Automated Music Transcription

using Convolutional Neural Networks

Cota Ionas-Calin

A thesis presented for the degree of
Bachelor of Computer Science

Computer Science
Babes Bolyai University
Romania
4/20/2019

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Music is universal and its significance is nothing to mess about. Every known civilization has it's own style of music. From baroque, classical, opera, jazz, traditional folk, rock, rap or contemporary pop music, the sharing of music provides endless joy for humans.

Music transcription is defined as the task of converting music from sound into a written, abstract notation. It is the inverse operation of music performance, which often involves a performer reading music notation and producing sound waves with the help of an instrument or their voice. Because music is a universal language people around the globe can share music with each other surpassing the language barrier.

Manual music transcription is a task difficult enough that even the best musicians struggle to achieve 100% accuracy. It takes a lot of time and practice to learn and even more to master. In order to facilitate the process and make transcription available to everyone, people tried to figure out ways to automate the process. It was at this moment, in 1977, that Automatic music transcription (AMT) was born.

For the past decades, this field of computer science research has been developing, even though it still has numerous unsolved problems. Every year shows new research with improved algorithms for various subtasks of AMT.

The goal of the present thesis is to introduce the field of automatic music transcription and provide a deep learning approach to pitch estimation using convolutional neural networks (CNNs).

Theia7, the proposed solution, is a lightweight application that aims to transcribe monophonic as well as polyphonic audio with the help of a CNN. For this task, we'll use the MAPS dataset [2] in order to train the model. In order to avoid the noise of .mp3 and .wav files, only MIDI files will be used to convert and predict. The input and output will be processed in a heuristic manner, as no digital signal processing background is available to the author. Given a song Theia7 will generate its music sheet as well as a .wav version of the prediction.

Chapter 2

Theoretical Background

2.1 Introduction to Deep Learning

Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to effectively perform a specific task without using explicit instructions, relying on patterns and inference instead. Machine learning algorithms build a mathematical model of sample data, known as training data, in order to make predictions or decisions without being explicitly programmed to perform the task. Machine learning algorithms are used in a wide variety of applications, such as email filtering [3] and computer vision [4] [5], where it is infeasible to develop an algorithm of specific instructions for performing the task. The representation of data fed into a ML algorithm plays a major role, as it affects the algorithm's ability to efficiently extract signals and make decisions. Thus, it is important to carefully select the information included in such a representation. Formally, the representation is composed of multiple features extracted from raw data. The process of creating new features requires good and intuitive understanding of the data at hand, becoming incrementally time-consuming with the sophistication of the new features. Thus, the biggest challenge of handcrafted features is deciding which features are important and relevant to the problem [6].

2.2 Neural Networks

This section introduces the main concepts related to neural networks. Neural networks have been around since the 1940s and could initially handle only one hidden layer. But with the development of technologies and hardware, it became possible to build deeper, more effective architectures, which lead to deep learning as we know it today.

2.2.1 Brief History

At first, neural networks were inspired by the functioning of the biological brain, which is why deep learning is also called artificial neural networks (ANNs) [6]. In biology, a neuron is the cell that receives, processes and transmits information to other neurons through connections called synapses [7]. On the other hand, artificial neurons are defined as computational units (usually mathematical functions) that take one or more inputs and generate an output.

McCulloch and Pits designed an initial version of the neuron as a linear model in 1943, aiming to replicate brain function [8]:

$$f(x, w) = x_1 * w_1 + x_2 * w_2 + ... + x_n * w_n \tag{2.1}$$

where $x_1, ..., x_n$ are the input values and $w_1, ..., w_2$ is a set of hand-chosen weights.

2.2.2 Components of an artificial neural network

A simple artificial neural network (ANN) consists of an input layer, hidden layer and output layer, where the values of the hidden layer are used as inputs for the output layer. A network with several layers is known as a deep neural network. Data flows through the neurons of the layer. Each neuron transforms the input it receives and sends it to the next layer. The neurons share the same characteristics regardless of the layer they are part of.

The Neuron, also called a node, is the basic unit of a neural network. Its main components include inputs, weights, activation function and output(s). From a high-level point of view, the inputs are multiplied by weights, then an activation function is applied to the result and finally, another function computes the output [9] [10].

- Weights are defined as adaptive coefficients, whose values are changed during the learning process. They represent the strength of the connection between units. A weight decides how much impact the input will have on the output
- The summation function helps combine the input and weights, before passing the result to the activation function. Denote the input as $X = [x_1, x_2, ...x_n]$ and the weight vector as $W = [w_1, w_2, ...w_n]$.
  The summation function could be defined as the dot product between these two vectors:

