral Entropy

Spectral entropy is a statistical measure that characterizes the level of uncertainty or randomness of the spectral amplitudes or power distribution in a signal. It indicates how the energy is been distributed as one across different frequency components.

Similarly to the Spectral Centroid, I proceed from the same article: An Entropy-Based Architecture for Detection of Sepsis in New-2 born Cry Diagnostic Systems [ZKT22], section 2.3.2. Spectral Entropy Cepstral Coefficients (SENCC). The authors describe spectral entropy as follows: "Evaluates the signal's energy distribution uniformity. This measure is an indicator of the complexity of the signal. It can also be employed to capture the peakiness in a signal. In order to compute the spectral entropy, the spectrum is written in terms of a Prob-285 ability Mass Function (PMF)-like function:

$$C_i = \frac{X_i}{\sum\limits_{i=1}^{N} X_i} \tag{2.3}$$

where $X_i$ is the energy of i-th frequency component of the spectrum."

Later on in the article the authors write: "After going through the Mel-filter banks, the spectral entropy of each sub-band is evaluated." From which I concluded that the Spectral Entropy Cepstral Coefficients (SENCC) are not dependent on MFCC because they are two different feature extraction techniques used in speech and audio processing. While both methods extract useful information from the spectral domain, they have distinct purposes and calculations. MFCC focuses on capturing the mel-frequency cepstral characteristics of the audio signal, which are based on the Mel filter bank analysis, natural logarithm and discrete cosine transform coefficients. So, it aims to represent the spectral shape of the signal with emphasis on perceptual properties. On the other hand, SENCC is used for capturing the information related to spectral complexity and variability, for it measures the entropy of the power spectrum of the audio signal by assessing the level of disorder/accidental character/randomness in the distribution of spectral energy, for it provides information about the distribution of spectral energy across different frequency components in a signal's spectrum.

The computational process of spectral entropy includes estimating the spectral power of a signal that is performed by executing a spectral analysis using Fast Fourier Transform. Spectral entropy values in range 0-1 (including both), where 0 indicates a completely deterministic or pure sinusoidal signal with all energy concentrated at a single frequency, and 1 indicates maximum disorder, where the energy is evenly spread across the entire frequency range. Maximum entropy is at value 1, see figure 2.3 below.

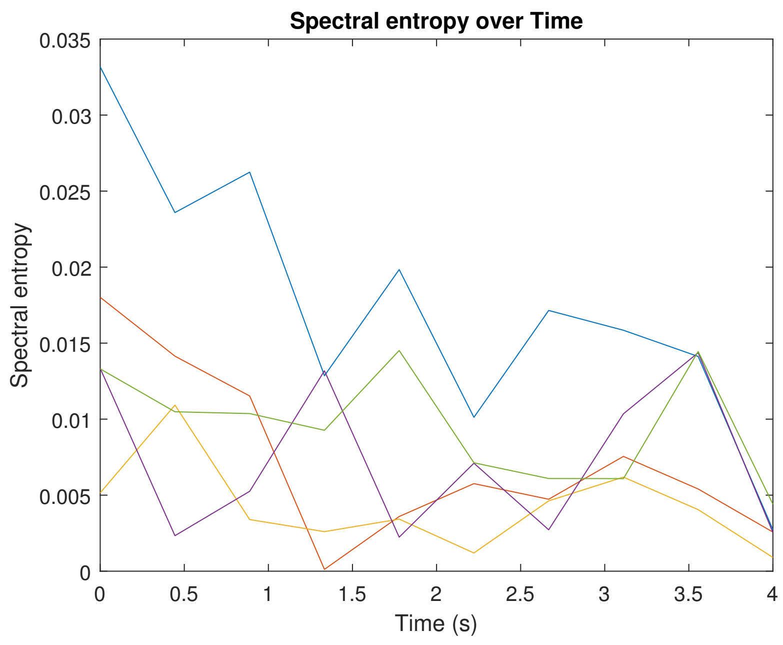


**Figure 2.3:** Spectral entropy of 5 detected peaks in the signal.

In audio processing and speech recognition, spectral entropy is used to characterize the spectral properties of audio, to changes in the spectral content or to detect transitions in order to distinguish between different types of sounds or speech parts.

But I used spectral entropy for a bit different reason, as a feature for subsequent machine learning as a pattern recognition, because next to spectral centroid and Mel frequency cepstral coefficients spectral entropy stands for complementary feature in capturing learning aspects with spectral characteristics in a signal.

The next piece of code describe the applied process for extraction the SENCC.

```
% compute spectral entropy of each frame
    for j = 1:length(str(n).detail)
        entropy = zeros(n_frame/2+1,1);

        for k=1:n_frame/2+1
            Xi = energy(k,j)./sum(energy(k,:));
            entropy(k) = -sum(Xi.*log2(Xi));
        end
        % apply DCT to spectral entropy sequence
        SENCC = dct(entropy);
        num_SENCC = size(SENCC,1)-1;

        % add to structures
        str(n).detail(j).SENCC = SENCC(1:end);
    end
```

In the final analysis the number of spectral entropy coefficients (SENCC) was chosen half of the number of frames +1.

## 2.2 Inverse Mel Frequency Cepstral Coefficients

To build the inverse Mel filter bank, by making reverse triangular-shaped filters, where each one is centered at a specific Mel frequency and extends to adjacent Mel frequencies. The shape of each triangular filter is defined by its center frequency and bandwidth, as much as with the classical Mel filter banks. The step to get the desired filters is the use the same formula 2.2, but with and inverse $h$. In particular, using Matlab the $flip()$ function was applied. I specifically used the function with the $dim$ parameter to reverses the elements in each row.[1]

The rest of the code is similar to the MFCC features extraction, with one difference compared to 2.1.5 row 5, where now is necessary to multiply the power spectrum $P1$ with the fliped $h$, precisely $hinv = flip(h, 2)$.

