BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION Computer Science, English

DIPLOMA THESIS

Automatic music transcription

Supervisor
Conf. Dr. Adrian-Ioan Sterca

Author
Dora-Maria Moraru

2022

UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA Informatică, Engleză

LUCRARE DE LICENŢĂ

Transcrierea automată a muzicii

Conducător științific
Conf. Dr. Adrian-Ioan Sterca

Absolvent
Dora-Maria Moraru

2022

ABSTRACT

There are many people that are interested in playing musical instruments and many who are interested in learning to do so. Even though the classical songs provide a good challenge and learning experience, people might find it more interesting to play contemporary songs, especially new learners might be more attracted to newer songs. The limitation is that most songs do not have a free public musical sheet that can be used by everyone.

We provide through this work a method of generating the corresponding transcription of a given musical composition in an automated manner. As songs might be complex and use multiple instruments, we plan in the beginning to generate the musical sheet for a single instrument, considering this as already being useful. We will employ sequence-processing algorithms like Fast-Fourier Transform, repetitiveness detection by using Audio Similarity Matrix.

The problem we try to solve consists of several sub-issues such as note, tempo and time signature detection. For each of them we dedicated a section where we present the solution implemented in this application. Since these problems are different by their characteristics, they are approached in distinct manners. As far as our knowledge, there is still no solution proposed that is addressed to the entire process of transforming a song into a musical score, but only to parts of it. In this paper, we gather all the parts that form a music sheet and compute them one by one.

Contents

Chapter 1

Introduction

Music surrounds us from our very first moments in life. It is present in the relaxing songs for the babies, in the Disney cartoons, on the radio, in the TV commercials, movies. It is a set of sounds that placed at the right time, with the proper duration and the correct accent, manage to create a musical piece that makes us humans, not only hear it, but also feel it. Therefore, songs are very complex by their composition. They are constructed by the accumulation of one or more instruments and/or voices, which emit different sounds that are formed by notes and chords of different durations which can be played at distinct rhythms.

Listening to songs is an entertaining experience for the music lovers and since the CDs first appeared in 1982, it got easier to live this experience more often. In our days, there are a lot of streaming services such as YouTube and Spotify which offer us the perfect song with only one tap of a button. But what still seems to be difficult to accomplish, in this period of time, is playing an instrument. This activity not only that gives pleasure to the player or the audience, but also helps the performer in developing certain skills such as coordination, attention, concentration, discipline, language and even maths. There are some people which simply have the gift of a great musical ear and with some knowledge of an instrument, they can perform a song only by hearing it. The others who are familiar with musical instruments, they need a lot of practice to read a music sheet, figure out the notes and their durations, the pace at which they should play the song. This is also not impossible to be accomplished once one has the music score. The problem appears when for a certain song, a music sheet, as far as our knowledge, does not exist yet.

In this paper we try to start solving this issue of not being able to perform any song that one may desire. The application presented in this thesis aims to transform a song into a musical sheet. As stated before, songs have a very complex composition, hence we tried splitting this big problem into more sub-problems which are treated in different manners. The problems that we attempt to solve are regarding: *note*, *tempo* and *time signature* detection.

The thesis is composed of 6 chapters which explain the parts that helped us develop such an application. The first chapter is the introduction which provides the motivation of the thesis and the main objectives of this work. In the second chapter, we present some musical concepts required for understanding the issues, as well as having a bigger picture of the components of a song. The third chapter is dedicated to other articles that approach any of the problems mentioned before. The theoretical background in terms of processing an audio signal, with algorithms such as Fast-Fourier Transform or visual representation as spectrograms, are detailed in the fourth chapter. In the fifth chapter, we start presenting the application by explaining the algorithms applied, the chosen architecture and technologies, the functionalities provided by our application, but also some experiments that we conducted. The last chapter presents information about future work and our final conclusions.

Chapter 2

Music structure concepts

In this chapter, we will explain the required knowledge related to music theory used in this application.

2.1 Pitch

Pitch is a characteristic which helps us classify a note as being high or low. It defines how a note sounds. The sound wave makes the air vibrate which makes the particles from nearby to vibrate from the sound source to our ears. The number of vibrations of a sound pitch in a unit of time is called frequency. If the sound vibrates at a high rate, the frequency will increase. The volume of the sound wave is given by the amplitude, which can be measured from the highest point of the wave (peak) to the lowest (trough). Wavelength is the distance between two peaks of the wave and is related to the frequency. The shorter the wavelengths are, the higher the frequencies will be. A visual representation of what we have explained can be seen in Fig. 2.1.

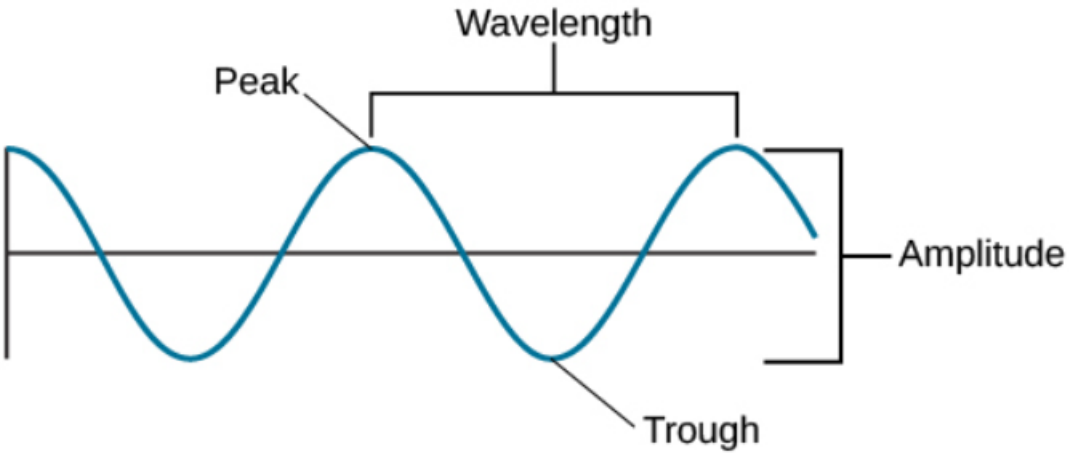

Figure 2.1: Sound wave [1]

Notes are formed by many different pitches, which are called harmonics or overtones. When the sound vibrates in halves, it produces the second harmonic, while when vibrating in thirds, it produces the thirds harmonic. A case as described before can be seen in Fig. 2.2. The first harmonic is the pitch that we hear, which is the fundamental frequency of the note, and the next ones are the multiples of it.

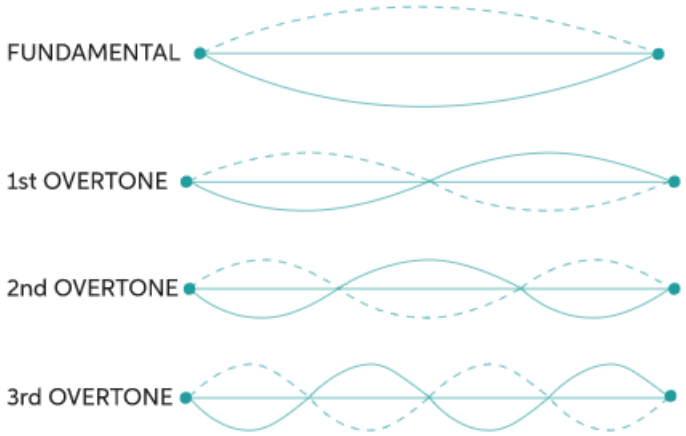

Figure 2.2: Representation of the relation between harmonics [2]

2.2 Tempo

Beat is a unit of time which usually remains constant through the whole song and it is associated with a value of a note. Musicians often count the beats to make sure they are synchronized and they play the note for the right duration. The tempo tells us how fast or slow the song is performed and it is the number of beats per minute (BPM). It can be also described in words such as "Adagio", "Allegro" (Fig. 2.3), but usually is represented in BPM (Fig. 2.4), value which is correlated with the denominator of the time signature.

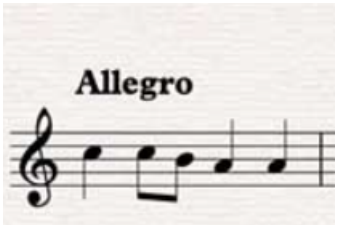

Figure 2.3: Tempo represented in words

[1]https://courses.lumenlearning.com/atd-herkimer-intropsych/chapter/waves-and-wavelengths/

[2]https://www.chegg.com/learn/physics/introduction-to-physics/overtones

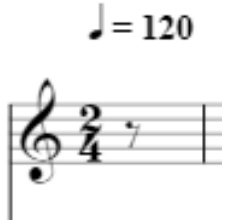

Figure 2.4: Tempo represented in BPM

2.3 Time signature

A song is composed by multiple parts which are called bars or measures. The length of a bar is written at the begging of the music score and it is formed by a numerator and a denominator. The numerator represents the number of beats which are in a bar and the denominator tells us which note value is associated with the beat. In Fig. 2.5 we can see a time signature with the value 3/4, where 3 indicates the number of beats in a bar and 4 indicates the note value which gives the beat (quarter note).

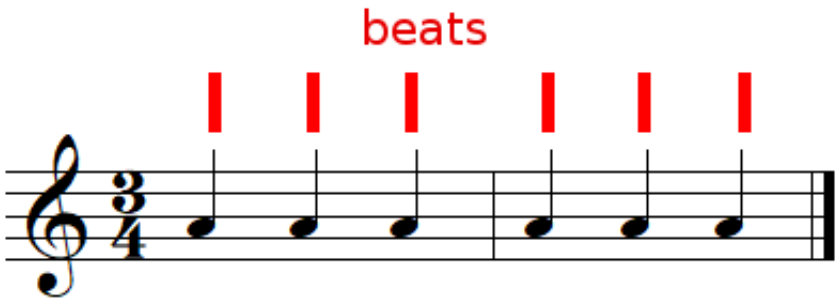

Figure 2.5: Time Signature 3/4

As explained before, there is a strong relation between beat, tempo, time signature, measure and note values. An example that will give a better understanding is given in Fig. 2.6. For the 3/2 time signature, since the denominator has the value 2, the half notes will take the beat. The numerator is 3, hence the number of beats from a bar is also 3. This can be translated into 3 half notes per bar, or any other combination of notes that make a total of 3 beats.

2.4 Duration

Besides their frequency, notes have another characteristic which we are interested in, namely duration. This attribute is relative to the tempo of the musical piece and to the note value. All the note lengths are derived from the whole note, namely dividing the whole note by two, we get the half note, dividing it by four, we get the quarter note (Fig. 2.7).

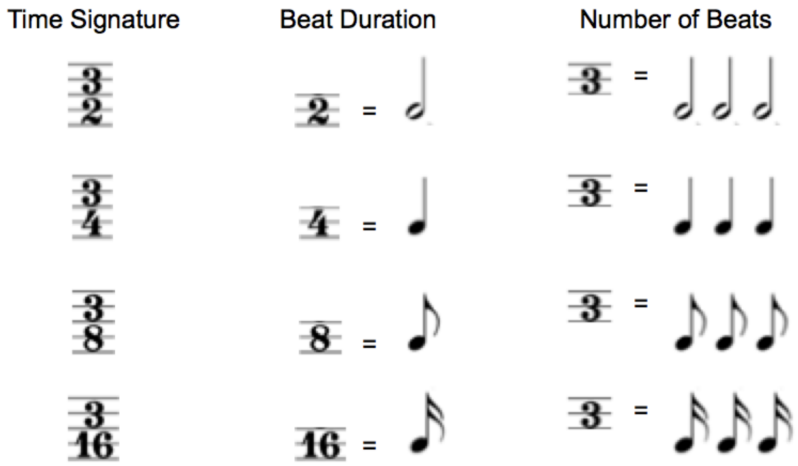

Figure 2.6: Relation between Time Signature, Beats, Measure

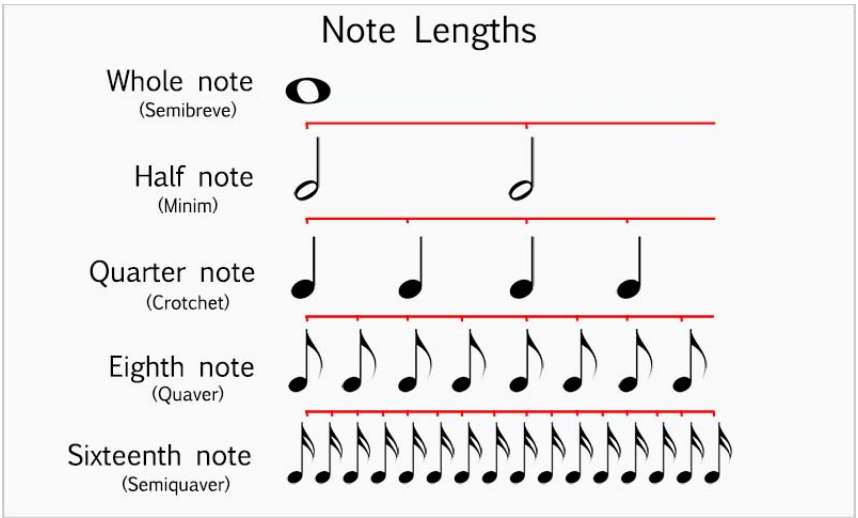

Figure 2.7: Note lengths

After we have the information about the note value and tempo we can find the exact duration in seconds of a certain note.

Chapter 3

Related work

As transforming a song into a music score is a complex problem that implies many sub-problems, much work in the scientific literature is focused strictly on one sub-problem at a time. To the best of our knowledge there is no single solution that presents how to solve all of the sub-problems that we have considered. The components that we envisioned are the following:

- note detection
- note identification
- note length detection
- time signature identification
- tempo recognition

The note detection problems concern finding the position of the notes in the song. There are many proposed solutions which try to solve this unknown. A way of solving this problem is presented in [BDA+05] and implies onset detection. The onset is the moment where the note starts and it can be easily visualised as its amplitude increases (Fig. 3.1).