  $$X \cdot W = x_1 \cdot w_1 + x_2 \cdot w_2 + ... + x_n \cdot w_n \tag{2.2}$$

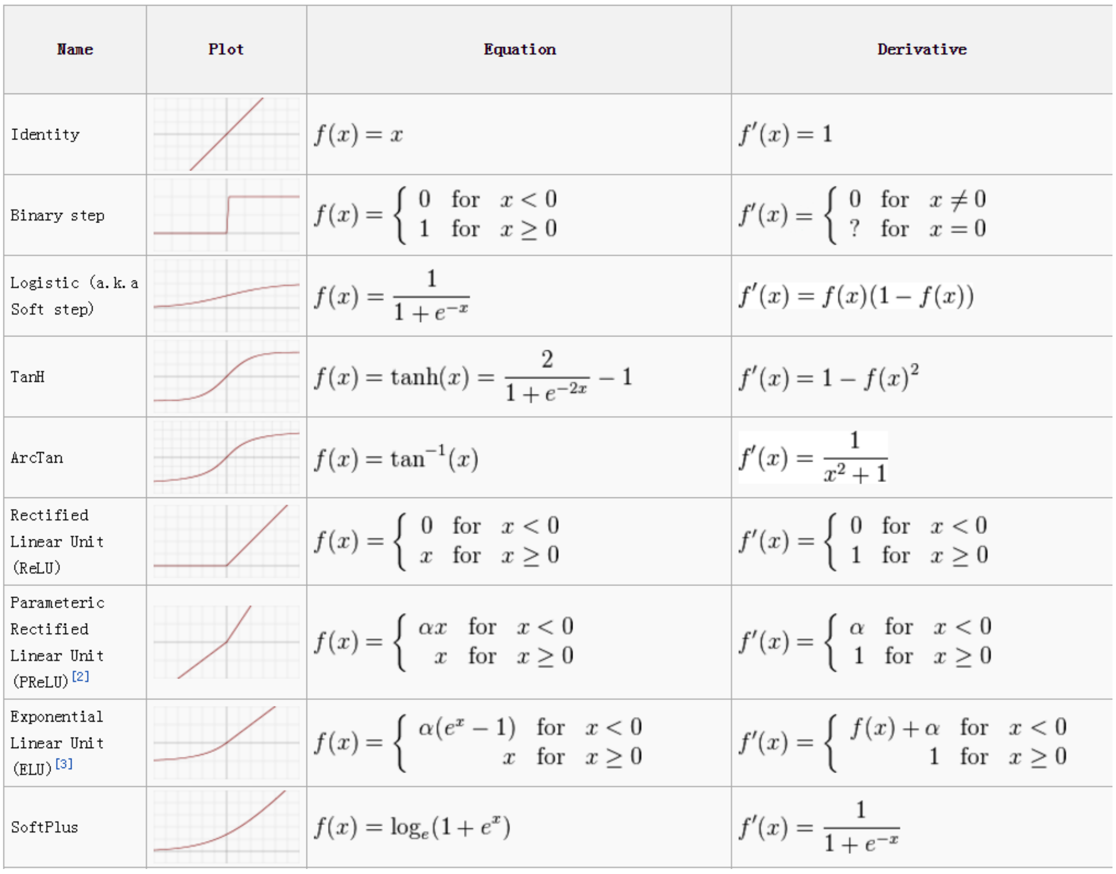
  The summation function could instead compute the minimum, maximum, etc. depending on the designated network architecture. The simplest form of an artificial neuron is a linear function which computes the weighted sum of inputs, to which, optionally, bias can be added:

$$y = \sum_{i=1}^{i=n}(x_i \cdot w_i) + b, \text{ where } b \text{ is the bias, } x_i \in X, w_i \in W \tag{2.3}$$

- The activation function transforms the result of the summation function (usually) in a non-linear way. Typically, it has a squashing effect. It serves as a threshold. It divides the original space into two partitions. Its main purpose is to make the neural network non-linear. We denote the activation function as g

$$y = g(\sum_{i=1}^{i=n}(x_i \cdot w_i) + b), \text{ where } b \text{ is the bias, } x_i \in X, w_i \in W \tag{2.4}$$

Figure 2.1: Common activation functions [11]



Name	**Plot**	**Equation**	**Derivative**
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

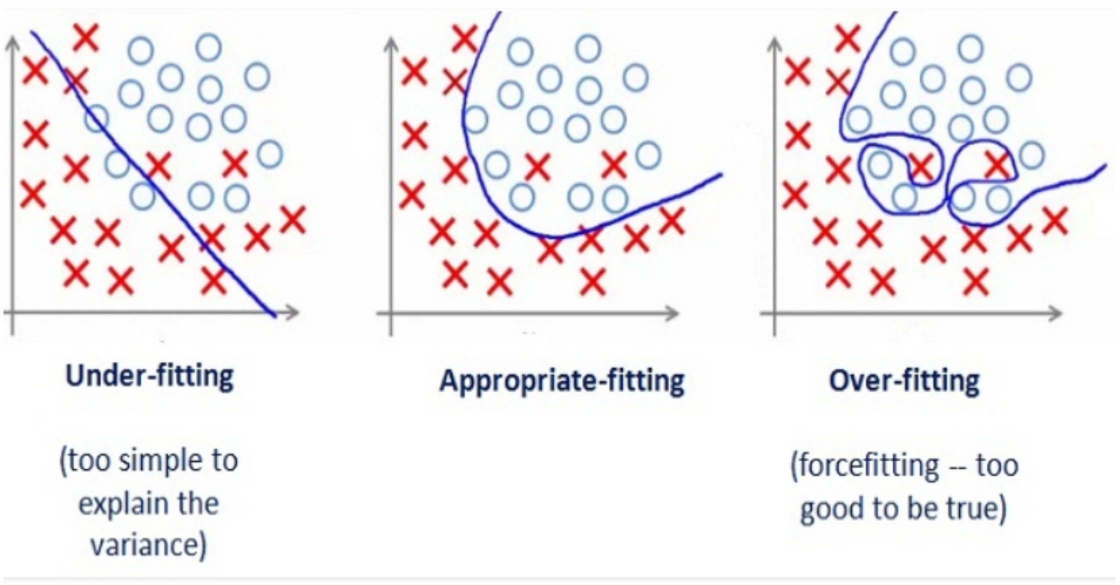
- The output is usually the result of an activation function

2.2.3 Under-fitting and Over-fitting

Neural networks are able to learn complicated non-linear functions to fit any training set. On the downside, this may lead to over-fitting where the neural network learns the training data so well that it is unable to generalize on new, unseen data. This problem can especially occur on datasets with a small amount of data to learn from.