The final product is in figure 2.4, the middle filter bank. Please note how Matlab in this case plotted the first (purple) filter, compared to 1.9. I really do not know the reason, though I implemented and tried more command to have them consistent.

## 2.3 Linear Frequency Cepstral Coefficients

```
    %% linear filtr
    linVec = linspace(lowf, highf, nfilt+1);
    for m=2:nfilt
```

[1]https://www.mathworks.com/help/matlab/ref/flip.html

```
        for k=1:fs/2
            hlin(m-1,k) = formula(k,linVec,m);
        end
    end
```

In the previous code snippet we see how the process to get linear filters is to start with a linear vector and not the Mel-scale one. The boundaries (upper and lower frequencies) have to be kept. The used formula remains the same, with the difference of the second parameter, which is the linear vector for this scenario. And the rest of code is the same to MFCC extraction in 2.1.5 with the difference around the multiplication the power spectrum $P1$, where $hlin$ has to be applied.

The final product is in figure 2.4, the lower filter bank. And again please note how Matlab in this case did not plotted the first (purple) filter, as much as in 1.10.

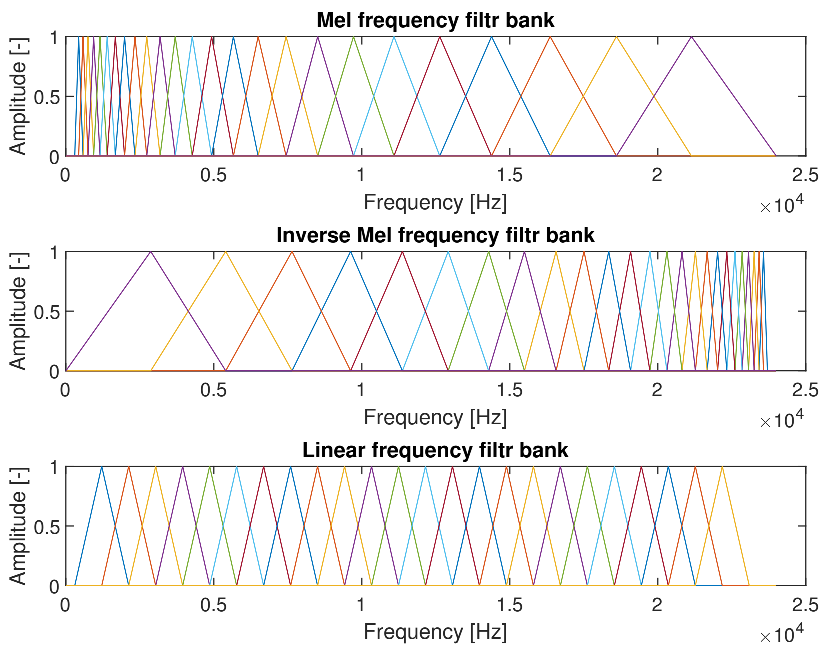


**Figure 2.4:** Mel bands with different scales.

## 2.4 Gammatone Filter Bank

Gammatone cepstral coefficients (GTCC) are another alternative to MFCC that utilizes gammatone filter bank instead of the traditional triangular Mel filter bank. GTCC aims to capture the spectral properties of the audio signal more accurately, particularly at lower frequencies.

The approach I used was through the already obtained MFCC:

```
    % gtcc
    numCoeffsRows = length(str(n).detail);

    % Compute center frequencies of triangular Mel filterbanks in Hz
    centerFreqs = hVec;

    % Compute Gammatone filterbank outputs for each MFCC
    gammWeights = zeros(numCoeffsRows, nfilt-1);
    for i = 2:nfilt
        gammWeights(:,i-1) = gammatoneFilter(mfccs(:,i-1), centerFreqs(i), 1);
    end

    gtcc = dct(log(gammWeights));

    % Normalize GTCCs to get unit variance
    gtcc = bsxfun(@minus, gtcc, mean(gtcc));
    gtcc = bsxfun(@rdivide, gtcc, std(gtcc));

function y = gammatoneFilter(x, f, q)
    % filter coefficients
    t = 1:length(x);
    a = 1/(2*pi*f*q);
    b = 1.019*f/q;
    c = 2*pi*f;
    g = a*t .* exp(-b*t) .* sin(c*t);

    % Apply filter to the signal
    y = filter(g, sum(g), x);
end
```

where $numCoeffsRows$ represents the number of detected peaks in the signal, from which follow the column number and that is the $nfilt - 1$ equal to the number of MFCC for each peak. The gammatoneFilter() function operate with three parameters: $x$ stands for the individual MFCC in particular columns, $f$ the center/carrier frequencies that define each filter derived from the Mel-scale frequencies, since the gammatone filter banks operate at the same frequencies like Mel, because both were originally designed to approximate the non-linear nature of human perception of sound.

Then I applied the Matlab 1-D digital filter function, that has at least 3 parameters: The Gammatone filter coefficient $g$ is computed based on the input parameters, and they define the impulse response of the filter. The $sum(g)$ is used to normalize the filter coefficients, ensuring that the total response energy is preserved. It represents the initial conditions of the filter and specifies the initial state of the filter's internal delay line. By using $sum(g)$

as the initial conditions, the filter function starts with an initial state that corresponds to the filter's impulse response, to ensure the filter's output is correctly computed by convolving the input signal with the filter coefficients.