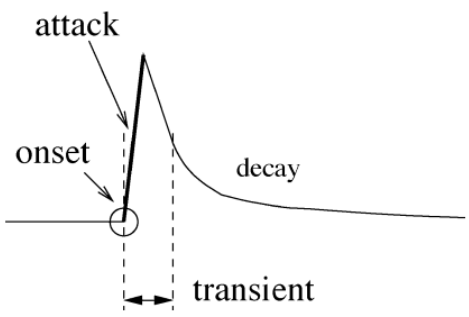

Figure 3.1: Onset of a note [BDA+05]

They start pre-processing the audio signal by applying a smoothing function in order to improve the performance of the next stages. Then, in the reduction step they transform the sound into a detection function with much less samples and this way reducing the complexity of the original signal. The authors of the paper took several approaches which were divided in two categories: reduction based on signal features and on probability models. In the final step, peak-picking algorithms are applied. For this, they computed an adaptive threshold which would help them detect the local maxima and therefore, the notes. They applied this method on piano and violin songs, but also on speeches. The results are really impressive because on each algorithm proposed they got over 80% positives.

Another article that got our attention was [PG15] because they concerned about three of our problems: note detection, note identification and note length. The authors also applied a smoothing function over the original signal by making an average of each 100 set of samples. On the next step, again a thresholding was needed for determining the peaks. They looked for a minimum width between the notes, given by the formula 3.1, where fs is the sampling frequency and A is the number of samples from the original song that corresponds to one sample from the average signal:

$$W_{min} = \frac{fs}{(A * 20)} \tag{3.1}$$

After detecting the position of the notes, the note identification problem was approached. They took the intervals of each note and treated them separately by applying the Hanning window function and the FFT, this way determining the fundamental frequency of the notes. Having also the intervals of the notes, it was easier for them to find also their lengths.

The authors G. Tzanetakis, G. Essl and P. Cook wrote a paper [TEC01] regarding beat detection. They decomposed this problem into five steps:

1. LPF - low pass filtering 3.2, where α has the value 0.99

$$y[n] = (1 - \alpha)x[n] - \alpha y[n] \tag{3.2}$$

2. FWR - full wave rectification 3.3

$$y[n] = |x[n]| \tag{3.3}$$

3. Downsampling 3.4

$$y[n] = x[kn] \tag{3.4}$$

4. Normalization 3.5

$$y[n] = x[n] - E[x[n]] \tag{3.5}$$

5. ACRL - autocorrelation 3.6

$$y[k] = \frac{1}{N} \sum_{n=0}^{N-1} (x[n]x[n+k]) \tag{3.6}$$

After applying these formulas, a histogram is created by iterating over the signal and taking the first five peaks and their periodicities in beats per minute. The periodicity corresponding to the highest peak is the final estimated tempo of the song.

A multi-task learning approach is described in the paper written by S. Bock, M.E.P. Davies and P. Knees [BDK19]. They try to solve the problems regarding beat-tracking and tempo detection. Since it is much harder to annotate an entire song with the exact positions of the beats, there are more datasets which contain information about tempo than beat-tracking. Beats positions are correlated with the value of the tempo, hence they constructed a system which not only handles both tasks, but if tempo or beat information is not existent for the training, then it will still be obtained an approximation for both of them. For this matter, they have chosen Temporal Convolutional Networks (TCN) as a machine learning algorithm. Their architecture contains a common part for both tasks (beat-tracking, tempo estimation), which is the backbone. The architecture also contains two dedicated parts, one for each task, which are the heads. The backbone consists of multiple layers, stacked over each other and also linked by skip connections. Skip connections copy the output from a layer which is at a lower level and offer it as an additional input to a layer which is at a higher level. These were proven to help train networks with a high number of layers [HZRS16]. The final output of the backbone is given to the beats head that will further process and obtain the predicted beats. For the tempo, the inputs are taken from multiple layers of the backbone through skip connections.

The time signature problem was approached in the article of M. Gainza and E. Coyle, in the paper [CG07], where they transformed the audio signal into frames by using the formula 4.6 in order to compute a spectrogram. The authors then found the starting moment of the song (when the first note enters) and eliminated that previous part from the spectrogram because it does not provide any useful information in their research. In the next stage, they constructed an Audio Similarity Matrix based on the equation 4.7. Having all this information, they looked for bars that were parallel with the main diagonal. M. Gainza and E. Coyle considered various lengths for the bar, from the interval 2 to 12, trying to find which one of these

values would fit best. They have answered to this question by constructing another similarity matrix for each possible length of the bar. By computing a similarity measure for each of these matrices and taking the highest value of similarity they found the numerator of the time signature. Then, the denominator of the signature will be found by taking the nearest power of two of the numerator. For example, if the numerator is determined to be 2, 3 or 5, then the time signature will correspond to the values 2/2, 3/4 and 5/4 respectively.

Chapter 4

Analyzing tools

In this chapter we will present the theoretical background needed for implementing our application.

4.1 Time domain vs Frequency domain

The time domain is an analysis of physical signals or mathematical functions in respect to time and shows how a signal behaves over time, while the frequency domain is an analysis of signals and mathematical functions in respect to frequency. The latter displays how the signal exists in a range of frequencies. Time domain signals are represented by amplitude on the Oy axis and time on the Ox axis. On the frequency domain representation we have the amplitude on the Oy axis and frequency on the Ox axis. We present a graphical representation in Fig. 4.1. Each of them are important, having different purposes and usages depending on what we are searching for. Also, the time domain can be converted to the frequency domain and the other way around.

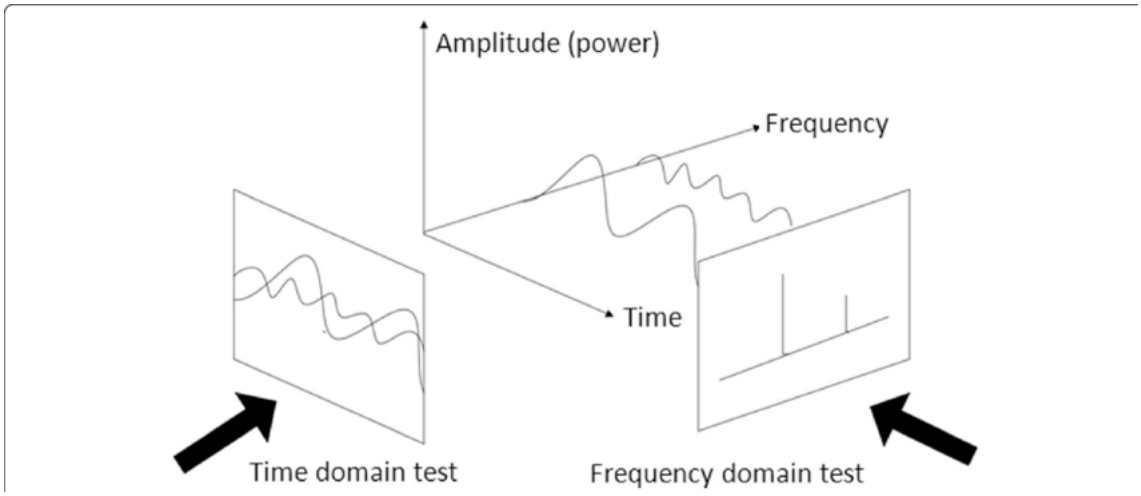

Figure 4.1: Time Domain and Frequency Domain Representation [Sun19]

4.2 Fourier transform

As stated before, the time domain can be converted to the frequency domain and this can be achieved using the Fast Fourier Transform (FFT) ([Wic10]). In fact, this is an optimised implementation of the Discrete Fourier Transform (DFT). The DFT transforms a sequence of samples into components of different frequencies. The main idea of the Fourier transform is to translate the signal into a periodic function of a sum of sine and cosine waves. Each wave constitutes an harmonic which is a multiple of the fundamental frequency of the period function. Adding these harmonics into a sum of Fourier series approximates that function (Fig. 4.2).

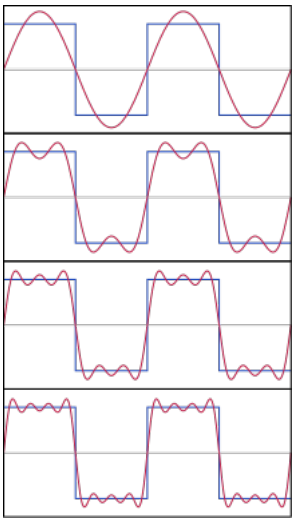

Figure 4.2: The first four partial sums of the Fourier series for a square wave [1]

The infinite series of sine and cosine terms of frequencies 0, ω, $2 * \omega$,..., $k * \omega$ is known as trigonometric form of Fourier series that can be written as in formula 4.1 and its coefficients (formulas 4.2, 4.3, 4.4), where T is a period satisfying the condition of Dirichlet (formula 4.5).

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty}(a_n * \cos(n\omega t) + b_n * \sin(n\omega t)) \tag{4.1}$$

$$a_0 = \frac{1}{T}\int_0^T f(t)\,dt \tag{4.2}$$

$$a_n = \frac{2}{T}\int_0^T f(t) * \cos(n\omega t)\,dt \tag{4.3}$$

$$b_n = \frac{2}{T}\int_0^T f(t) * \sin(n\omega t)\,dt \tag{4.4}$$

[1] https://en.wikipedia.org/wiki/Fourier_series

$$\int_0^T |f(t)|\, dt \, is finite \tag{4.5}$$

An implementation of the FFT is made using the Cooley-Tukey algorithm ([Wic10]) which uses a divide and conquer technique which recursively splits any component of size N into two pieces of size N/2. After that, it applies the DFT on the resulted subsequences and then it combines the results in order to produce the DFT of the entire sequence. When all the computations are done, the final outcome is a frequency domain representation.

4.3 Spectrogram

A waveform displays how a signal's amplitude changes over time, but a spectrogram contains information about the amplitude, frequency and time. It is basically a two-dimensional graph, frequency on the Ox axis and time on the Oy axis, with a third value, amplitude, which is showed using colors (darker colors for low values and bright colors for big values, respectively). From this visual representation one can observe how the energy values change in time, the repetitiveness of the frequencies and the latter being unable to be noticed from the waveform graph (Fig. 4.3).

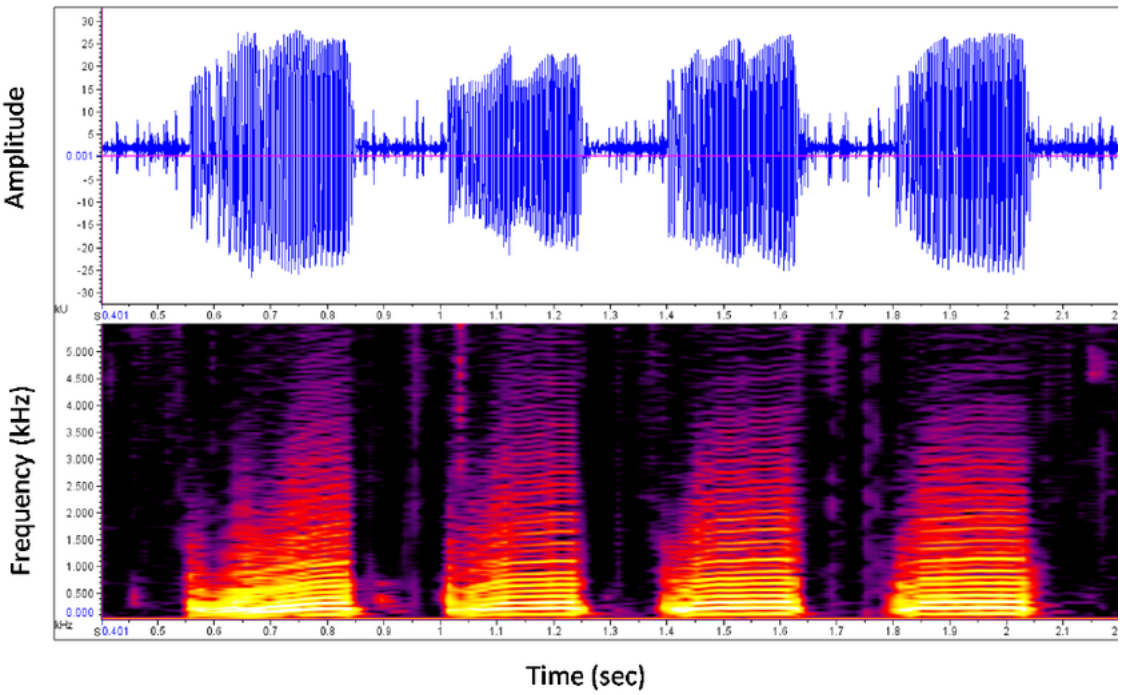

Figure 4.3: A spectrogram generated from a waveform [KL22]

In order to compute such a spectrogram, the time domain signal needs to be divided into smaller frames of equal length which are overlapping with a constant

hop-length. A window function is applied on the resulted frames in order to smooth them. Intending to obtain the magnitude of the frequency for each chunck, the Short-Time Fourier Transform (STFT) is used. Such a formula is described in 4.6, where L is the length of the frames, H is the hop-length, $w(n)$ is the windowing function, $x(n)$ is the input signal, m is the frame index, N is the FFT length, k is the bin number. Afterwards, each frame correlates with a vertical line from the spectrogram.

$$X(m,k) = |\sum_{n=0}^{L-1}(x(n+mH)w(n) * e^{-j(\frac{2\pi k}{N})*n})| \tag{4.6}$$

4.4 Audio Similarity Matrix

When it comes to data analysis, the Self-Similarity Matrix (SSM) is a visual representation of the similarity between sequences of data ([Foo99]). This similarity can be computed in different manners, such as spacial distance, spectral properties or comparison. In order to compute such a similarity matrix, one transforms the input data into a series of vectors containing the necessary attributes of the data. Afterwards, a function which calculates the similarity is applied on the pairs formed by all the possible combinations of these vectors.