Under-fitting, the counterpart of over-fitting, happens when a machine learning model isn't complex enough to accurately capture relationships between a dataset's features and a target variable. Under-fitted models result in problematic outcomes on new data or data that they were not trained on, and many times perform poorly even on training data.

Figure 2.2: Example of over-fitting and under-fitting [12]

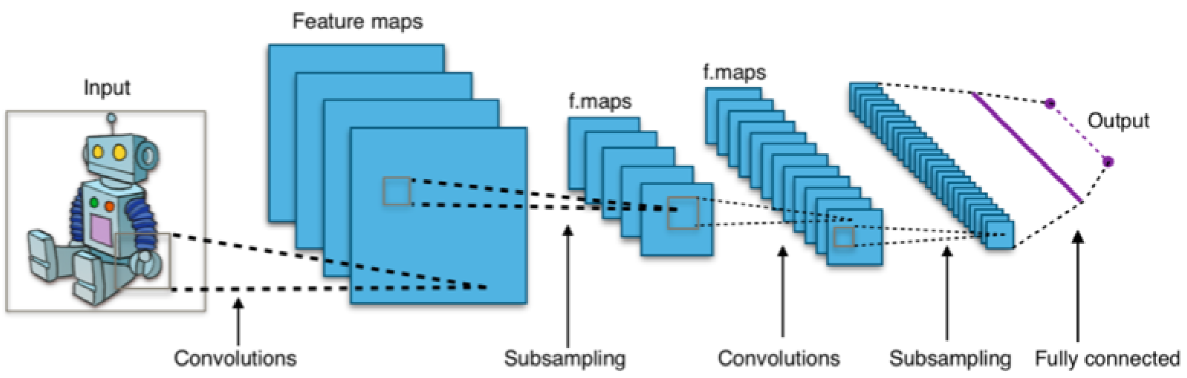

Under-fitting
(too simple to explain the variance)
Appropriate-fitting
Over-fitting
(forcefitting -- too good to be true)

2.2.4 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class of Deep Neural Networks, specialized in analyzing images. Their development was inspired by biological processes. The connectivity pattern between neurons resembles the animal visual cortex [13].

A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers are typically convolutional layers, RELU layers, pooling layers, fully connected layers and normalization layers [15].

Figure 2.3: Typical CNN architecture [14]



- **Convolutional** layers are the core building block of convolutional networks. They are responsible for feature extraction and do most of the computation
- **Pooling** layers reduce the dimension of the data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer
- **Normalization** layers apply a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1
- **Fully connected** layers link every neuron in one layer to every neuron in another layer
- **Dropout** layers have a % chance to deactivate the neurons of a particular layer. This is a technique used to improve over-fitting by improving generalization. Forcing neurons to deactivate forces the network to learn the same "concept" with different neurons. It is mostly used after fully connected or pooling layers

2.3 MIDI format

The Musical Instrument Digital Interface (MIDI) protocol is an industry-standard defined in the early '80s to represent musical information [16]. It is the most spread binary protocol for communication intended to connect electronic musical instruments such as synthesizers, to a computer for the purpose of recording and editing [17]. Unlike other formats (.wav or .mp3), MIDI files don't contain any audio by themselves, only the instructions to reproduce it. It allows sending of messages over 16 channels, each of them could be a different instrument.

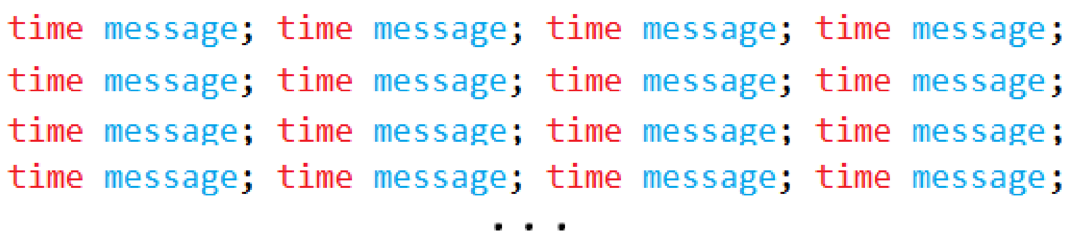
2.3.1 Structure of the MIDI file

At the top level, MIDI files contain events. Each event consists of two parts: MIDI time and MIDI message. They follow each other in a successive manner as shown in Figure 2.4. The time value represents the time to wait before playing the next message. This method of specification is called delta time($\Delta time$) (Table 2.1).

Table 2.1: Understanding event time based on delta time

ΔT	Elapsed time until the event
$t_1 = \Delta T_1 = T_1 - 0$	$T_1 = t_1$
$t_2 = \Delta T_2 = T_2 - T_1$	$T_2 = t_1 + t_2$
$t_3 = \Delta T_3 = T_3 - T_2$	$T_3 = t_1 + t_2 + t_3$
$t_4 = \Delta T_4 = T_4 - T_3$	$T_4 = t_1 + t_2 + t_3 + t_4$
...	...