Row 13 in the snippet code: The gammatone filter outputs are passed through the Discrete Cosine Transform (DCT) and logarithmic transformation to compute the Gammatone Cepstral Coefficients (GTCCs): The natural logarithmic operation is applied to the Gammatone filter bank outputs to mimic the non-linear perception of loudness in the natural human auditory system. To compresses the dynamic range of the filter bank outputs, emphasizing the lower energy components and reducing the impact of higher energy ones. The DCT is used to decorrelate the logarithmically transformed filter bank outputs, to reduce the redundancy and capturing the most relevant information from the filter bank outputs.

The GTCC normalizing as the final step ensures that the GTCC are less sensitive to the energy level variations across different utterances. It helps in achieving better robustness and comparability of the GTCC features.

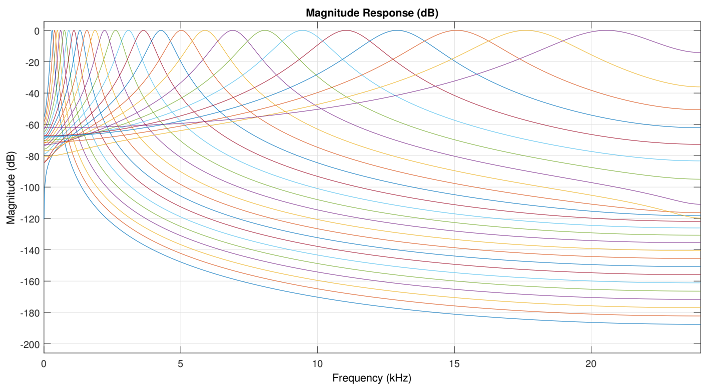


**Figure 2.5:** Filter bank on a Mel-Scale with Gamma curving in frequency domain.

The amplitudes of each filter will be different and greater due to the filter order occurring in the exponent of time, where each magnitude is greater with the increasing filter order, so much in time domain with the amplitude, compare to 2.6.

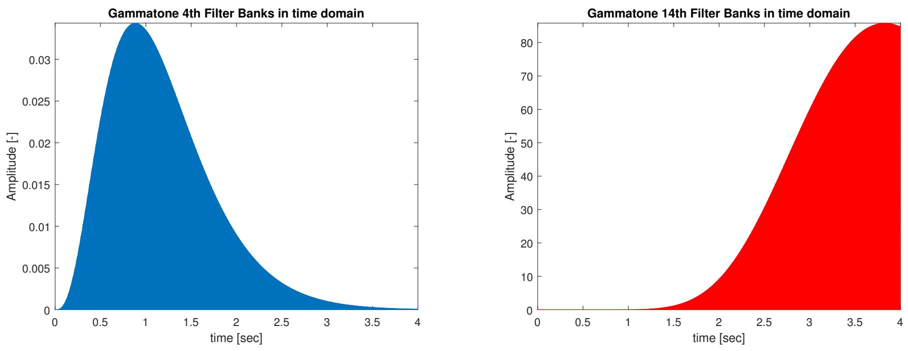


**Figure 2.6:** 2 orders of Gammatone filter bank in time domain.

# Chapter 3
# Classification

Now after finishing all the necessary details regarding signal analysis, I received a lot of data in *.wav* format and others in .mat structures from my supervisor, which I had to convert each file from the mat structure to a *.wav* file using the Matlab audiowrite function.

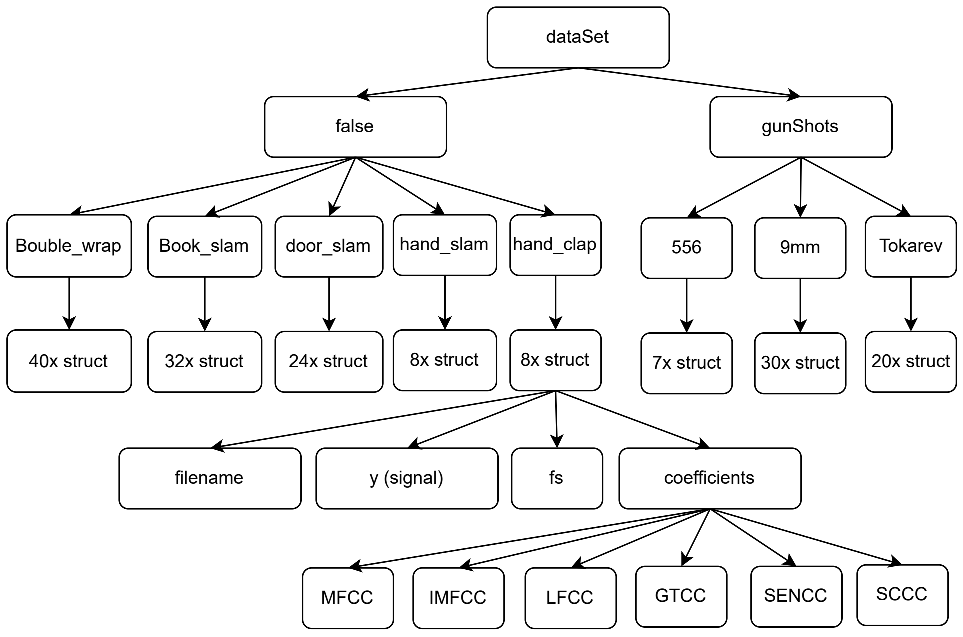


**Figure 3.1:** The final structure's shape used before classification.

Note: The software's final version before classification is designed in an adaptable and flexible way that by loading the dataSet structure with false alarms and gun shots, any *.wav* file could be read, added to structure, its features get computed, and in the end be added to the dataSet structure to the right place according to its features.