In case of musical signal, it is advised to convert it into a suitable sequence which allows proper comparison. In this paper, we use the Audio Similarity Matrix to find the repetitiveness of fragments of the song. This means that a certain succession of notes is repeated. Therefore, we aim to search for similarity between frequencies from a short time interval. As stated before, we need to obtain a vector that contains the necessary information, so for this problem, the best choice is the spectrogram because it contains the frequencies in respect to time.

After obtaining the spectrogram from the audio input, the Audio Similarity Matrix is computed by comparing each two spectrogram frames using the Euclidian Distance Matrix. A useful mathematical representation is presented in formula 4.7, where a and b are two frames.

$$ASM(a,b) = \sum_{k=1}^{\frac{N}{2}}(X(a,k) - X(b,k))^2 \tag{4.7}$$

As for the visual representation, the color of the cells of the matrix shows the level of similarity between frames. The darker the color, the bigger the similarity. An example of such a ASM is shown in Fig. 4.4.

[2]https://samgoree.github.io/2017/12/29/Musical_studies_capstone_thesis.html

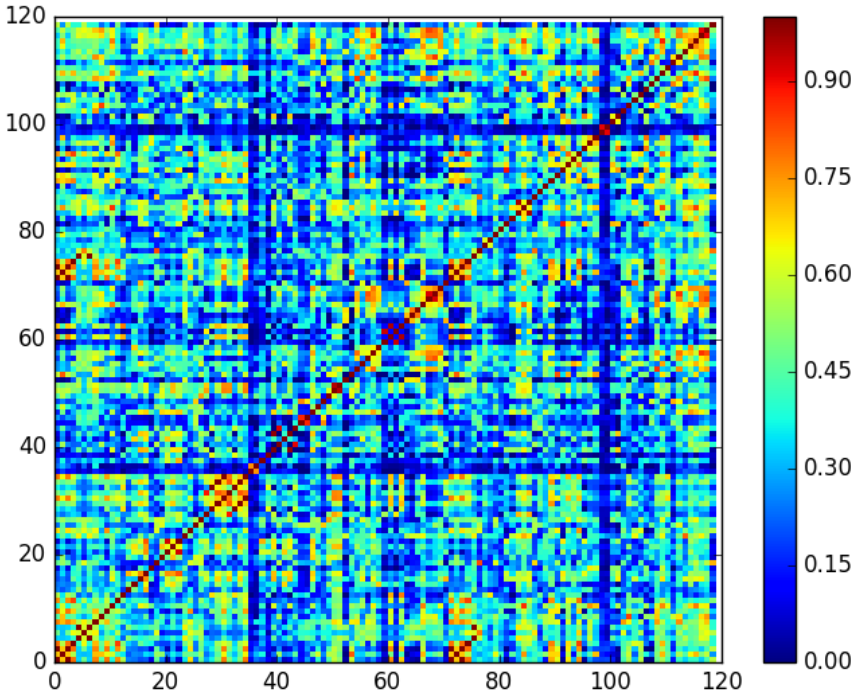

Figure 4.4: Audio Similarity Matrix [2]

4.5 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are machine learning algorithms which were originally developed for processing images, but they were later extended to other use cases. They are formed by stacking multiple layers, each one processing the output of the previous. These layers can be of different types.

A first very important type is the convolutional layer. This layer convolves a set of filters over an input to produce the output. In the one dimension (1D) case, the simplest option is to have an array as input and the convolution layer to have a single filter which is another array. The filter is moved over the input and at each position, element-wise multiplication is computed between the filter and the available part of the input (Fig. 4.5). The obtained values at this positions are added and a non-linear "activation" function is applied. A very popular activation function is the Rectified Linear Unit (ReLU) $ReLU(x) = max(0, x)$. The layer may have inputs with multiple channels (values on each position of the input) and one convolutional layer may have multiple filters.

Batch normalization is a technique (and also a layer type) which standardizes the outputs of the previous layer by rescaling them. This was proven to help the

[3]https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/1d-convolution

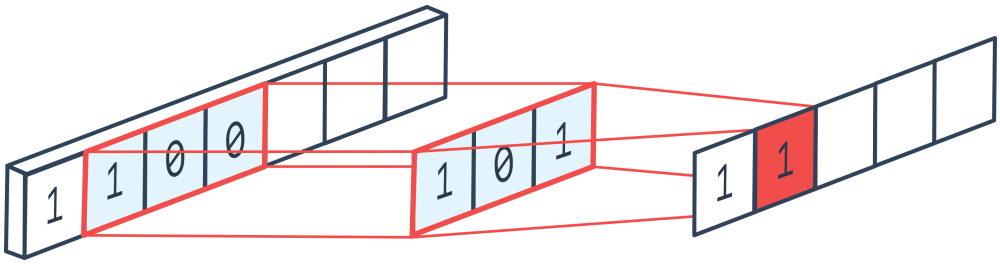

Figure 4.5: Convolutional Filter 1D [3]

networks learn faster during training [IS15].

There are multiple other types of layers that can compose a CNN, but as we do not use them in this work, we do not describe them.

4.6 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are machine learning algorithms designed to process sequences [LBE15]. They keep an internal state which is updated after processing each value in the sequence. This internal state is used to compute the outputs for each input.

Most modern RNNs use Long Short-Term Memory (LSTM) cells as hidden units [HS97]. These cells were proven to help RNNs to find dependencies between values that are far from each other in the sequence.

The computations done by LSTM are presented in formulas 4.8-4.13. Here W parameters are learnable weights and b parameters are learnable biases. The superscripts represent the step at which we refer (t is the current timestep and $t-1$ is the previous one). x is the input and h is the hidden state of the network. ϕ and σ are the hyperbolic tangent and sigmoid functions which are described in equations 4.14, 4.15. g is the input node and s is the internal state. i, f, o are the input, forget, output gates respectively.

$$g^{(t)} = \phi(W^{gx}x^{(t)} + W^{gh}h^{(t-1)} + b_g) \tag{4.8}$$

$$i^{(t)} = \sigma(W^{ix}x^{(t)} + W^{ih}h^{(t-1)} + b_i) \tag{4.9}$$

$$f^{(t)} = \sigma(W^{fx}x^{(t)} + W^{fh}h^{(t-1)} + b_f) \tag{4.10}$$

$$o^{(t)} = \sigma(W^{ox}x^{(t)} + W^{oh}h^{(t-1)} + b_o) \tag{4.11}$$

$$s^{(t)} = g^{(t)} \cdot i^{(i)} + s^{(t-1)} \cdot f^{(t)} \tag{4.12}$$

$$h^{(t)} = \phi(s^{(t)}) \cdot o^{(t)} \tag{4.13}$$

$$\phi(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{4.14}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{4.15}$$

In the classical one directional case, the sequence is processed value by value from the first to the last. There is also the variation of bidirectional RNNs which process the sequence in both directions. This gives a better context of the surroundings for processing each input value. A graphical representation can be seen in Fig. 4.6.

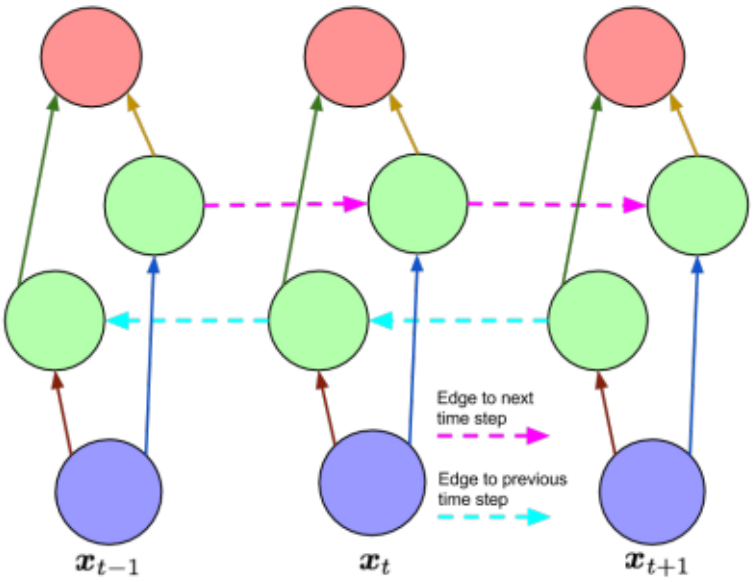

Figure 4.6: Bidirectional RNN

Chapter 5

Detecting musical structure in music

The application's purpose is to convert a song into a music sheet. In order for this to be accomplished, the user must upload a WAV file which will be sent to the server, where it can be processed. After all the computations are done, a MusicXML file is created and sent as a response back to the front-end. In the GUI, this XML file is shown as a music score. The latter is iterated and the notes from it are played on the press of a button.

5.1 Specification

The application must provide the following functionalities:

- Convert a WAV file into a music score by:
  - Finding the notes and their duration from the song
  - Detecting the tempo of the musical piece
  - Obtaining the time signature of the music file provided
- Play a music score
- Show in real-time the iteration of the music sheet
- Download the generated music score
- Change the processing algorithms for tempo and time signature detection

The Use Case Diagram of the application can be seen in Fig. 5.1.

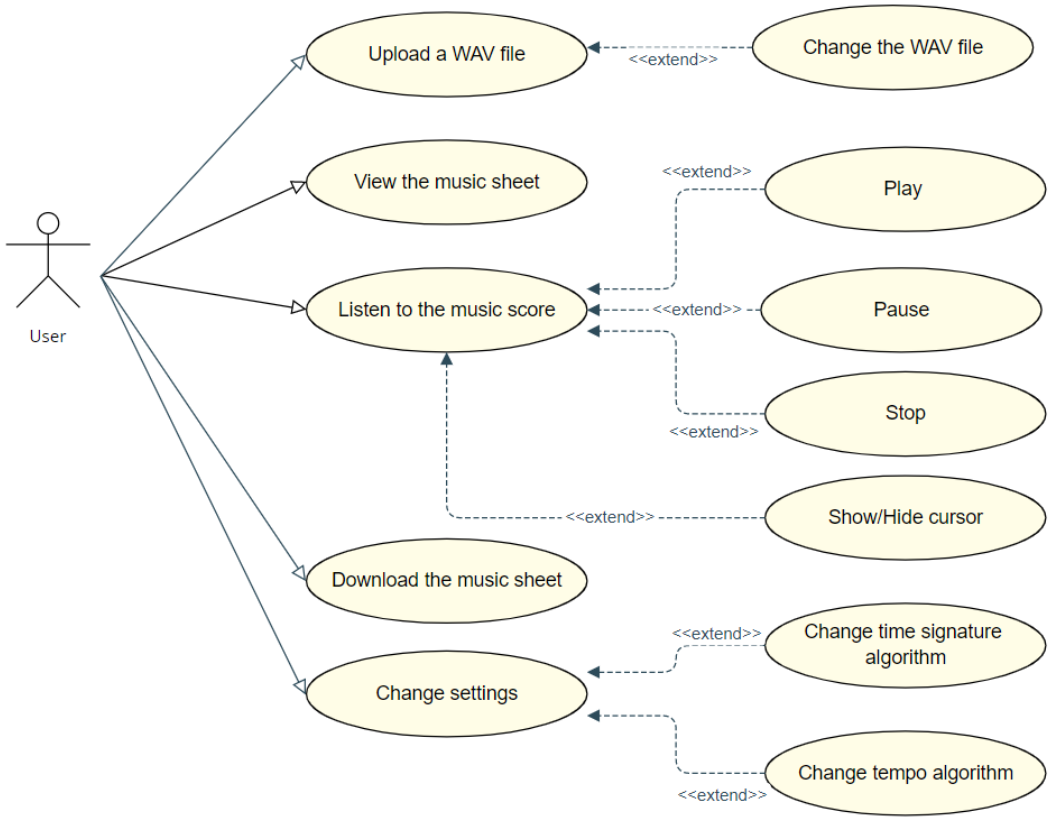

Figure 5.1: Use Case Diagram

5.2 Algorithms

5.2.1 Note detection

The set of algorithms starts with the detection of the notes from the musical piece. The signal waves fluctuate a lot and in a very short time, hence it is very difficult to work with such data. In order to make the computations easier and clearer, a smoothing function is applied over the signal. Firstly, the data is read from the WAV file extracting also the sample frequency. The obtained result from this operation is a vector of values which are positive but also negative numbers, because of the fact that the local air pressure is lower or higher than the ambient pressure. In consequence, for the next computations, an absolute function is applied element-wise on the array (Fig. 5.2). In this part of the process, we aim to find the intervals with higher amplitudes, since these represent the intervals of notes.

As stated before, a smoothing function has to be used. The article written by M. Asadullah and S. Nisar [AN17], presents a method of detecting the sequences with low amplitudes which contain unnecessary information. To obtain such a result, the array of signals is split into frames of 512 samples and the root mean square (RMS) value of each frame is computed as in formula 5.1.