Figure 2.4: MIDI file messages



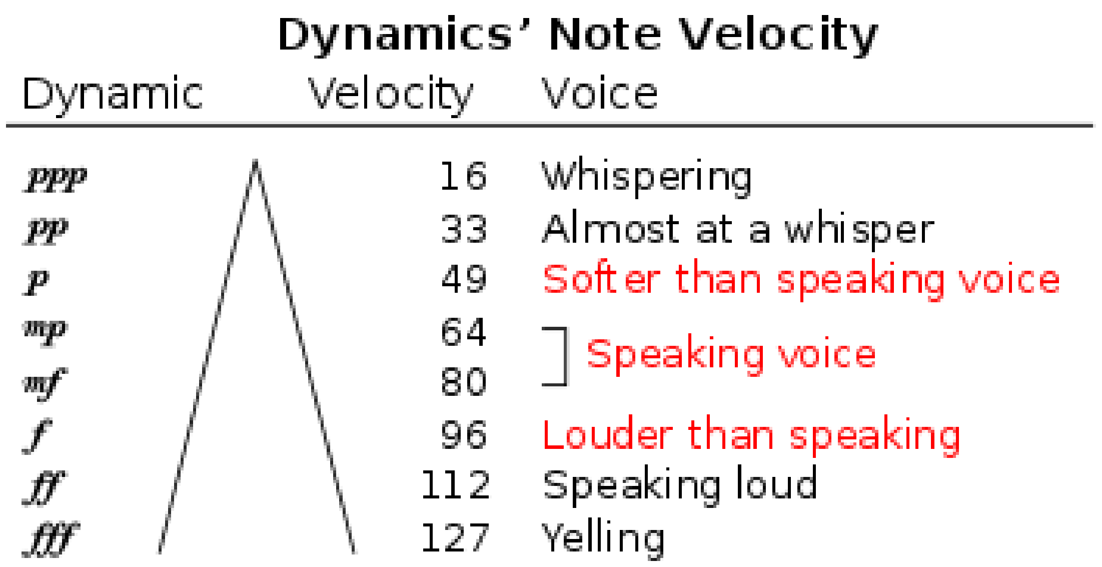
```
time message; time message; time message; time message;
time message; time message; time message; time message;
time message; time message; time message; time message;
time message; time message; time message; time message;
...
```

2.3.2 MIDI messages

The main messages of a MIDI file are **note on** and **note off**. The former is sent when a key is pressed on the music keyboard. It contains parameters such as the pitch and velocity (i.e. intensity) of the note. When a synthesizer receives this message it starts playing that note until a **note off** event arrives for that pitch.

The velocity ranges between 1 and 127. It corresponds to the nuances (i.e. dynamics) found in music notation as shown in Figure 2.5.

Figure 2.5: Dynamics and velocity associations [18]



Dynamics' Note Velocity

Dynamic	Velocity	Voice
ppp	16	Whispering
pp	33	Almost at a whisper
p	49	Softer than speaking voice
mp	64	Speaking voice
mf	80	Speaking voice
f	96	Louder than speaking
ff	112	Speaking loud
fff	127	Yelling

2.4 Digital Signal Processing

Digital Signal processing (DSP) is an engineering field focused on analyzing and altering digital signals. It takes real-world signals like voice, audio, video and then mathematically manipulates them [19].

Signals need to be processed so that the information they contain can be displayed, analyzed or converted to another type of signal. Analog-to-Digital converters take signals from the real-world and turn them into binary digital format. At this point, the DSP takes over by capturing the digitized information and processes it, later to be fed back for use in the real-world.

2.4.1 Sound

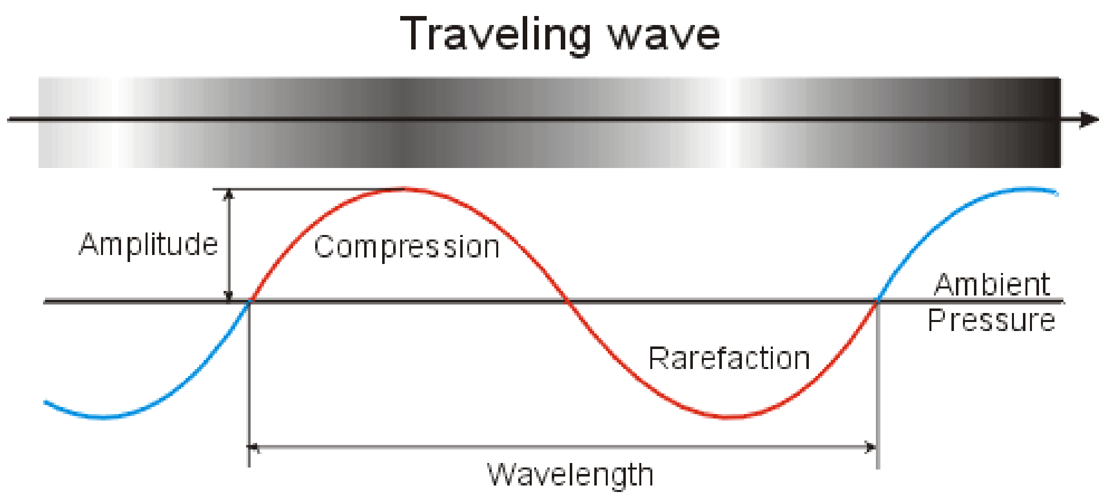
Sound is produced when something vibrates. The vibration causes the medium around it to vibrate as well. Vibrations propagated through air are called traveling longitudinal waves [20], which we can hear. A sound wave is made out of two areas of high and low pressure called compressions and rarefactions (figure 3).

The pattern of the wave repeats after one wavelength. The height of the wave is called **amplitude**. It is what determines how loud the sound will be (the greater the amplitude, the louder the sound).