Before training the Convolution Neural Network (CNN), it's important to pre-process the data as per the requirements of the final model. This process may involve scaling, reshaping, normalization, and other pre-processing steps necessary to enhance the training process.

## 3.1 Data collection

The fist step before classification starts is that all features from all files have to be gathered in one array, according to their type. For example, one array collecting only MFCC from gun shots, another MFCC array assembling all MFCCs from false alarms, an so forth. I ended up with 8 array, 4 from each substructure representing their cepstral coefficients. The preparatory structure to classification has the following shape 3.2:

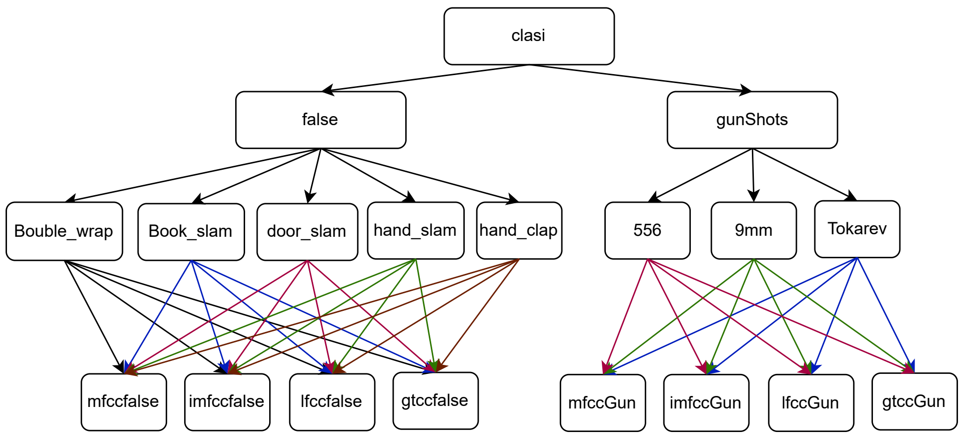


**Figure 3.2:** The modified structure for classification.

As a code snippet I provide the following:

```
% gather Guns MFCC
mfcc = struct('MFCC', {0});
% initialize array size, to ensure columns size
mfccGun = zeros(1,25);

cnt=zeros(1,length(dataSet.gunShots));
i=1;
mfccGun(1,:)=[];
for k=1:length(dataSet.gunShots)
    for j=1:length(dataSet.gunShots(k).details)
        cnt(k)=cnt(k)+size(dataSet.gunShots(k).details(j).
coefficients.MFCC,1);
        mfcc(i).MFCC=dataSet.gunShots(k).details(j).
coefficients.MFCC;
        mfccGun = [mfccGun;vertcat(mfcc(i).MFCC)];
        i=i+1;
    end
end
```

The counter, e.i. *cnt* was added to find out the exact number of rows collected from each gun shot type. The numbers are very crucial for the CNN, which is going to take the individual rows of cepstral coefficients and assign to them their exact type later on using categorical function. The same process was repeated for all types within gun shots or false alarms, in order to gather all cepstral coefficients from dataSet structure. The last step was to create a new structure for the purpose of saving memory for future starts, where only cepstral coefficients gather in arrays without nested structures.

## 3.2 Data partition

Now working with the new structure *clasi*, I had to split the data into individual groups according to the gun shots or false alarms types with the right sizes using the *cnt* or *cntf* (for false alarms) variables, respectively. Below the snippet is just for the gun shots MFCC for illustration.

```
    mfccGun=clasi.mfccGun;

    % Splitting the data into groups' types
    fiveMF=mfccGun(1:cnt(1),:);
    nineMF=mfccGun(cnt(1)+1:cnt(1)+cnt(2),:);
    tokaMF=mfccGun(cnt(1)+cnt(2)+1:end,:);
```

All the sizes are 1 row vectors, next to each other, because by doing one column each label next to each other, Matlab will throw an error that dimensions of arrays are not consistent, for not all columns are of same size.

Now to apply the cvpartition function to create a partitioned data set for cross-validation or other evaluation strategies. It helps in dividing a dataset into subsets for tasks like training, validation, and testing [1]. The first parameter should be equal to the number of rows in the MFCC matrix. It represents the size of the data that we want to partition, for it is the total number of observations in the dataset.

The second parameter is the method used for partitioning the data. It specifies how the data will be divided into subsets for purposes such as cross-validation or holdout validation. Matlab offers more parameters, such as 'Stratified', 'LeaveOut'. I worked with 'KFold' and 'Holdout'. 'KFold' performs K-fold cross-validation, where the data is divided into K equally sized subsets, where each subset is used as a validation set once, while the remaining subsets are used for training. While 'Holdout' just creates a simple random partition of the data into two subsets, for holdout validation. In both case we need to specify the desired proportion of the data to be held out as a validation set using the third parameter in 'Holdout' or specify the desired value of K as the third parameter in 'KFold'.

For our specific case in here it is necessary to compute the sizes for training and validation sets for each group, and not just in general. Because from each gun shot or false alarm we want to keep an amount for validation on the basis of the desired proportion.