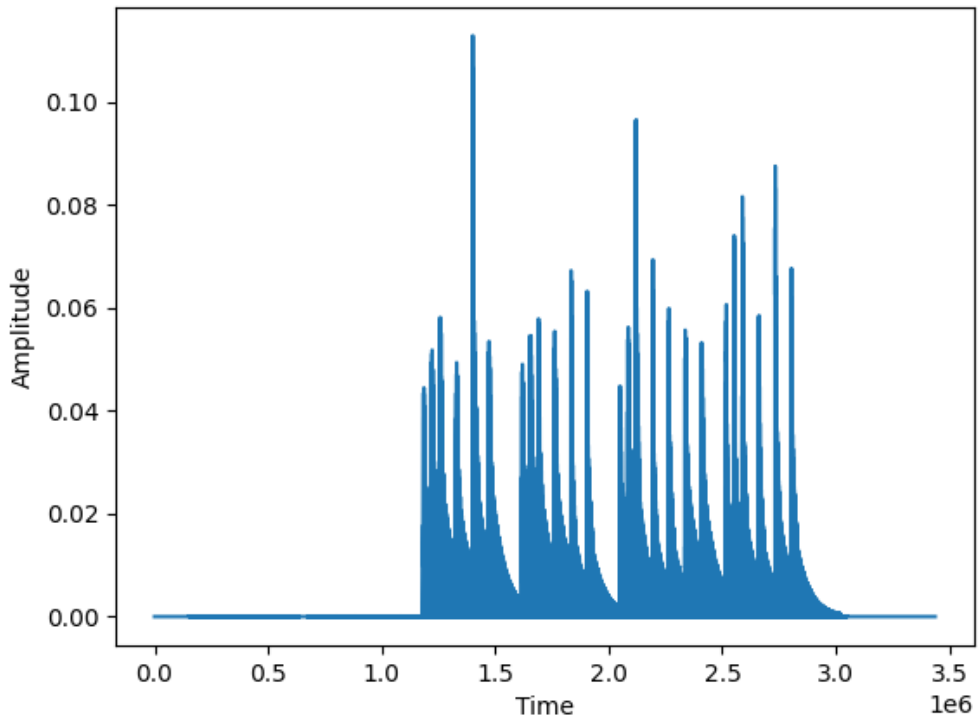

Figure 5.2: Audio signal after applying the absolute function

$$RMS_{Segment} = \sqrt{mean(Segment)^2} \tag{5.1}$$

For finding the parts that fulfill the constraint of having high amplitudes, we calculate a threshold and compare each frame with it. The threshold value is obtained from the formula 5.2, where ν is the minimum RMS value of the first 100 loudest frames and μ is the mean value of the first 100 silent frames.

$$R_{th} = \frac{\mu + \nu}{2} \tag{5.2}$$

The graphical representation of a song over which we applied the previously mentioned computations can be seen in Fig. 5.3. The orange line represents the threshold.

After having all these information, we start searching for intervals with low amplitudes, that may contain unnecessary data. The following calculations are made on the new array formed by the RMS values. Parsing the data, we look for consecutive values which are below the threshold, in order to form intervals. A new array is constructed by tuples which contain the start and end position of such an interval. As we previously mentioned, the signal waves tend to fluctuate a lot, hence there might be some intervals which, even though they attain the condition of being un-

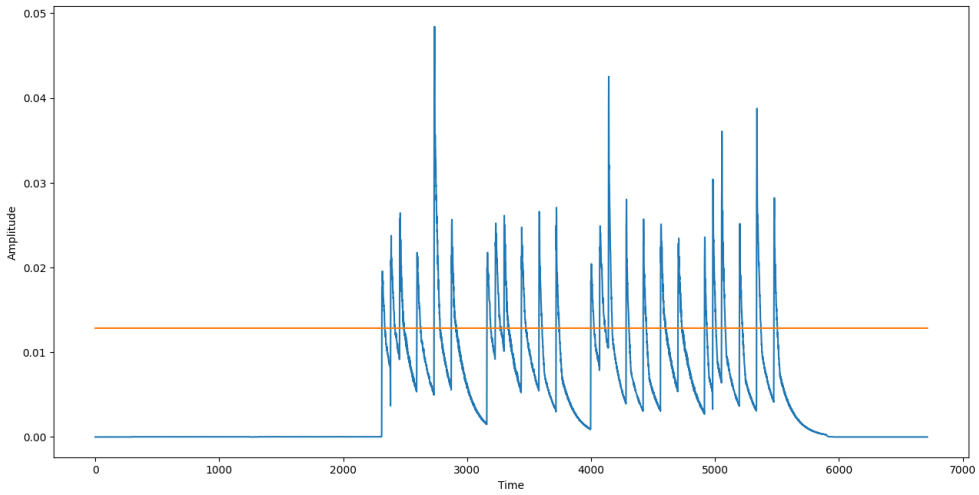

Figure 5.3: The graphic of a song after computing the RMS and threshold

der the threshold, are too short to be taken in consideration. The minimum duration of silence between two notes is known to be 0.5ms, therefore, the intervals are filtered so their length satisfy this requirement. The result can be observed in Fig. 5.4, where the red dot is corresponding to the start of the silence interval and the green dot to end of the interval.

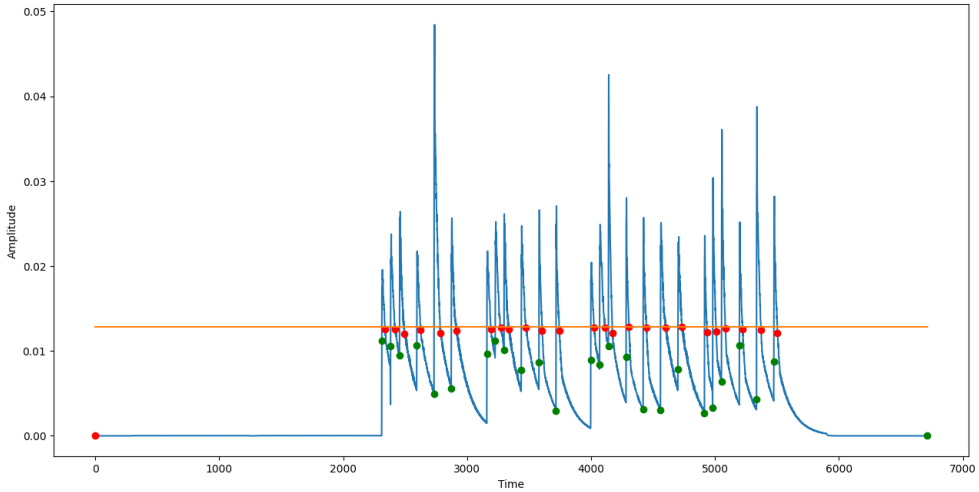

Figure 5.4: The graphic of the silence intervals

The intervals of notes can now be easily identified from the results obtained until now, by considering the end of a silence interval (green dot) as the beginning of a note and the start of the next silence interval (red dot) as the end of the note. Each derived sequence of data belonging to one note is then passed to another function

which will compute the fundamental frequency of the note.

Fundamental frequency

In order to find the fundamental frequency of a note, we need to transform the time domain input into a frequency domain and for this matter the Cooley-Turkey algorithm which implies the FFT will be applied. The Fast Fourier Transform works based on the assumption that the signal is periodic and has an integer number of periods in that time interval. Although, in many cases this is not true and the sound may have a truncated waveform, therefore the endpoints are not continuous. These discontinuities appear in the graphic representation of the FFT as high-frequency components which are not present in the original signal. To minimize these effects, a windowing method is used. The latter reduces the discontinuities at the endpoints of each sequence, by making the amplitudes smoother and gradually going towards zero on the edges. Hence, in our case, we applied a Hanning window over the input sound, then the FFT algorithm described in section 4.2.

As presented before, the fundamental frequency is graphically represented as the first peak of the frequency domain. The following peaks are the harmonics and are multiples of the fundamental frequency. Therefore, the next step is determining a threshold of the resulted domain. The values obtained after this processing respect a Gaussian distribution, making it appropriate to use mean and standard deviation for computing the threshold. This was calculated by the following formula 5.3, where $mean$ represents the mean of the amplitudes and std is the standard deviation of the amplitudes.

$$threshold = mean + 3 * std \tag{5.3}$$

We are interested in frequencies with the highest amplitudes, and this is why we used this threshold which filters out 99.7% of data.

All the peaks above the threshold are collected in an array and used for finding the fundamental frequency. Since the first possible fundamental frequency is 16.35Hz, any peak located before that frequency is ignored.

The idea we came for the algorithm of determining the fundamental frequency is the following:

1. Take the first peak and consider it as a possible fundamental frequency ($freq$). Set the number of peaks found ($nrPeaks$) to 1 and the multiplier to 2 (mul)

2. Set an acceptable range for the difference between the actual multiples of the fundamental frequency and the possible ones from the signal ($range = 0.15$)

3. Set the minimum number of harmonics to be found ($nrHarmonics = 5$)

4. Compute the next possible harmonic of the fundamental frequency considered until now ($harmonic$)

5. Compute the favorable distance with respect to the harmonic in the range where the actual harmonic may be located $dist = max(freq * range, 2)$

6. Compute the interval of the next possible harmonic

   $start = freq - dist$

   $end = freq + dist$

7. If the end of the array of peaks is reached, return the considered fundamental frequency ($freq$)

8. Search if one of the next peaks is in the favorable interval $peak \in (start, end)$

9. If no peak is found, we go back to step 1, but this time we consider the next peak the fundamental frequency

10. If a peak is found ($foundPeak$), recompute the possible fundamental frequency as we now have new information about it (other harmonics)

    $freq = (freq * nrPeaks + foundPeak/mul)/(nrPeaks)$

11. Minimize the range where the next harmonic we are looking for might be located $range = range/1.1$. Increase the number of peaks found, and the multiplier

12. If we found the required number of harmonics, then we return the considered fundamental frequency. Otherwise, go back to step 4 and continue the process.

The visual representation of the G3 note, after applying the FFT algorithm, can be observed in Fig. 5.5. The orange line represents the threshold and the red dots are the peaks above the threshold which are considered for finding the fundamental frequency.

A zoomed in part of Fig. 5.5 is represented in Fig. 5.6. G3 has a fundamental frequency of 196.00 Hz. In this graphic, one can observe that the first peak is positioned close to that value of 196.00 Hz. The following peaks also seem to be placed on equal intervals (of 196.00 Hz) from one to another. This means that the first peak is the fundamental frequency of the note G3 and the next ones are the harmonics.

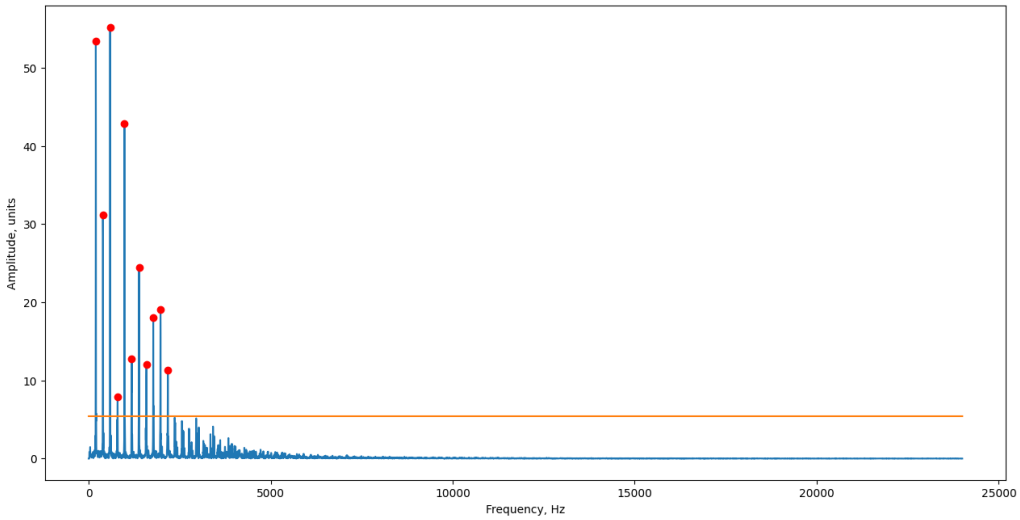

Figure 5.5: Graphic of a note G3 after applying FFT

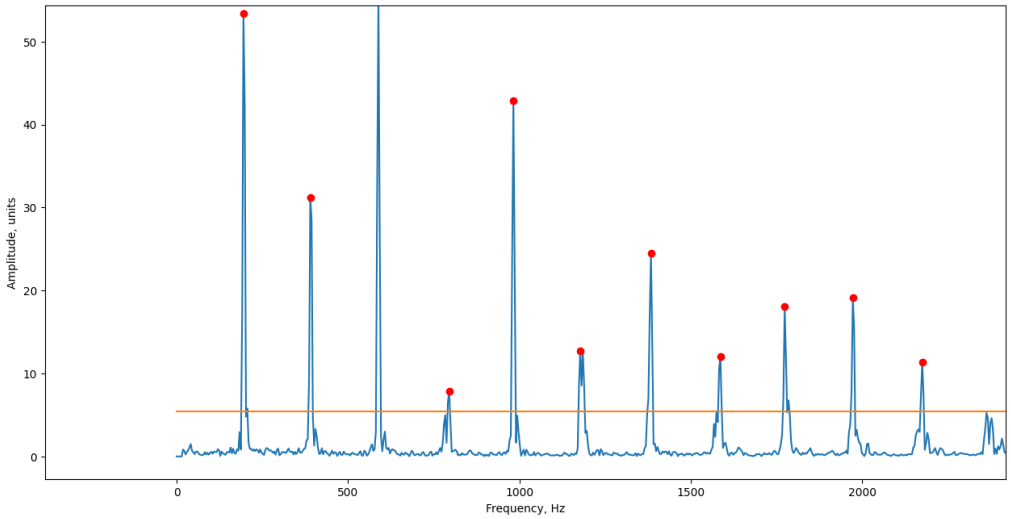

Figure 5.6: Zoomed in graphic of a note G3 after applying FFT

Find note name

The frequencies of notes are situated on an equal-tempered scale and are computed using the formula 5.4, where f_n is the frequency of the note, f_0 is the frequency of the reference note, a is $2^{\frac{1}{12}}$ and n is the number of half steps away from the input note.

$$f_n = f_0 * a^n \tag{5.4}$$

A part of such table can be seen in Fig. 5.7.

Note	Frequency (Hz)
C_0	16.35
$C^{\#}_0/D^{b}_0$	17.32
D_0	18.35
$D^{\#}_0/E^{b}_0$	19.45
E_0	20.60
F_0	21.83
$F^{\#}_0/G^{b}_0$	23.12
G_0	24.50
$G^{\#}_0/A^{b}_0$	25.96
A_0	27.50

Figure 5.7: Frequencies for equal-tempered scale, A4=440Hz [Not]

The reference note (f_0) is A4 with the frequency 440Hz. After having this table computed, the note correlated with the frequency that is the closest to the previously found fundamental frequency is considered.