The wavelength and the speed of the wave determine the pitch (frequency of the sound).

$$c = f \cdot \lambda, \text{ where } c = speed,\ f = frequency,\ \lambda = wavelength \tag{2.5}$$

Figure 2.6: Traveling wave components [21]



2.4.2 Pitch

In music, the pitch tells how low or high a note is. In physics, it is measured in a unit called Hertz (Hz) and it is known as frequency. A note that vibrates at 256Hz will be caused by a sound wave vibrating at 256 times/second.

The speed is influenced by the medium in which the sound wave travels. Under standard conditions of temperature and pressure, sound is speed is 343 meters per second [22].

The equation (5) can be rewritten as:

$$f = \frac{c}{\lambda}, \text{ where } c = speed,\ f = frequency,\ \lambda = wavelength \tag{2.6}$$

2.4.3 Discrete Fourier transformation

The Discrete Fourier Transformation (DFT) is one of the most important operation of DSP. It is any quantity or signal that varies over time, such as the pressure of a sound wave, sampled over a finite time interval (often defined by a window function) [23].

$$X[k] = \frac{1}{N}\sum_{j=0}^{N-1}(x[j] \cdot e^{-j\cdot(\frac{2\pi}{N}))\cdot n\cdot k} \text{ for } k = 0...N\text{-}1 \tag{2.7}$$

The DFT shows what frequencies are present in your signal and in what proportions.

It has a complexity of $O(n^2)$ so in practice, the Fast Fourier Transform (FFT) algorithm is used instead. FFT runs in $O(n \cdot log(n))$.

2.4.4 Fast Fourier transform

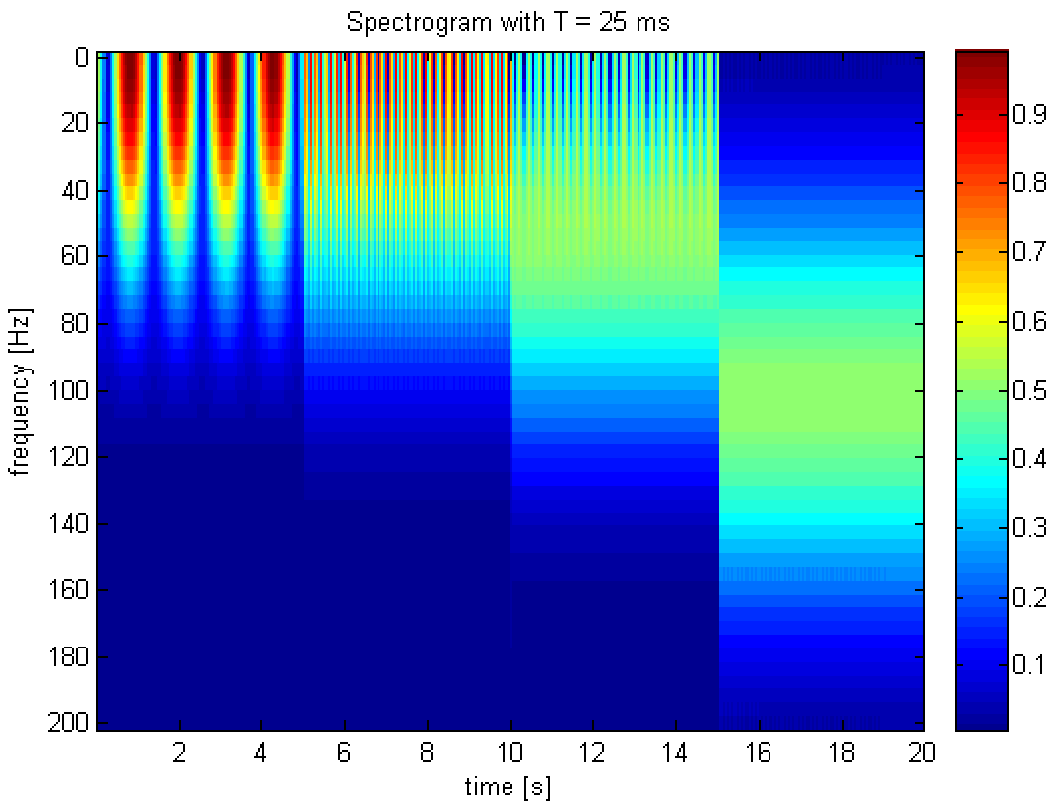
The fast Fourier transform(FFT) computes the DFT of a sequence, or its inverse (IDFT) [24]. It rapidly computes such transformations by factorizing the DFT matrix into a product of sparse factors. As a result, it manages to reduce the complexity of computing the DFT from $O(n^2)$ to $O(n \cdot log(n))$, where n is the data size. The difference in speed can be huge, especially for large data sets where n can reach thousands of millions. Because of this, FFTs are widely used for applications in engineering, science and mathematics.

2.4.5 Short-Term Fourier transform

While DFT is really good by itself, if used on an entire song it would only tell what frequencies exist, but not when they occur. This is where Short-Term

Fourier Transform (STFT) comes in handy. It computes DFT over a full signal but in small segments. Because of this, we can see how frequencies change over time, which makes it a good way to compute spectrograms. (Figure 4)

Figure 2.7: Spectrogram using STFT [25]