[1]https://www.mathworks.com/help/stats/cvpartition.html

```
trainProp = 0.7;
validProp = 0.3;

% sizes for training and validation sets for each group
numTrainSamplesFiveMFCC = round(trainProp * cnt(1));
numTrainSamplesNineMFCC = round(trainProp * cnt(2));
numTrainSamplesTokaMFCC = round(trainProp * cnt(3));

% make cvpartition objects for each group
cvfiveMF = cvpartition(cnt(1), 'Holdout', validProp);
cvnineMF = cvpartition(cnt(2), 'Holdout', validProp);
cvtokaMF = cvpartition(cnt(3), 'Holdout', validProp);
```

## 3.3 Training and validation sets

Now, after the partitioning, I had to split the data into training and validation sets. By defining the data proportion, Matlab itself splits the whole set into *trainingData* and *validationData* according to set proportion. Before that I had to generate the training and validation indices for each group, based on the partition data for cross-validation from the previous code snippet. Then use the indices to extract the training and validation sets for each type. The last step is to combine the training and validation sets for all groups. In the following snippet I illustrate only the steps regarding one type of gunshots to save space for text. The same steps are required for the rest of gunshots' types, or all types in false alarms.

```
% Generate the train and valid indices for each group
trainIdxFiveMFCC = training(cvfiveMF);
validIdxFiveMFCC = test(cvfiveMF);

% Use indices to extract train and valid sets for each
trainDataFiveMFCC = fiveMF(trainIdxFiveMFCC, :);
validDataFiveMFCC = fiveMF(validIdxFiveMFCC, :);

% Combine the training and validation sets for all groups
trainingData = [trainDataFiveMFCC; trainDatanineMFCC;
trainDataTokaMFCC];
validationData = [validDataFiveMFCC; validDataNineMFCC;
validDataTokaMFCC];
```

## 3.4 Labels

Now after knowing the sizes of each gun shot and false alarm we set the labels. Labels are necessary for convolutional neural networks because they represent the target values associated with each input sample. In my classification problem, labels indicate the category to which each input sample belongs. We can choose to represent these labels either as a categorical array or as a numerical array.

I run across one-hot encoding [2]: If I understood correctly, then in one-hot, each category is represented by a binary vector where all elements are zero except for the element corresponding to the class label, which is set to one (one-hot). For example, if we have three categories (A, B, C), one-hot encoding would represent them as [1 0 0], [0 1 0], and [0 0 1]. On the other hand, categorical encoding assigns a unique numerical value to each category. For example, in a same scenario (A, B, C), categorical encoding might represent them just as 1, 2, and 3. This encoding is useful when the numerical order of the categories does not carry any meaningful information.

In the end, I decided for categorical encoding over the one-hot, because one-hot encoding will result in a label array with a shape of (number of samples, number of categories), where each sample is represented by a binary vector, whereas categorical encoding will result only in a label array with a shape of (number of samples, 1), where each sample is represented by a numerical label. I made that decision after implementing and trying the one-hot encoding, where i realized 2 disadvantages: The first is better representation that suits my specific problem and the requirements of CNN model was the categorical one, for their clarity, since they are obvious by the first look. And the other disadvantage was a Matlab complication regarding matrices size. (Later, I will show both implementations.)

Immediately after it, I had to extract the labels corresponding to the samples included in the training set defined by training(cv), or in the test set defined by test(cv).

```
    % Categorical labels for the training and validation sets
    trainingLabels = categorical([ ...
    repmat({'fiveMF'}, size(trainDataFiveMFCC, 1), 1);
    repmat({'nineMF'}, size(trainDatanineMFCC, 1), 1);
    repmat({'tokaMF'}, size(trainDataTokaMFCC, 1), 1)]);

    validationLabels = categorical([ ...
    repmat({'fiveMF'}, size(validDataFiveMFCC, 1), 1);
    repmat({'nineMF'}, size(validDataNineMFCC, 1), 1);
    repmat({'tokaMF'}, size(validDataTokaMFCC, 1), 1)]);
```

[2]https://www.mathworks.com/help/deeplearning/ref/onehotencode.html

In the code snippet above is the Categorical assigning, whereas in the below snippet is the one-hot encoding. Where I implemented it myself in order to test the validation behind it. I used Matlab's Unique function to get the unique values in array [3], or the precise number of different sets there.

```
% one-hot encoding
uniqueLabels = unique(trainingLabels);
numClasses = numel(uniqueLabels);

trainingLabels = zeros(size(trainingLabels,1), numClasses);
trainingLabels(1:numTrainSamplesFiveMFCC, 1) = 1; %'fiveMF';
trainingLabels(1+numTrainSamplesFiveMFCC:
    numTrainSamplesFiveMFCC + numTrainSamplesNineMFCC, 2) = 1;
    %'nineMF';
trainingLabels(1+numTrainSamplesFiveMFCC+
    numTrainSamplesNineMFCC:end, 3) = 1;%'tokaMF';

trainingLabels = categorical(trainingLabels);
```

The sizes of trainingData and trainingLabels have to match: If trainingData size is (N, F), where N is the number of samples and D is the number of features. The trainingLabels size is supposed to be (N, 1). A confirmation is necessary that there are no missing elements or NaN values whether in trainingData and trainingLabels. Similarly, ensure that the dimensions of validationData and validationLabels are compatible. The number of rows in validationData should be equal to the number of elements in validationLabels. That's the reason, as mentioned above regarding my choice for the categorical sorting over the one-hot encoding.