Note duration

For finding the actual duration of a note, the use of previously found notes intervals and computing another threshold is required. The intervals of notes contain the essential information for finding the fundamental frequency, but may not contain the entire length of the note as the threshold used was pretty high (because we needed to find peaks). By sorting in increasing order the RMS values, and eliminating all the zero values from it, we obtain a new array. The new threshold is considered to be the 100th value from that new array. Below this threshold all the noise and silence

are supposed to be situated. In Fig. 5.8 is represented the graphic of the RMS values with the silence threshold (orange line).

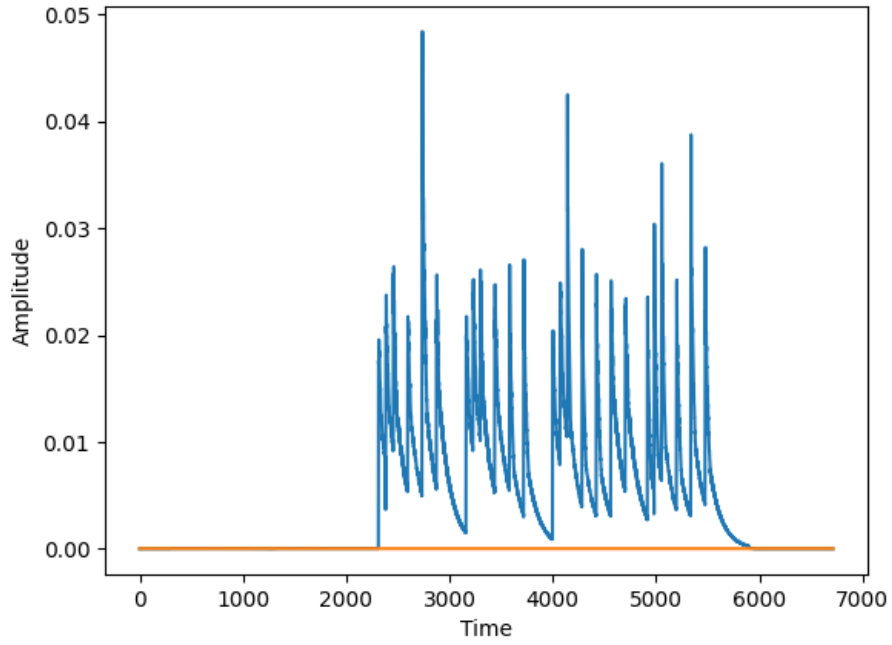

Figure 5.8: Graphical representation of RMS with the silence threshold

The starting position of all the notes are kept, because these ones are considered to be the onsets. In the meantime, the end of the note can be either the start of the next note, or the beginning of an interval that goes below the threshold.

Having the duration in seconds of a note, we must associate these seconds to a note value. For this, we need information about the tempo and time signature. Namely, diving 60 (seconds) to the tempo (number of beats per minute) we obtain the number of seconds which the note associated with the denominator of the time signature lasts. From this point on, all the other note values can be derived from this information. For example, for a tempo equal to 80 and a time signature of 4/4, the quarter notes will have a duration of 0.75 seconds and the half notes of 1.5 seconds.

5.2.2 Tempo detection

Tempo implies beat tracking which is still a problem that as far as our knowledge has no perfect solution. There are theories of how it might be solved, one of them being presented in this paper [TEC01]. By implementing this approach, the results obtained were very good in some cases, but in others were far away from the truth. Hence, we end up using a library called Aubio which contains a function that returns the positions of the beats. From that point on, a new array is created, having

elements which represent the differences between 2 consecutive beats. This value is expressed in seconds, hence it is converted to beats per second. By computing the median over these values, we obtain the most frequent tempo (formula 5.5).

$$bpm = median(60/differenceOfBeats) \tag{5.5}$$

5.2.3 Time signature detection

Audio Similarity Matrix approach

The time signature detection is a complex problem which, to the best of our knowledge, is a relatively unexplored area [CG07]. The method approached by us, proposed by the authors M. Gainza and E. Coyle, relies on the fact that a song has a repetitive structure, meaning that a part of the musical piece is reapeted in different parts of the song. Certainly, this is not case for all the songs, especially for the most complex ones, like the classical ones, but for the ones that we hear on the radio, for example, this is definitely the case.

Time signature is formed by a numerator and denominator. The numerator represents the number of beats from a bar and the denominator is the note value which the beat is associated with. For determining the numerator we implemented the following presented approach.

The first step is to create a spectrogram using the formula 4.6, where L is $1/32$ of the beat duration and H is $1/64$ of the beat duration. The beat duration is computed from the tempo information. An obtained spectrogram can be visualised in Fig. 5.9.

Having this representation, an Audio Similarity Matrix is constructed by computing the Euclidian Distance between two frames. Such a formula is presented in 4.7. A visualisation of the ASM for the same song (Twinkle Twinkle Little Star) can be observed in Fig. 5.10.

The darkness of each cell shows the similarity between two frames. Hence, the bright cells represent the dissimilarity and the dark ones the similarity.

The time signature values can vary a lot, therefore the possible solutions which we considered for the time signature are from 2/2 to 12/8, including the complex time signatures (11/8, 5/4, 7/8). As for this step, the numerator is related to the number of beats from a bar and the spectrogram frame is $1/64$ of the beat duration, the bar lengths that we will consider will be in range $2 * 64$ to $12 * 64$. Since the ASM is symmetric with respect to the main diagonal, we start parsing only one half of the matrix. The relevant information is located on the diagonals parallel with the main diagonal. Hence, for each possible length of bar, we apply the following steps:

1. Make non-overlapping groups of the bar's size on the diagonals

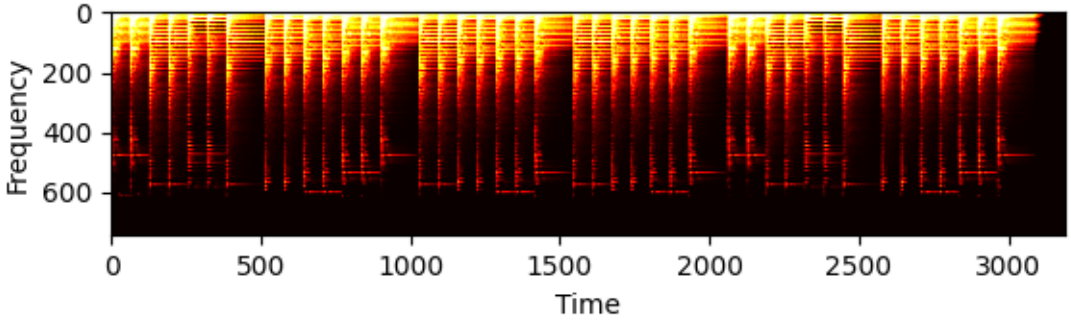

Figure 5.9: Spectrogram of Twinkle Twinkle Little Star

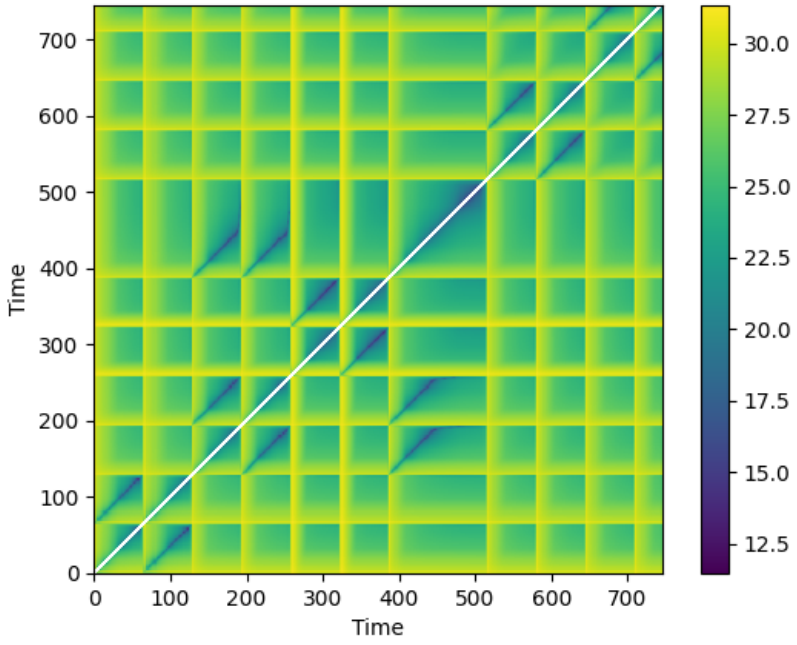

Figure 5.10: ASM of Twinkle Twinkle Little Star

2. Compute the similarity measure of each resulted group, using formula 5.6

$$SCS = \sqrt{\frac{\sum_{i=1}^{Bar} G_i^2}{Bar}} \tag{5.6}$$

3. Since not all groups will be complete, the formula 5.7 is used for those, where r is the length of the incomplete group

$$SIS = \sqrt{\frac{\sum_{i=1}^{r} P_i^2}{r}} \tag{5.7}$$

4. In order to measure the similarity for each length of bar, equation 5.8 is applied:

$$SM = \frac{Bar * \sum_{i=1}^{s_c} SCS_i + \sum_{i=1}^{s_i} r * SIS_i}{Bar * s_c + r * s_i} \tag{5.8}$$

5. This obtained similarity measure (SM) is added to a histogram.

After performing the previously steps for all the possible lengths of bar, we take from the histogram the bar length corresponding to the highest SM (Fig. 5.11).

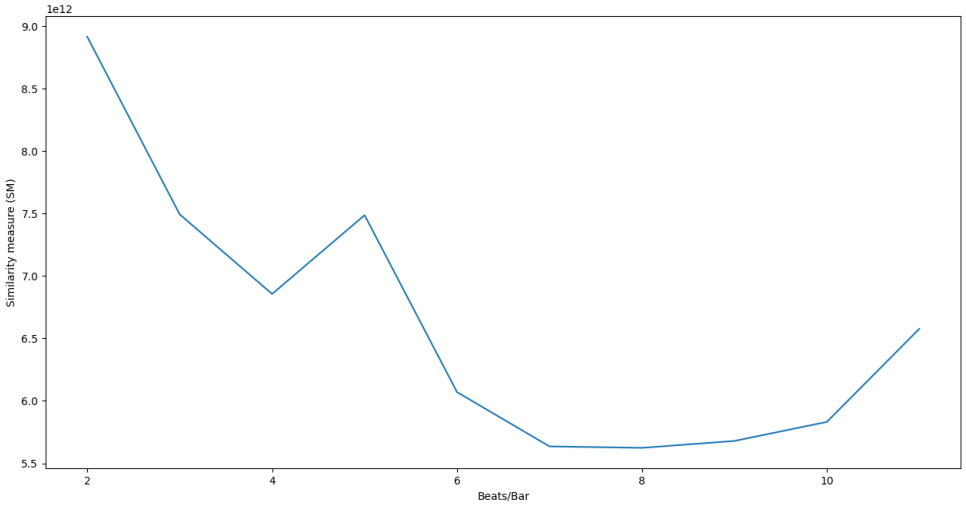

Figure 5.11: Histogram of Twinkle Twinkle Little Star

Having the numerator of the time signature, thus the number of beats in a bar, the denominator is left to be found. The latter can be obtained by rounding the numerator to the closest power of 2. Hence, for example for the number of beats 2, 3, 4, 5, the denominator will be estimated to 2, 4, 4, 4. Then, the time signatures 2/2 and 8/8 can be converted to 4/4. Although, there exist time signatures such as 3/8 which cannot be determined from our method, but this does not mean that our approach is wrong. The denominator tells us the note value that will take the beat

and it is also related to the tempo. For example, if the denominator should be 8, but it is detected as being 4, then the quarters will take the beat instead of eights. One may think that is slowing the song, but actually the pace of the song is the same because of the way we are computing the values of the notes (section 5.2.1).

Anacrusis is the phenomenon where the first notes from a song do not form a whole bar. Since this first bar does not contain all the beats required to form a whole bar, the anacrusis maximum length is the numerator of the time signature minus one. In order to find whether in a song exists such a phenomenon or not, we shift the ASM for all possible values of anacrusis multiplied by 64 (for the same reasons we did it for the bar length) and check if we obtain a better SM. If this is the case, then we have an anacrusis with the number of beats correlated to the highest SM.

Another method worth mentioning is based on the detection of the accent on the begging of a bar. Usually, the first note of a bar is more accentuated than the others, thus the amplitude is higher. A method for finding the beginning of a bar is described in the paper written by A.P. Klapuri, A.J. Eronen and J.T. Astola [KEA06]. We did not implement this approach due to the lack of data (only sophisticated songs have such accents).

Machine Learning approach

Our algorithm takes as input a song and outputs two sequences: the beats and the downbeats. The output sequences consist of zeros and ones, where ones mark the presence of the beats/downbeats. The output sequence is not at the same samplerate as the input one, as we consider we do not need such a high rate. More exactly, we use an output samplerate of 100. A graphical representation of our architecture can be seen in Fig. 5.12.

Figure 5.12: Architecture

First, we convert the audio from raw wave form to mel-spectrogram. This type of spectrogram was proposed to better reflect the differences of the sound as perceived by the human ear. It groups the frequencies in a logarithmic scale as opposed to the linear one which is usually employed by the spectrograms.

The mel-spectrogram is passed to the neural network. The first part is common for both tasks (beat, downbeat) - backbone. Our backbone is a bidirectional RNN that uses LSTM as hidden units. We have three layers, each having 128 units for each direction.

For the backbone we use dropout with a chance of 20%. Dropout is a regularization technique that helps neural networks learn. During training, part of the outputs of a layer are masked for the following layer.

After the backbone we have an upsampling layer. We need this in order to have an output that has a necessary length. The output length needs to be correlated to the length of the input song fragment and the output samplerate.

Next we have the parts that are dedicated to the beats and downbeats - the heads. Both heads have the same architecture:

1. convolutional with 50 filters and width 51
2. batch normalization
3. ReLU
4. convolutional with 10 filters and width 51
5. batch normalization
6. ReLU
7. convolutional with 2 filters and width 51

The last layer will have two values for each position in the sequence, one being the score for having the beat/downbeat and the second for not having it. The maximum score is used to decide. To preserve the length of the sequence, the convolutional layers use zero-padding.