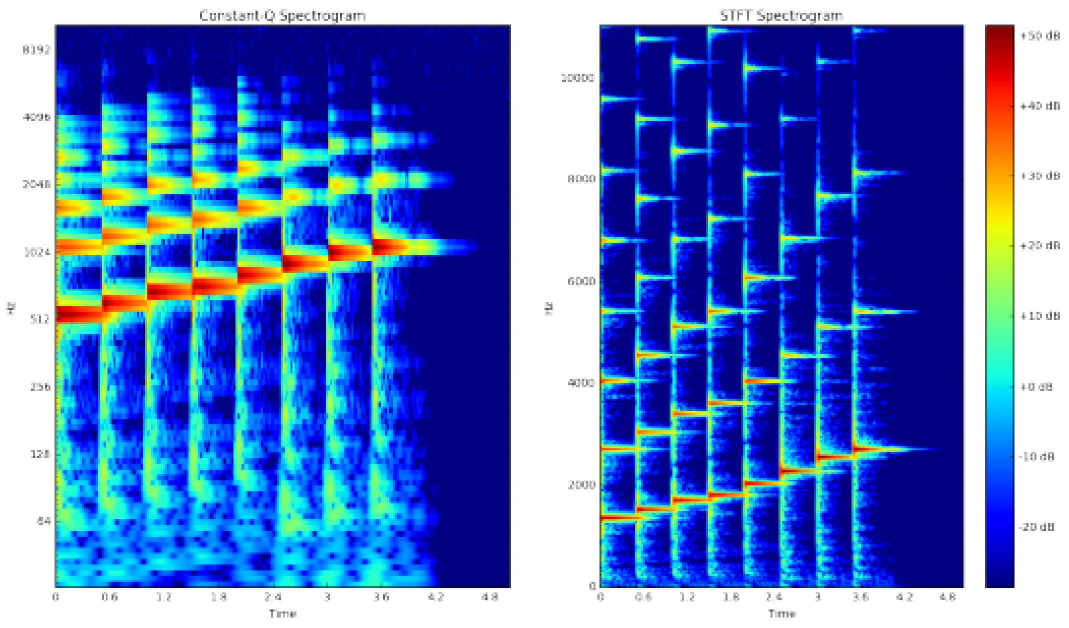


2.4.6 Constant-Q transform

In general, the transform is well suited to musical data, and this can be seen in some of its advantages compared to the fast Fourier transform. As the output of the transform is effectively amplitude/phase against log frequency, fewer frequency bins are required to cover a given range effectively, and this proves useful where frequencies span over several octaves. As the range of human hearing covers approximately ten octaves from 20 Hz to around 20 kHz, this reduction in output data is significant [26].

See Figure 5 for a comparison between the Constant-Q transform and STFT.

Figure 2.8: Constant Q (left) vs STFT (right) spectrogram of C major scale



2.5 Music Transcription

In music, transcription is the process of creating a music sheet from a musical piece or sound. The sheet contains music notation, which consists of different symbols that can be interpreted by musicians, hence it is important for various reasons. Without it, composers such as Mozart and Beethoven could not have passed their masterpieces across generations. In modern days it helps musicians play songs they had never heard before. It is also universal so even if two musicians do not speak the same language, they can read the same notation.

2.5.1 Traditional music transcription

In the beginning, transcription was done by humans. This is called musical dictation in ear training pedagogy [27]. It is a skill by which musicians learn to identify pitches, intervals, melody, chords and other elements of music solely by hearing. It is a really hard skill, requiring serious training and study. Even the best musicians struggle to get 100% accuracy.

There are some tools to help with the process:

- Musical instruments. They help musicians test for certain sounds, trying to mimic what they hear

- Tape recorders
- Current software

Music transcription can be especially difficult and time-consuming when the recordings have many overlapping pitches [28]. The difficulty of this task can be understood in comparison to the ease with which humans can read passages of text and the difficulty of writing down what someone is saying. Another thing that adds to the complexity is that humans often process pitch relatively, rather than absolutely. There are some humans with perfect pitch, which is the ability to recognize pitch in isolation. For those without it, the best approach is the guess-and-check method, which is extremely time-consuming.

2.5.2 Automatic music transcription

The term "Automatic Music Transcription"(AMT) was used for the first time in 1977, by audio researchers James A. Moorer, Martin Piszczalski, and Bernard Galler [29]. With their knowledge about digital engineering, they believed that computers could be programmed to analyze digital recordings of songs such that they could identify things like rhythm, melodies, pitch, bass lines. It is not an easy task. For more than three decades, researchers have been trying to crack it open.

Fundamentally, AMT is about identifying the pitch and duration of played notes, so they can be converted into traditional music notation on a sheet.

It has many advantages over traditional transcription, as it:

- Aids experienced musicians in the process of transcribing pieces, increasing their accuracy
- Makes music transcription available to more people, especially beginners, giving them a chance to share their ideas with others
- Helps people learn new songs. There are a lot of music sheets online that are not free
- Speeds up the process. Manual transcription takes a lot of time

The subtasks of AMT:

- **Pitch Estimation**

  There are two versions of this problem: single pitch and multi-pitch estimation. The real challenge lies in the latter. It is still an unsolved problem [30]. The best algorithms were able to achieve around 70% accuracy, as of 2017 [28]

- **Beat detection**

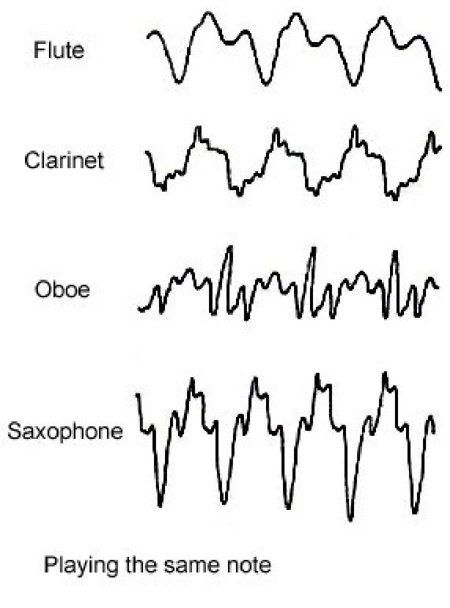
  Beat tracking is the determination of a repeating time interval between perceived pulses in music [29]. Songs are frequently measured for their Beats Per Minute (BPM) in determining the tempo of the music, whether it is fast or slow. Beat can be described as *foot tapping* or *hand clapping* in time with the music. Despite the intuitive nature of the former, which most humans are capable of, developing an algorithm to detect those beats is difficult