## 3.5 Layers

By now, all the necessary data components are finished, just to define the architecture by setting the layers and options to train the neural network. The following figure 3.3 describes the layers applied for CNN architecture.

```
X = [fiveMF;nineMF;tokaMF];
%Define the architecture of the CNN
layers = [
imageInputLayer([size(X, 1), size(X, 2),1], 'Name', 'input')
%%%
convolution2dLayer(3, 16, 'Padding', 'same', 'Name', 'conv1')
reluLayer('Name', 'relu1')
maxPooling2dLayer([3, 1], 'Stride', [2, 1],'Name','maxpool1')
%%%
convolution2dLayer(3, 32, 'Padding', 'same', 'Name', 'conv2')
reluLayer('Name', 'relu2')
```

[3]https://www.mathworks.com/help/matlab/ref/double.unique.html

```
maxPooling2dLayer([3, 1], 'Stride', [2, 1],'Name','maxpool2')
%%%%
fullyConnectedLayer(64, 'Name', 'fc1')
reluLayer('Name', 'relu3')
fullyConnectedLayer(numClasses, 'Name', 'fc2')
softmaxLayer('Name', 'softmax')
classificationLayer('Name', 'classification')
];
```

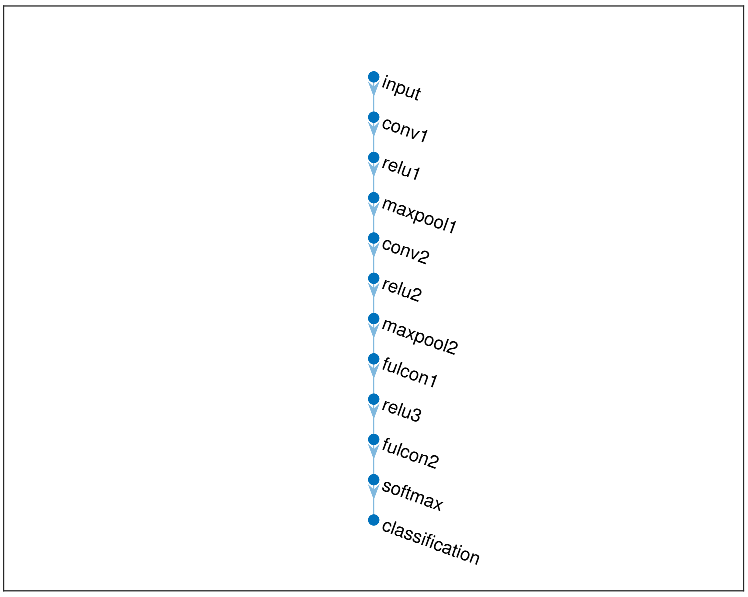


**Figure 3.3:** The individual layers used in the Convolution Neural Network.

As the first layer that defines the input to the network I used *imageInputLayer*. It particularizes the size of the input MFCC features in this case (images in general) using their dimensions. The third parameter in the array is the number of channels, referring to the depth in an image, or to the number of color channels. For grayscale images, there is one channel representing the intensity values. For colorful images, there are 3 color channels: red, green, and blue. In the MFCC data I used just single-channel input, since it is unlikely for MFCCs to have multiple channels. The 'Name' parameter assigns the name 'input' to this layer.

I also tried 'sequenceInputLayer' [4] in hope of getting a better result, because I expected, based on the label, that 'imageInputLayer' works only for images and here I wanted to elaborate with audio features. Fact of the matter is, 'imageInputLayer' works with dimensions, without regard to the original data

[4] https://www.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.sequenceinputlayer.html

source.

The second applied layer is *convolution2dLayer*, which performs 2D convolution on the input feature maps by applying a set of learnable filters to the input feature maps, to produce a set of output feature maps. Each filter performs a convolution operation by sliding across the input feature maps, computing element-wise multiplications and summations. The applied layer has 16 filters of size 3x3, where the size determines the receptive field of the filters, influencing the patterns this layer can detect. The 'Padding' parameter controls the treatment of the borders of the input feature maps during convolution, and by setting it to 'same', the layer does not add padding to the borders, allowing the output feature maps to have the same spatial dimensions as the input, to preserve the spatial dimensions of the input.

Follows the *reluLayer*, i.e. Rectified Linear Unit. It introduces non-linearity into the network, allowing the model to learn complex patterns and relationships in the data. It applies a ReLU activation function element-wise to the input tensor or feature map, which is to the output of the previous layer. That activation is beneficial in deep learning, because it provides thinness in the network, as it sets negative values to zero, leading to more efficient and sparse representations.

$$f(x) = max(0, x) \tag{3.1}$$

where x is the input, and by comparing it to zero gives the maximum value to the output.

*maxPooling2dLayer* performs a 2D max pooling on the input feature maps, which divides the them into non-overlapping rectangular sections called pooling regions, where within each one the maximum value is selected and passed to the next layer, while the other values are thrown away. This operation effectively down-samples the feature maps by keeping the most prominent features and discarding less important ones. Parameter [3, 1] represents a pooling region with a height of 3 and a width of 1. While the [2, 1] Stride specifies the amount by which the pooling regions are shifted vertically (2 units) and horizontally (1 unit).

Dense layer or *fullyConnectedLayer* that connects every neuron in the previous layer to the neurons in the current layer. It does a linear transformation on the input data by multiplying them with a weight matrix and adding a bias vector, where the size of the weight matrix determines the number of neurons in the current layer, and the size of the bias vector is equal to the number of neurons in the current layer. This one has 64 neurons. After the linear transformation in the first fully connected layer, an activation function is applied to introduce non-linearity into the network, such as a reluLayer follows the first fully connected layer, which applies the ReLU activation function to the output of the fully connected layer. The number of neurons in the second fully connected layer corresponds to the number of classes in the classification problem.