During training, the network learns to create the appropriate beats and downbeats sequences which need to be as similar as possible to the ground truth ones. An example graphical representation of the results for a song fragment are presented in Fig. 5.13. It includes the input mel-spectrogram, the predicted beats and downbeats positions, and the ground truth beats and downbeats.

For training the network, we used the Ballroom dataset [GKD+06], [KBW13].

We extract the positions of the beats and downbeats and count how many beats are in between two consecutive downbeats. We compute the median over these

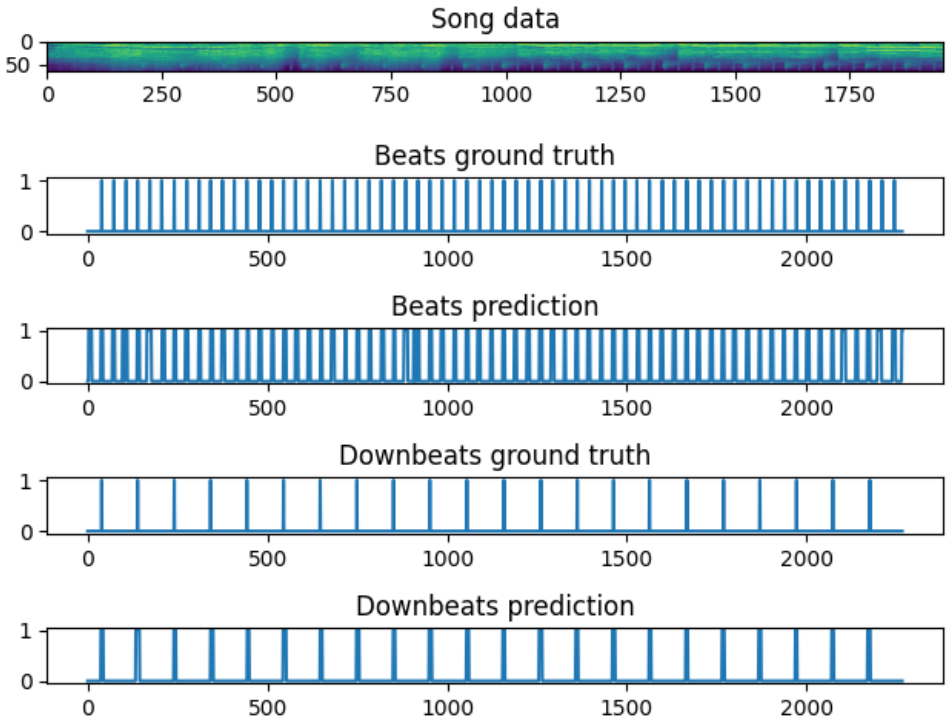

Figure 5.13: Prediction of network

counts and that is the numerator of the time signature. The denominator is computed in the same manner as we did in the Audio Similarity Matrix approach.

5.3 Software design

5.3.1 Technologies

Since the application is in charge with many problems, hence functionalities, the technologies used on both front-end and back-end needed to be new, efficient and reliable.

Front-end

For the Graphical User Interface (GUI) part, the Angular framework is used (Fig. 5.14). It is based on the TypeScript language which actually is an extension of the JavaScript language. Angular is created by the Google company and is one of the most popular web-frameworks. AngularJS is its ancestor which serves the same purpose but it is based on JavaScript. The same team which developed AngularJS also created Angular. Its initial release was in 2016, but its popularity and usage is growing since then, hence its highest peak cannot be established yet.

[1]https://en.wikipedia.org/wiki/Angular_%28web_framework%29

Figure 5.14: Angular logo [1]

This framework helps the developers create not only web-applications, but also mobile and desktop applications. Other frameworks, such as Karma for unit tests, Ionic for mobile applications, can easily be integrated because of its extensibility. Angular has its own tags and attributes that help code writing and development much easier. Not only that the number of tags increases in comparison with the common HTML tags because of the Angular framework, but the latter also allows you to create your own tags with your own attributes and functions. Integration and development have never been easier.

On top of all this, there is also an User Interface library called Angular Material that can be used by the developers in their Angular projects in order to make the GUI more beautiful, modern and intuitive. It has components like:

- Button
- Chips
- Checkbox
- Datepicker
- Dialog
- Form field
- Icon
- Menu
- Progress spinner/bar

Many of the Angular Material components are used in this application, because they beautify it and make it easier for it to be extended in the future.

In order to be able to display a MusicXML file that is returned by the back-end and also handle it in different manners, another library is used called OpenSheetMusicDisplay (OSMD). It is free-source library based on the JavaScript language. The functionalities used in our application from OSMD are:

- load() and render() for displaying the music score from the MusicXML file
- cursor.show(), cursor.hide(), cursor.next() to handle the cursor when the song is played
- AudioPlayer class which parses the music sheet and produces the sound corresponding to the score

For creating a PDF file and downloading it, another two libraries were used: jspdf and svg2pdf. The latter helps us converting a SVG element to a PDF and the jspdf serves for creating the PDF and downloading it.

Back-end

Figure 5.15: Python logo [2]

For the back-end application the Python language was chosen (Fig. 5.15). This one is a high-level programming-language which highlights code reliability and uses indentation for structuring and delimitation. It is one of the most popular programming language which can serve for web applications and even for complex designs and maintenance. It first appeared in 1991, but its popularity and usage continue to grow, as more and more useful and important libraries are added to it. NumPy, SciPy and Matplotlib are some of the libraries that make the use of the Python language very effective in scientific computing. Even the machine learning frameworks such as PyTorch uses at the core the NumPy library.

NumPy (Fig. 5.16) is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. We used it a lot because working with arrays in many ways and of many types is what we do and it also has methods regarding signal processing.

[2]https://en.wikipedia.org/wiki/Python_%28programming_language%29
[3]https://numpy.org/

Figure 5.16: NumPy logo [3]

Figure 5.17: SciPy logo [4]

SciPy (Fig. 5.17) is a free and open-source Python library used for scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering.

Figure 5.18: Matplotlib logo [5]

Matplotlib (Fig. 5.18) is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK. This library helps us visualise at each step the information we have until now and understand much better the endless arrays.

Aubio is a library that contains a set of tools for audio and musical analysis. At the core, this one also uses NumPy in order to make the computations more efficient. Some of the functionalities that are offered by the Aubio library are: reading audio from file, detection of onsets (note attacks), fundamental frequency estimation, beat detection. The one that was used in this application is the beat detection function that returns the moments where a beat occurs.

Flask (Fig. 5.19) is a web framework written in Python which is used in this ap-

[4]https://www.fullstackpython.com/scipy-numpy.html
[5]https://matplotlib.org/3.5.0/gallery/misc/logos2.html
[6]https://medium.com/analytics-vidhya/easiest-python-and-flask-tutorial-f1ba963281a

Figure 5.19: Flask logo [6]

plication for the connection with the front-end (Angular module). It is also classified as a microframework because it does not require other libraries, making it simple and scalable. It does not offer database support, but for achieving this, other extensions can be added. Flask is very Pythonic, hence it is explicit, increases readability and with very few lines of code one can create relevant things.

5.3.2 Functionalities

The web-application presented in this paper is a complex one, because it has important features in the back-end module, but also on the front-end. It is all of the functionalities that make the application complete.

Front-end

The application contains one page developed with the Angular framework, therefore it requires a web browser to run on. This page has the following features:

- Change the algorithms that compute the time signature and tempo (Fig. 5.20, Fig. 5.21)

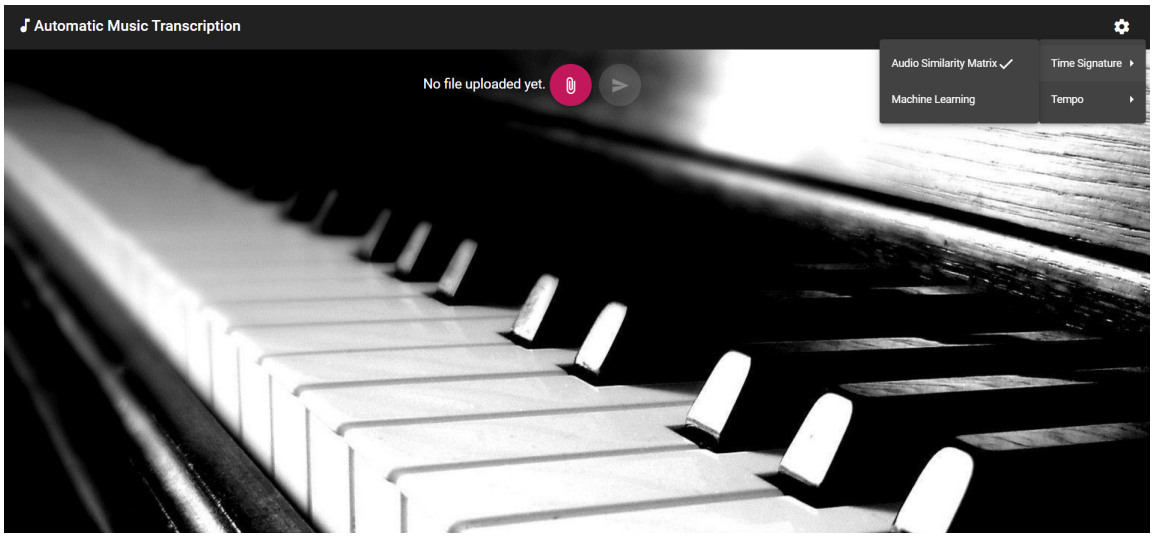

Figure 5.20: Automatic Music Transcription Application - Time Signature algorithm

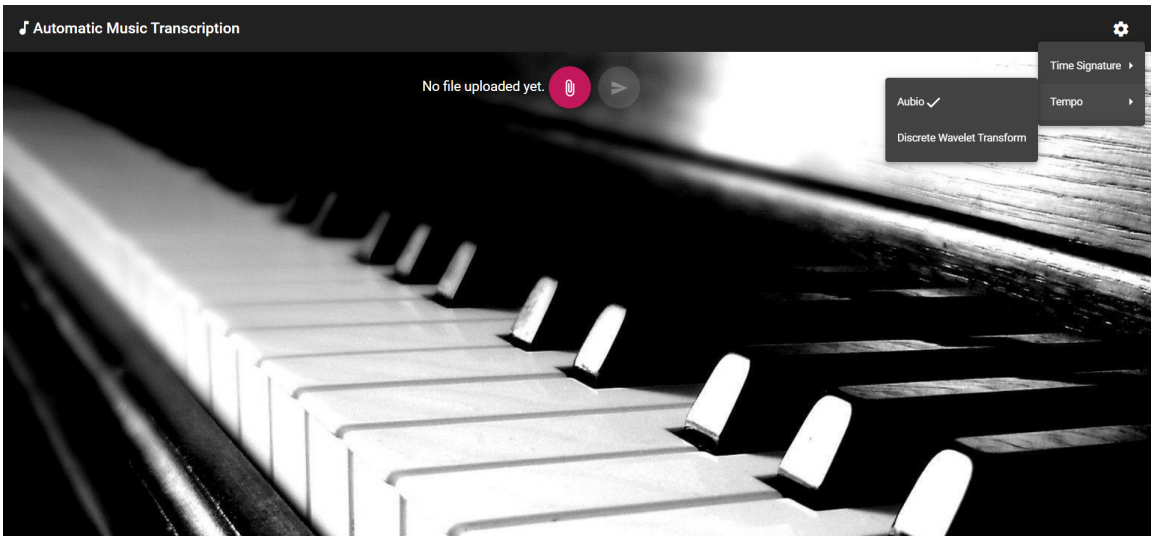

Figure 5.21: Automatic Music Transcription Application - Tempo algorithm

- Select a WAV file by pressing the button represented with the ”attach file” icon (Fig. 5.22)

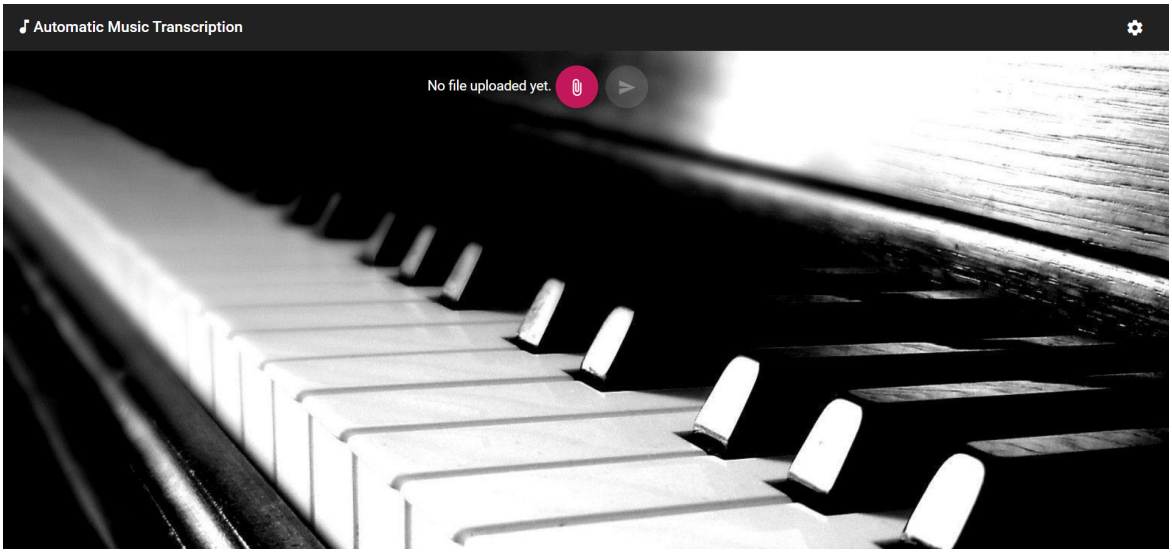

Figure 5.22: Automatic Music Transcription Application - No file selected

- After uploading a WAV file, the option ”send” is made available. If the user presses the button, then the uploaded file will be send to the back-end where it will be processed. While the back-end handles the file, the user will be kept waiting, being signaled by a progress spinner that his/her song is being converted into a music sheet (Fig. 5.23)
- When the computations are done, the result expressed as a MusicXML string will be sent as a response. This one it will be then shown in the GUI as a regular music score with the help of the OSMD library.