- **Instrument detection**

  Given an audio recording, the goal is to identify the musical instrument(s) playing each note. Like pitch estimation, this problem faces the same monophonic and polyphonic problems, with the latter being still unsolved [31]. This is often simplified to classifying a recording in terms of instrument family, not between specific instruments.

  The most common strategy is an evaluation of various features present in a note as its harmonic shape develops over time. These features are different for instrument families (Figure 2.9)

Figure 2.9: Waveform differences between instruments [32]



2.5.3 Semi-automatic music transcription

In between traditional and automatic music transcription comes semi-automatic music transcription, also called user-assisted transcription. It is a system in which the user provides a certain amount of information about the recording which can be used to guide the transcription process [33].

For certain use-cases, semi-automatic transcription is better than the others as it is faster and more accurate than manual transcription and more practical than the fully automatic one. For lightweight usage, where you have a song and you know its genre or other types of information about the recording, the semi-automatic approach will provide a better transcription.

Where semi-automatic transcriptions fall short is when it comes to databases too large to be done by hand, as they would require way too much user input or projects where no data about the recordings are available. With such projects, fully automatic transcription is the only way to go.

Chapter 3

Related Work

Automatic music transcription (AMT) has been attempted since the 1970s and polyphonic music transcription dates to the 1990s [34].

3.1 State of the art in AMT

Substantial progress has been made in the field of AMT. Neural networks, in particular, have met and surpassed the performance of traditional pitch recognition techniques on polyphonic audio.

Poliner et al. [35] proposed a solution that uses 87 Support Vector Machine (SVM) classifiers to perform frame-level classification with the advantage of simplicity, and then a Hidden Markov Model (HMM) post-processing was adopted to smooth the results. On top of it, Nam et al. [36] added a Deep Belief Network(DBN) to learn a higher layer representation of features. Since none of the approaches has reached the accuracy of human experts, most music transcription work is completed by musicians.

The first major AMT approach was proposed by Smaragdis et al. [37]. It uses Non-Negative Matrix Factorization (NMF). This is the main methodology employed in software for automatic transcription, but it has its limitations. For example, it needs to know how many individual notes are desired for the transcription (information that is not always available).

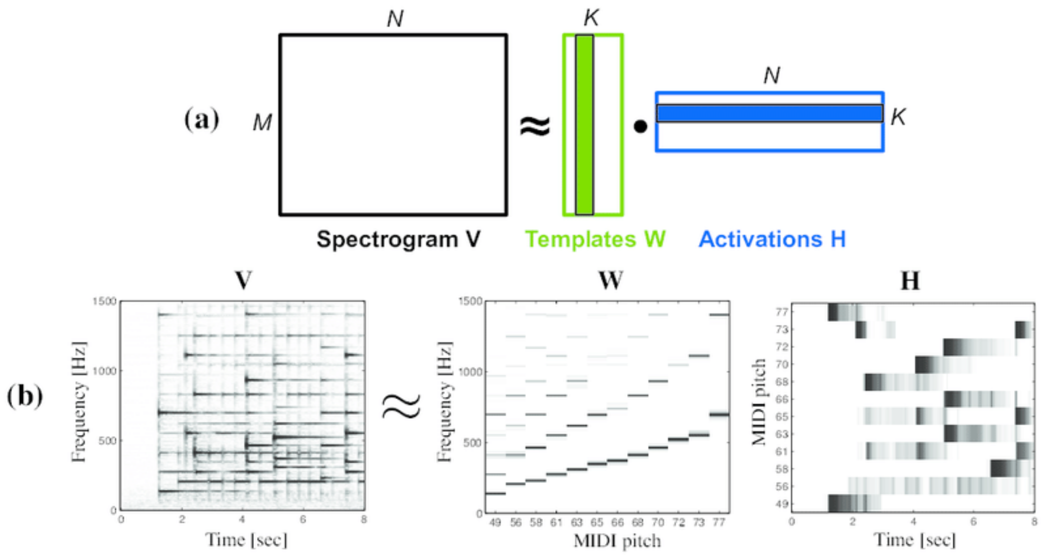
The next work worth mentioning is the one of Emiya et al. [38], not because of their transcription system (as it was out-performed in the same year), but because of the dataset they created that has become the standard in evaluating any multi-pitch estimation system. They created the MIDI-Aligned Piano Sounds (MAPS) data set composed of around 10,000 piano sounds either recorded by using an upright Disklavier piano or generated by several virtual piano software products based on sampled sounds. The dataset consists of audio and corresponding annotations for isolated sounds, chords, and complete pieces of piano music.

Melodyne is a popular plugin used for Music Transcription and Pitch Correction. It costs up to \$700. The Melodic and Polyphonic algorithms offer, in the case of vocals as well as both mono- and polyphonic instruments, full access to the notes of which the sound is composed as well as to their musical parameters. There's no public information about what approach they used.