*softmaxLayer* converts the network's raw outputs into understandable probabilities. These probabilities indicate the likelihood of each class being the correct prediction using a cross-entropy loss function as and suitable one. The output of the fully connected layer (64 neurons) is fed into a softmax layer to obtain the final class probabilities or predictions. This layer is used for multi-class classification problems, because it applies the softmax activation function to the input and produces a probability distribution over the classes by applying exponentiation function to each element of the input and afterwards normalizes the values by dividing them by the sum of all exponentiated values. This normalization ensures that the output values lie in the range of [0, 1] and sum up to 1, representing probabilities.

$$softmax(x) = exp(x)/sum(exp(x)) \tag{3.2}$$

where x is the input vector.

*classificationLayer* is the final layer of the network to compute the cross-entropy loss between the predicted class probabilities and the true class labels, and perform classification based on the predicted probabilities. The output is a vector of class probabilities, where each element in it represents the probability of the corresponding class.

Adding a connection between layers in a neural network is an essential step to define the flow of information during forward and backward propagation, for each connection represents a track through which data is passed from one layer to another. Adding connections between layers ensures that the network is properly connected and can effectively spread information during training. Adding a connection that does not already exist is necessary to establish an architectural configuration of the neural network. The connections determine the flow of information and the interaction between layers, affecting the network's ability to learn from the training data.

```
% Convert Connections table to cell array
connections = table2cell(lgraph.Connections);

% Check if connection already exists before adding it
connectionExists = false;
for i = 1:size(connections, 1)
    if strcmp(connections{i, 2}, 'maxpool1') && strcmp(
connections{i, 1}, 'relu1')
        connectionExists = true;
        break;
    end
end

% Add connection if it does not exist
if ~connectionExists
    lgraph = connectLayers(lgraph, 'relu1', 'maxpool1');
end
```

By checking if a connection already exist, the code avoids adding duplicate connections, which could lead to throw errors in the network.

But before that I had to convert the connections between layers in the layer graph into a cell array, to simplify the subsequent operations for flexible handling, and later checking of the connections. The code then could be easily accessed over the individual connections using array indexing in the for loop that iterates through each connection in the cell array to check whether a specific connection does already exists. The strcmp function compare strings of the source and destination layers of each connection. So, by converting the connections into a cell array allowed me for more easier and efficient operations, such as checking if a specific connection already exists or modifying the connections between layers.

## 3.6 Training options

The training options are necessary to specify various settings for teaching the neural network. It allows to define various aspects of the training process, and how the whole process should be performed with specific arrangement.

```matlab
% Set training options
options = trainingOptions('adam', ...
    'MiniBatchSize', 32, ...
    'MaxEpochs', 20, ...
    'Shuffle', 'never', ...
    'ValidationData', {validationData, validationLabels}, ...
    'ValidationFrequency', 1, ...
    'Verbose', false, ...
    'Plots', 'training-progress');
```

The first parameter specifies the optimization algorithm to be used during training. *adam*, i.e. Adaptive Moment Estimation was chosen over stochastic gradient descent (SGD), though it requires additional memory usage, and also SGD's noise prevents overfitting and improving the model's ability to generalize to new unseen data.

Adam adapts the learning rate for each parameter individually based on the historical gradients (more memory). Because my training is not expected to be any large scale model, so the memory usage was not a concern. Adam maintains a separate learning rate for each parameter and updates it dynamically throughout the training process. And because Adam's adaptive learning rate allows him to handle different parameters updates effectively, it provides a better performance. And also Adam anticipates a bias correction during the early iterations of training, which helps him mitigate the prejudices towards zero that can occur in the estimates of the first moments of the gradients. This correction allows him to prepare more accurate updates to the network's parameters, especially at the beginning of training.

*mini-batch* determines the number of samples that are fed into the network together at the same time before the gradient update is performed. Size of 32 means that during each iteration of the training algorithm, 32 samples will be processed at the same time, and the gradient update will be based on these 32 samples. The choice of its size involves a tradeoff between computational efficiency, memory usage, and generalization performance. Smaller mini-batch sizes may can provide better generalization, but lead to longer training times.

*MaxEpochs* defines the maximum number of training epochs, where an epoch refers to a complete pass through the entire training file in the training process. In each epoch, the network goes through all the training samples, updates the weights and biases based on the calculated gradients, and modifies the model parameters to minimize the loss function. Once the specified number of epochs (20) is reached, the training process stops even if the confluence criteria will not be met yet. Training a CNN with a large number of epochs can be time consuming.

*Shuffle* is used to specify whether and how the training data should be shuffled before each epoch. Shuffling the data helps in randomizing the order of samples and reduces any bias that might be introduced due to the order of the data. The *never* parameter specifies that shuffling should not be performed at any epoch. Compared to the other limit that is $every - epoch$ which shuffles before every epoch and by that increases the randomness and reduces the possibility of the model learning any systematic teaching, because it would make the model seeing the data in a different order in each epoch.

The *ValidationData* option allows to specify the validation data to be used during the training. The validation data is a separate dataset that is used to monitor the model's performance and generalization of the network during training. It consists of validationData (the input data) and validationLabels (the corresponding ground truth labels). This is the step, where the validationData and validationLabels must match, where each row of the validationData matrix corresponds to the correct label in the validationLabels vector, in order that the network's performance is correctly evaluated during training.

The *ValidationFrequency* option particularizes the frequency at which the validation accuracy is computed, to control how often the validation data is evaluated throughout the training process. The larger validation frequency is, the less frequently is performed, resulting in fewer evaluations on the validation set and less checkpoints to monitor the model's performance during training. Yet a advantage to it, is the speed of the training process, for it might boom as less computational resources are assigned to validation evaluations. In the case of parameter equals 1, the validation frequency is computed every training iteration (mini-batches).