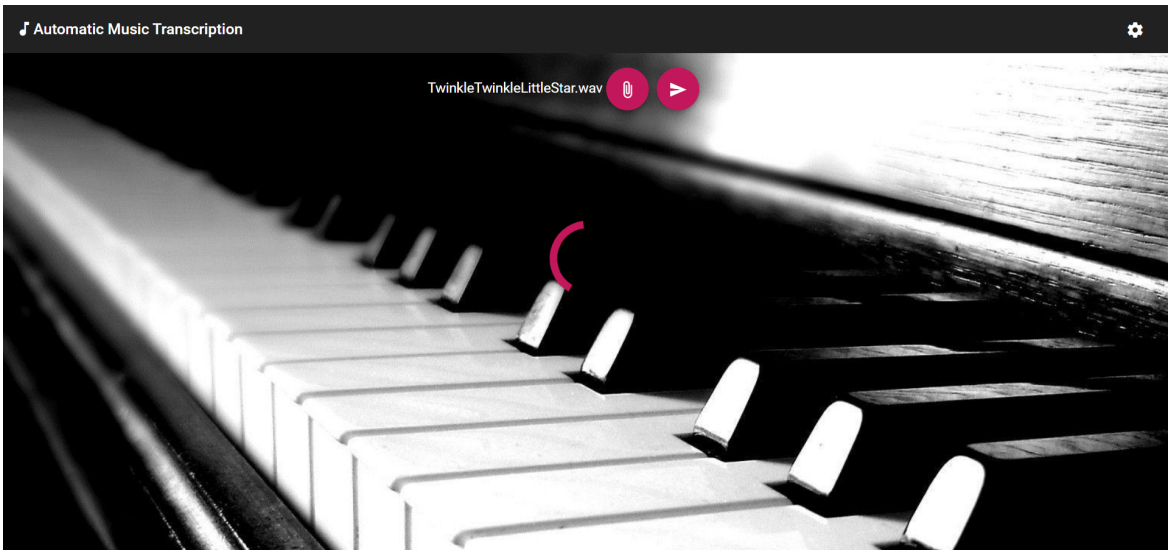

Figure 5.23: Automatic Music Transcription Application - Loading

- At the same time when the music sheet is shown, 5 other buttons appear in the GUI: play, pause, stop, show/hide, download (Fig. 5.24)

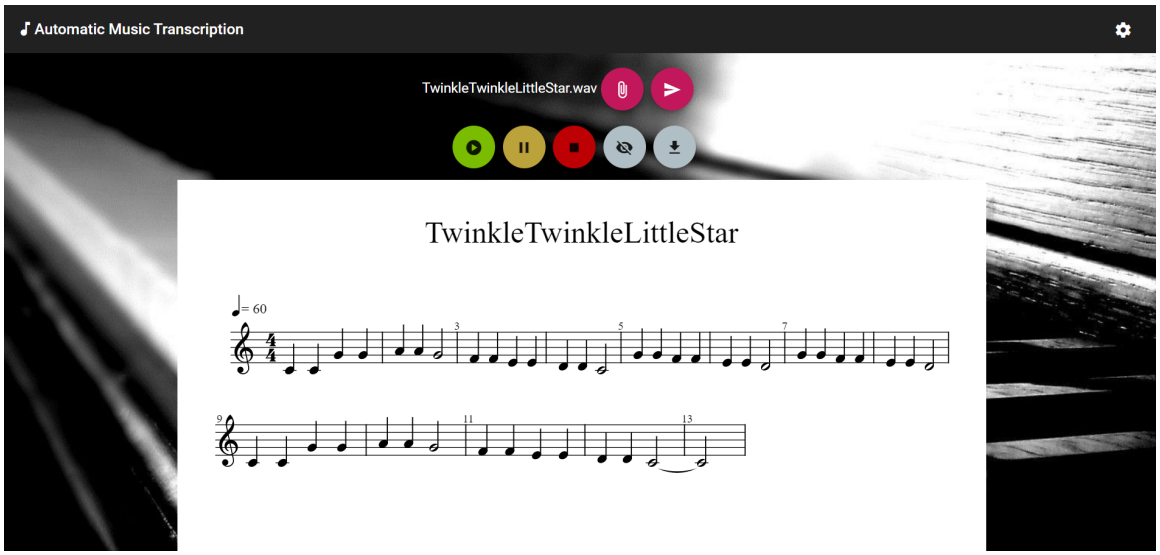

Figure 5.24: Automatic Music Transcription Application - Music Sheet

- The play, pause, stop buttons are used for handling the user's wish of playing or not the notes from the music score
- The show/hide button has the functionality of making the cursor visible or not (Fig. 5.25). The cursor shows the note that is being played, hence it helps the user track the notes on the music sheet and correlate a sound and/or a duration with the note symbol
- The download button triggers a function that converts the tag containing the

information about the music sheet into a SVG, then into a PDF, which is later saved on the user's computer.

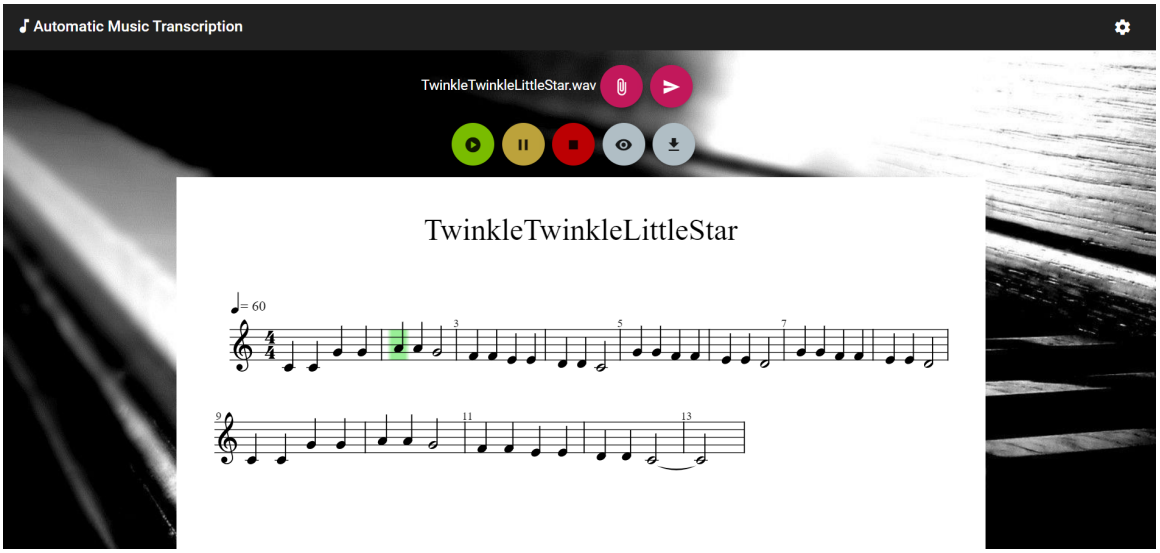

Figure 5.25: Automatic Music Transcription Application - Music Sheet with Cursor on it

For the connection with the back-end application, an Angular service is created and injected in the classes that is needed. The HTTP requests are performed with the help of HttpClient Angular's class which was injected in our service. The methods get, post, put, delete of this class return an Observable which contains the response from the back-end. The other functions that wait for responses have to subscribe to this Observables in order to retrieve the content.

Back-end

On the back-end part, the Python application has a module containing the Flask components, meaning the functions that receive requests and send responses to the Angular application 5.26. These functions have instances of the classes that belong to the computational module. The latter is charged with the most complex part of the application, namely, transforming an audio file into a music score. The application also contains a package entitled "ml" which is in charge with the machine learning part.

The computational module has a NoteDetector class which contains the main method that handles the processing of the WAV file received from the front-end. When the filename arrives in the "computeForASong" method, the process goes as follows:

1. The data is read from the file

```python
@app.route("/send_wav", methods=['POST'])
def send_wav():
    if request.method == 'POST':
        f = request.files['file']
        f.save(os.path.join(uploads_dir, secure_filename(f.filename)))
        time_signature_alg = request.form.get('ts')
        tempo_alg = request.form.get('tempo')
        noteDetector = NoteDetector()
        music_sheet = noteDetector.computeForASong(uploads_dir + '/' + f.filename, time_signature_alg, tempo_alg)
        return music_sheet
```

Figure 5.26: Flask - POST method

2. The bpm (Beats Per Minute) value is calculated in the ”Tempo” class or ”TempoDWT” class, depending on the user’s selection
3. The array of data is transformed using the RMS formula 5.1
4. The thresholds for the notes (formula 5.2) and for the silence are computed
5. The minimum duration of silence is extracted from the formula 3.1
6. The intervals of silence are found and therefore the sequences of notes
7. The time signature is detected as presented in section 5.2.3 in the ”TimeSignature” class or the ”SongMeasureDetector” class, depending on the user’s selection
8. Since the data is represented in samples, it is necessary that the duration of a sample to be found, in order to find the duration of a note (Fig. 5.27)

```python
def get_duration_of_one_sample(self, data, samplerate):
    duration_of_song = data.shape[0] / samplerate
    duration_of_one_pos = duration_of_song / data.shape[0]
    return duration_of_one_pos
```

Figure 5.27: Method for finding the duration of one sample

9. Having this information, the duration in seconds of each note is computed
10. Passing the notes intervals to another function, the fundamental frequency of each note is determined as described previously (section 5.2.1). The method of finding the peak that represents fundamental frequency can be observed in Fig. 5.28
11. After finding the frequency of the note, we can correlate this one with an actual name taken from the equal-tempered scale. This task is handled by the ”Note” class.

```
def my_fourth_peak(self, peaks, freq, init_ok_range=0.15, nr_confirm=4):
    for starting_position in range(len(peaks[0])):
        mean_freq = freq[peaks[0][starting_position]]
        n_points = 1
        ok_range = init_ok_range
        mul = 2
        while True:
            mid = mean_freq * mul
            dist = max(mean_freq * ok_range, 2)
            start = mid - dist
            end = mid + dist
            if end > freq[peaks[0][-1]]:
                return mean_freq
            found_peak = None
            for i in range(starting_position+1, len(peaks[0])):
                peak = peaks[0][i]
                if start <= freq[peak] <= end:
                    found_peak = freq[peak]
            if found_peak is None:
                break
            else:
                mean_freq = (mean_freq * n_points + found_peak / mul) / (n_points + 1)
                n_points += 1
                mul += 1
                ok_range /= 1.1
                if n_points >= nr_confirm:
                    return mean_freq
```

Figure 5.28: Method for finding the peak corresponding to the fundamental frequency

12. After having all this information, we pass it all to the ”GenerateMusicSheet” class which will create the MusicXML file, hence the music score

A visual schema of how the classes and methods are organized, can be seen in the Unified Modeling Language (UML) diagram represented in Fig. 5.29.

5.4 Evaluation

In this section, we will present the experiments that we have conducted, testing different functionalities of our application.

In some cases, the algorithm that detects the intervals of notes fails, because the distance between two notes is too small above the threshold, so the algorithm consider them as one note, fact which also leads to bad fundamental frequency detection.

The table 5.2 gives information about the durations of the notes from the Happy Birthday song.

In the table 5.3 we noted the experiments made for the fundamental frequency algorithm on the Happy Birthday song.

In the 5.4 table we made a comparison between our tempo values using the Aubio beat-tracking, an online tool (Tunebat [Tem]) which also makes BPM detec-

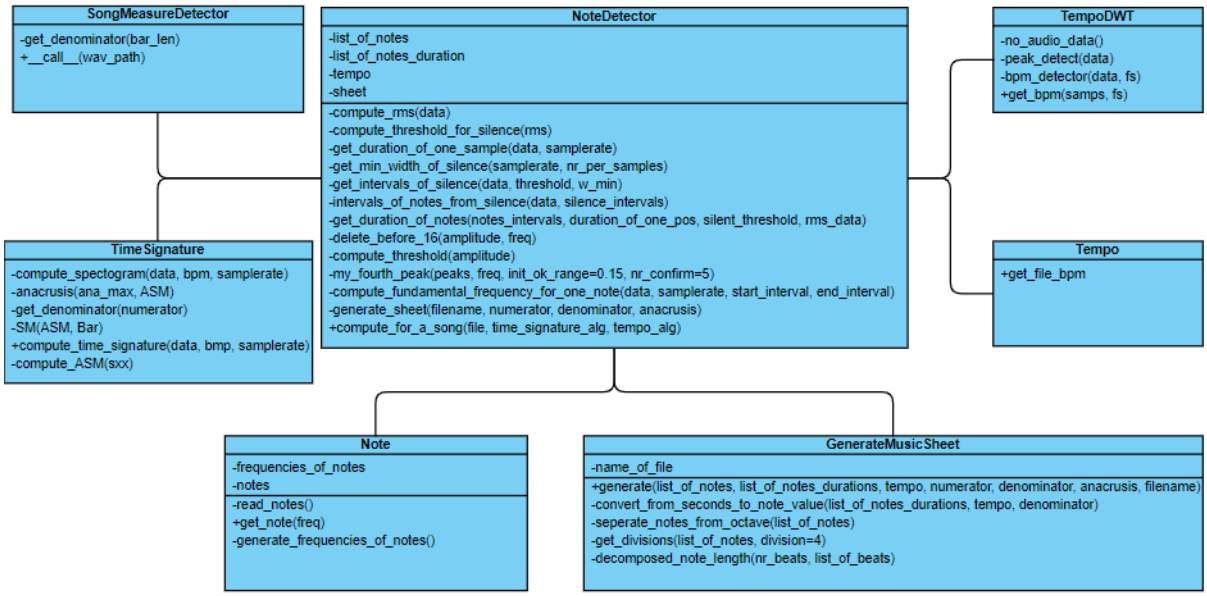

Figure 5.29: UML Diagram

Table 5.1: Number of notes

Song	No. of notes detected	Actual number of notes
Happy Birthday	25	25
Twinkle Twinkle Little Star	42	42
Old McDonald Had a Farm	59	59
If You're Happy And You Know It	134	134
Silent Night	46	46
Merry Christmas	48	48
Fly Me To The Moon	79	82

tion and the method implemented by us using Discrete Wavelet Transform (DWT) [TEC01].

In the table 5.5 we present the time signatures obtained using the Audio Similarity Matrix and Machine Learning algorithm.

The time signature detected for the Happy Birthday song is 6/8, but it is not wrong since it can be easily converted to 3/4 just by doubling the tempo. The idea is that the transcription of a song to a music score does not have only one correct answer, but multiple, as long as the final sound is the same with the original.

We also made comparisons between our generated music sheet (Fig. 5.30), another generated music score (Fig. 5.31) from a free online application called "Melody Scanner" [Mel] and the original music sheet (Fig. 5.32). The music sheets look very

Table 5.2: Durations of notes

Detected duration of note	Actual duration of note	Difference (abs)	Difference (%)
0.7573	0.75	0.0073	0.973
0.768	0.75	0.018	2.4
1.4826	1.5	0.0174	1.16
1.4826	1.5	0.0174	1.16
1.4826	1.5	0.0174	1.16
3.072	3.0	0.072	2.4
0.7146	0.75	0.0354	4.72
0.7573	0.75	0.0073	0.973
1.4826	1.5	0.0174	1.16
1.5253	1.5	0.0253	1.68
1.4613	1.5	0.0387	2.58
2.9973	3.0	0.0027	0.09
0.7573	0.75	0.0073	0.973
1.5253	1.5	0.0253	1.686
1.4613	1.5	0.0387	2.58
1.493	1.5	0.007	0.46
1.504	1.5	0.004	0.26
2.28	3.0	0.72	24
0.704	0.75	0.046	6.13
0.7573	0.75	0.0073	0.973
1.546	1.5	0.046	3.06
1.472	1.5	0.028	1.86
1.493	1.5	0.008	0.53
6.592	6.0	0.592	9.86

Table 5.3: Fundamental frequency

Detected note	Actual note	Detected fundamental freq	Actual fundamental freq
G3	G3	196.3	196.00
G3	G3	195.95	196.00
A3	A3	220.44	220.00
G3	G3	196.72	196.00
C4	C4	261.57	261.63
B3	B3	247.06	246.94
G3	G3	196.72	196.00
G3	G3	196.371	196.00
A3	A3	220.185	220.00
G3	G3	196.104	196.00
D4	D4	293.13	293.66
C4	C4	261.79	261.63
G3	G3	196.30	196.00
G3	G3	196.02	196.00
G4	G4	392.119	392.00
E4	E4	329.687	329.63
C4	C4	261.793	261.63
B3	B3	246.90	246.94
A3	A3	220.4	220.0
F4	F4	348.687	349.23
F4	F4	349.207	349.23
E4	E4	331.665	329.63
C4	C4	261.53	261.63
D4	D4	293.60	293.66
C4	C4	261.69	261.63

Table 5.4: Tempo

Song	Our tempo prediction	Tunebat prediction	DWT prediction
Happy Birthday	81	80	93
Twinkle Twinkle Little Star	60	120	119
Old McDonald Had A Farm	72	72	143
If You're Happy And You Know It	80	80	160
Silent Night	143	71	94
Merry Christmas	101	100	100
As It Was	142	140	137
Havana	127	126	125

Table 5.5: Time Signature

Song	Detected time signature (ASM)	Detected time signature (ML)	Real time signature	Detected anacrusis (ASM)	Real anacrusis
Happy Birthday	6/8	4/4	3/4	4	1
Twinkle Twinkle Little Star	4/4	3/4	4/4	0	0
Silent Night	4/4	4/4	3/4	1	0
As It Was	4/4	3/4	4/4	0	0
The Second Waltz	3/4	3/4	3/4	0	0
Believer	4/4	4/4	4/4	0	0
Let It Be	4/4	4/4	4/4	1	1

similar and by analysing them, one can observe that the time signature and notes durations are correct in both of the generated scores (Fig. 5.30, Fig. 5.31).

The only difference between the music sheets is that in the music score generated by the ”Melody Scanner” application some notes are wrong (with one octave higher) and one is not detected.

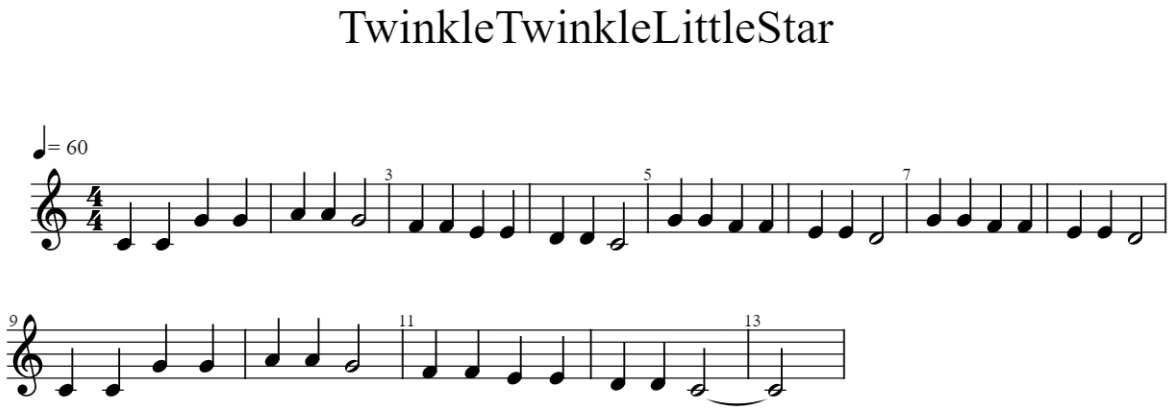

Figure 5.30: Our music score - Twinkle Twinkle Little Star

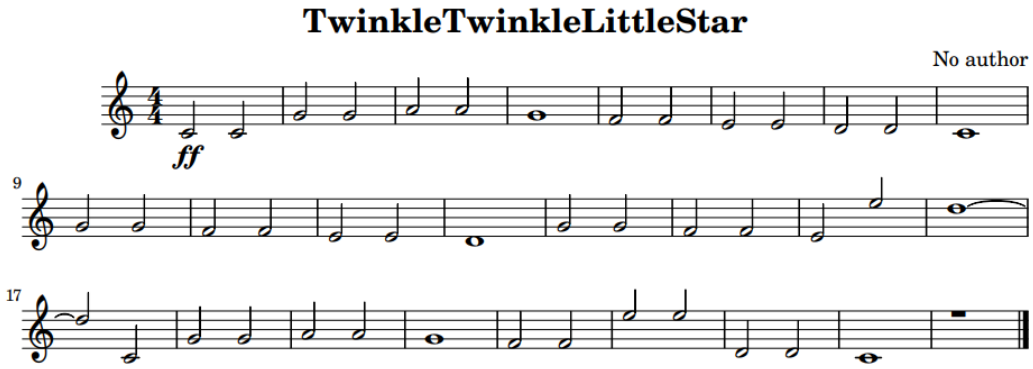

Figure 5.31: Another music score - Twinkle Twinkle Little Star

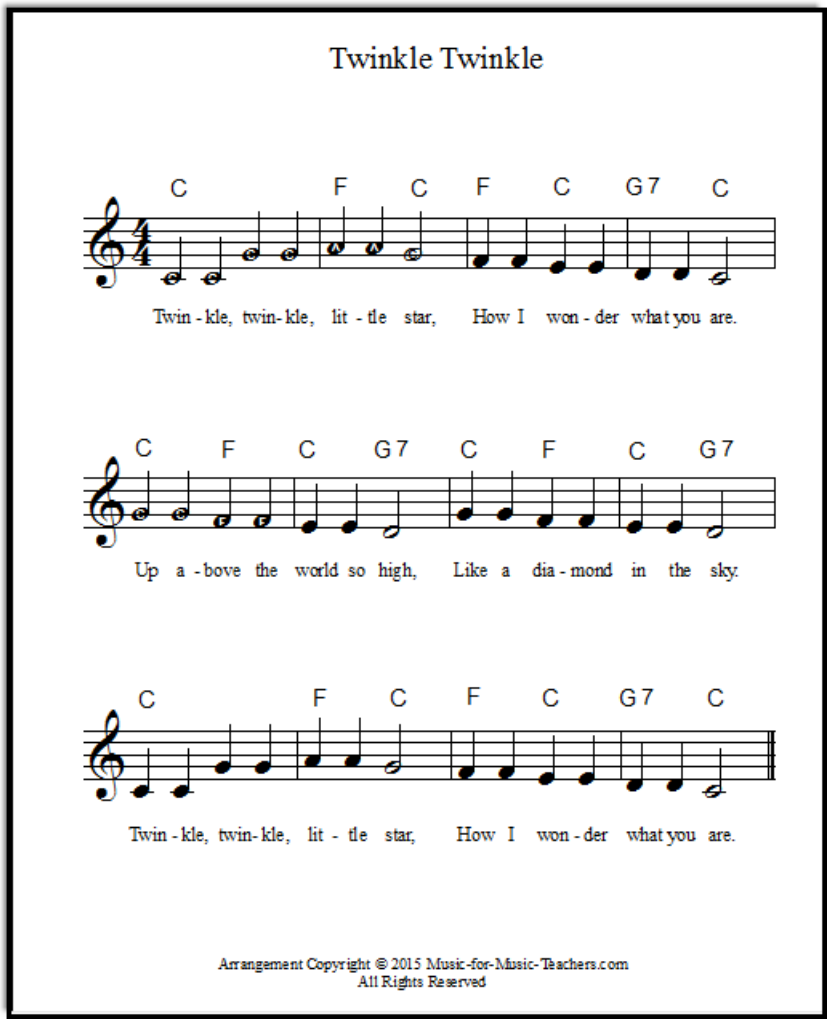

Figure 5.32: Original music score - Twinkle Twinkle Little Star

Chapter 6

Conclusions

An automatic music transcription application requires many functionalities that would handle all the possible characteristics of a song. As music is not an exact science, it is more difficult to come up with a solution that could have multiple correct answers.

Our application can deal with simple songs, with one note at a time and the algorithm for the time signature detection works better when applied on musical pieces that have repetitive parts. Therefore, we propose that our future work would solve these issues, meaning that we will be able to process chords, determine the instruments from a song and make a music sheet for each of them. We plan on finding an algorithm that would detect the beats and down-beats, hence the detection of the tempo and time signature will be more accurate. A testing tool, other than our human ear, would be very useful in checking if the result provided by the application is getting close to reality.

Even though the application needs to improve its flexibility and accuracy, it serves the purpose it was designed for: convert a song into a music sheet. Complex problems with even more complex solutions were approached in this paper, implemented with modern technologies and altogether starting the creation of a new automatic music transcription application.

Bibliography

[AN17] Muhammad Asadullah and Shibli Nisar. A silence removal and endpoint detection approach for speech processing. *Sarhad University International Journal of Basic and Applied Sciences*, 4(1):10–15, 2017.

[BDA+05] Juan Pablo Bello, Laurent Daudet, Samer Abdallah, Chris Duxbury, Mike Davies, and Mark B Sandler. A tutorial on onset detection in music signals. *IEEE Transactions on speech and audio processing*, 13(5):1035–1047, 2005.

[BDK19] Sebastian Böck, Matthew E. P. Davies, and Peter Knees. Multi-task learning of tempo and beat: Learning one to improve the other. In *ISMIR*, 2019.

[CG07] Eugene Coyle and Mikel Gainza. Time signature detection by using a multi-resolution audio similarity matrix. In *Audio Engineering Society Convention 122*. Audio Engineering Society, 2007.

[Foo99] Jonathan Foote. Visualizing music and audio using self-similarity. In *Proceedings of the seventh ACM international conference on Multimedia (Part 1)*, pages 77–80, 1999.

[GKD+06] Fabien Gouyon, Anssi Klapuri, Simon Dixon, Miguel Alonso, George Tzanetakis, Christian Uhle, and Pedro Cano. An experimental comparison of audio tempo induction algorithms. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(5):1832–1844, 2006.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[KBW13] Florian Krebs, Sebastian Böck, and Gerhard Widmer. Rhythmic pattern modeling for beat and downbeat tracking in musical audio. In *Ismir*, pages 227–232. Citeseer, 2013.

[KEA06] A.P. Klapuri, A.J. Eronen, and J.T. Astola. Analysis of the meter of acoustic musical signals. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(1):342–355, 2006.

[KL22] Kathryn Kovitvongsa and Phillip Lobel. Convenient fish acoustic data collection in the digital age. 06 2022.

[LBE15] Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

[Mel] Melodyscanner. `https://melodyscanner.com/App#/`. (accessed May 10, 2022).

[Not] Notesfrequency. `https://pages.mtu.edu/~suits/NoteFreqCalcs.html`. (accessed March 11, 2022).

[PG15] Jay K Patel and ES Gopi. Musical notes identification using digital signal processing. *Procedia Computer Science*, 57:876–884, 2015.

[Sun19] Jian Sun. Research on vocal sounding based on spectrum image analysis. *EURASIP Journal on Image and Video Processing*, 2019, 01 2019.

[TEC01] George Tzanetakis, Georg Essl, and Perry Cook. Audio analysis using the discrete wavelet transform. In *Proc. conf. in acoustics and music theory applications*, volume 66. Citeseer, 2001.

[Tem] Tempodetection. `https://tunebat.com/Analyzer`. (accessed May 10, 2022).

[Wic10] Mladen Victor Wickerhauser. *Mathematics for Multimedia*. Birkhäuser Boston, MA, 2010.