The *Verbose* option determines whether or not to display the training progress and other information during training, and by setting it to false value means that training process will run silently without any output messages, such as the current epoch, mini-batch progress, or training metrics. And also having the Verbose false reduces the computational overhead associated with console output. Other options are mini, normal, or detailed that displays the same information as normal choice, but additionally shows the training and validation loss with accuracy for each mini-batch iteration.

Setting the *Plots* option to training-progress shows the training and validation accuracies as the training progresses with a visual representation of the model's performance. These plots give a visual feedback on how the network is learning and how the performance metrics change over time. By visualizing the training progress, one may acquire a better understanding of how the network is learning, identify patterns in the loss and accuracy, and detect potential issues such as overfitting or insufficient learning on the other hand.

## 3.7 Training network

The last step I implemented before training the network was to check whether any of the trainingData NaN (Not a Number), in order to avoid any error in the computation. If it does find any, sets it to 0.

```
    trainingData(isnan(trainingData)) = 0;

    % Train the CNN model
    numObservationsData = size(trainingData, 1);
    numObservationsLabels = numel(trainingLabels);
    numValidationsData = size(validationData,1);
    numValidationsLabels = numel(validationLabels);

    if (numObservationsData == numObservationsLabels &&
         numValidationsData == numValidationsLabels)
        disp('in')
        trainedNet = trainNetwork(trainingData,
                            trainingLabels, lgraph, options);
    end
```

When Matlab threw an error saying: “Number of observations in X and Y disagree.” I more the double-checked all observations in the program and set them to right order, they are supposed to be based on Matlab's instructions. Though I am certain of that, I sadly do not know where the error lies. Unfortunately, I could not plot a graph or provide any table according to which I could have made reflection on the built neural network.

# Chapter 4 Conclusion

In conclusion, this thesis focused on exploring the effectiveness of Mel Frequency Cepstral Coefficients (MFCC) and other cepstral coefficients, along with spectral entropy and spectral centroid, for audio signal analysis. The objective was to leverage these features in the development of a Convolutional Neural Network (CNN) for audio classification.

Throughout the research, various audio processing techniques were applied to extract MFCC and other cepstral coefficients, which capture essential characteristics of the audio signals. Additionally, spectral entropy and spectral centroid measures were calculated to provide additional insights into the frequency distribution and energy concentration within the signals. The extracted features demonstrated their significance in representing audio signals and capturing relevant information for classification tasks. They offered a compact representation of the audio data, reducing dimensionality while retaining important discriminative information.

The obtained cepstral coefficients have been proven as an effective tool in capturing the spectral shape of the signal with emphasis on perceptual properties. MFCC provides a compact representation of the spectral information and is relatively robust to noise and other variations in the audio signal. However, MFCC may not capture fine-grained spectral details and may have limitations in capturing non-linear variations in the spectral domain.

To further enhance the analysis and classification capabilities, a CNN architecture was designed and implemented. The CNN architecture comprised multiple layers, including convolutional, pooling, and fully connected layers, followed by a softmax activation layer for classification. The training phase of the CNN, which aims to optimize the network parameters, plays a crucial role in achieving accurate and reliable classification results. However, due to certain challenges with the dimensions encountered during the training of the built CNN, this specific aspect could not be completed within the scope of this thesis.

Nevertheless, the research presented a comprehensive analysis of MFCC and other cepstral coefficients, along with spectral entropy and spectral centroid, highlighting their potential for audio signal analysis and classification. The work laid a solid foundation for future investigations and improvements in training the CNN model, which could involve adjusting parameters, refining the architecture, or employing alternative optimization algorithms.

Overall, this thesis contributes to the field of audio signal processing and classification by providing valuable insights into the application of MFCC and cepstral coefficients, alongside spectral entropy and spectral centroid, and laying the groundwork for the development of a CNN model to further enhance audio classification capabilities.

As a goal to the future could be an implementation of a multi-modal monitoring system that integrate visual and auditory information to improve situational awareness and enhance the accuracy of event detection and recognition which ultimately is the best choice that may be involved using a combination of cameras and microphones, as their respective strengths can complement each other and provide a more comprehensive understanding of the environment. Integration of visual and auditory data can enhance situational awareness and enable a more robust analysis of complex scenarios. For example, combining camera footage with audio data can enable more comprehensive monitoring of complex areas.

# Appendix A

# Bibliography

[Fay16] Haytham M. Fayek, *Speech processing for machine learning: Filter banks, mel-frequency cepstral coefficients (mfccs) and what's in-between*, 2016.

[LG09] Howard Lei and Eduardo Gonzalo, *Mel, linear, and antimel frequency cepstral coefficients in broad phonetic regions for telephone speaker recognition*, 09 2009, pp. 2323–2326.

[Lyo09] James Lyons, *Mel Frequency Cepstral Coefficient (MFCC) tutorial*, `http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs`, 2009.

[Mat22] MathWorks, *mfcc*, `https://www.mathworks.com/help/audio/ref/mfcc.html#mw_42368d66-4f36-4220-a679-1ab088533c79`, 2022.

[SH23] Jakub Svatos and Jan Holub, *Impulse acoustic event detection, classification, and localization system*, IEEE Transactions on Instrumentation and Measurement **72** (2023), 1–15.

[ZKT22] Yasmina Kheddache Zahra Khalilzad and Chakib Tadj, *An Entropy-Based Architecture for Detection of Sepsis in Newborn Cry Diagnostic Systems*, `https://www.mdpi.com/1099-4300/24/9/1194`, 2022.