3.2 NMF approach to AMT

The work of Smaragdis et al. [37] was the first major AMT work. This approach used NMF. It works by factoring a non-negative matrix $X \in R^{\geq 0, MxN}$ into two-factor matrices: $H \in R^{\geq 0, MxR}$ and $W \in R^{\geq 0, RxN}$, where R is chosen, in the case of a piano roll it is 88. H represents when each component is active, known as the Activation Matrix and W shows the frequency spectrum of what should only be a single note, known as Basis Matrix [39].

Figure 3.1: Non-negative matrix factorization [40]



While this is the main technique used in software for AMT, it has its limitations. It needs to know how many individual notes are desired for the transcription, something which is not always available.

3.3 CNN approach to AMT

With the development of deep learning in recent years, researchers were inspired to apply neural networks to achieve AMT. Many models were proposed, including CNNs, Recurrent neural networks (RNN), and Long-short term memory networks (LSTM) [34] [41]. Five models were compared and CNN was reported to have the best performance [1].

Sigtia et al. [42] built the first AMT system us ing CNN, outperforming the state of the art approaches using NMF. Convolutional Neural Networks are a discriminative approach to AMT, which has been found to be a viable alternative to spectrogram factorization techniques. Discriminative approaches aim to directly classify features extracted from frames of audio to the output pitches. This approach uses complex classifiers that are trained using large amounts of training data to capture the variability in the inputs, instead of constructing an instrument-specific model.

Table 3.1: Optimal parameters and architecture for ConvNet in Sigtia et al. [1]

Parameter Name	Value
Window Size (Spectrogram Frames)	7
Number of ConvNet Layers (Conv+tanh+Pooling)	2
Number of Fully Connected Layers	2
Window Shapes(1,2)	(5,25),(3,5)
Pooling Size	(1,3)
Fully Connected Widths (1,2)	1000,200

Sigtia et al.. [42] explored various models for pitch detection. In addition to CNNs, they tried Deep Neural Networks and Recurrent Neural Networks. The results have shown that their CNN based model outperformed the others for this task. In their paper, they propose a Music Language Model (MLM) that's based on RNNs in order to handle the polyphonic musical data [43].

There are some products on the market (i.e. AnthemScore), that use CNNs for AMT. They approach note detection as an image recognition problem by creating spectrograms of the audio. They show how the spectrum or frequency content changes over time. The method used for creating the spectrograms is the constant Q transform instead of the more common Short Time Fourier Transform (STFT) method.

Chapter 4

Theia7

4.1 Problem statement

Automatic music transcription consists of analyzing digital recordings of songs and identify things like rhythm, melodies, pitch, bass lines. The main subtasks of AMT are:

- **Pitch estimation**
- **Beat detection**
- **Instrument detection**

Because of the polyphonic approach of music, AMT is still unresolved today. The state of the art approaches struggle to get an accuracy greater than 70%. It's a problem that requires a lot of digital signal processing knowledge in order to optimize the data and split the subtask even further in onset and offset detection, genre detection and velocity detection. In order to have a full prediction, you need to cover all of these tasks.

The proposed solution attempts the **Pitch estimation** part of AMT, using a heuristic approach to find the beat and creating spectrograms of the audio in order to get the instrument out of the equation.

Fundamentally, AMT is about identifying the pitch and duration of played notes, so they can be converted in traditional music notation on a sheet

4.2 Data set and input representation

The data set used was the MAPS database [2]. It is a piano database for multi-pitch estimation and automatic transcription of music. It contains MIDI-annotated piano recordings, composed of isolated notes, random-pitch chords, usual musical chords and pieces of music. It provides a diverse range of sounds from various recording conditions.

The recordings are CD quality (16-bit, $44 - kHz$ sampled stereo audio) and the related aligned MIDI files contain the ground truth. The overall size of the database is 40GB.

Working with sound in neural networks is different from dealing with images. The input contains audio files so before feeding it to the network we need to turn it into a visual representation. The most common way to represent a sound is the audio representation in the time domain. (Figure 4.1)

Figure 4.1: Example of audio representation in the time domain [44]

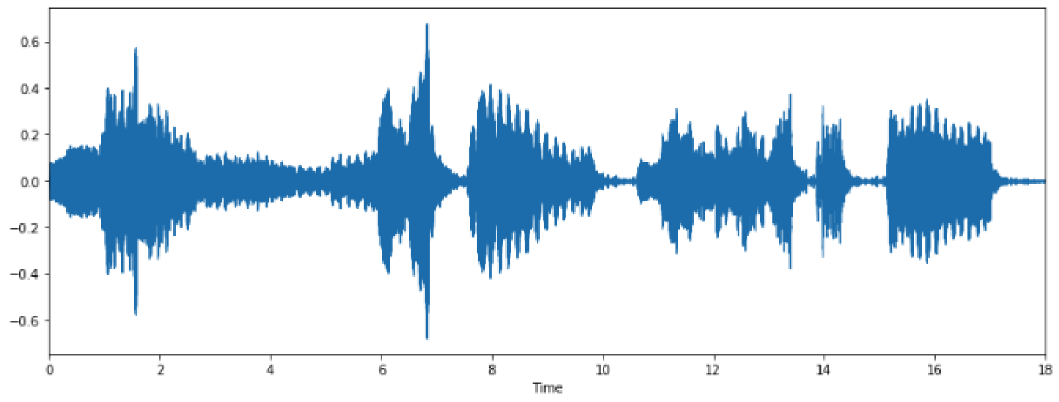


Figure 4.1 shows the evolution in time of the song and you can see the oscillation of the signal. The Y axis represents the amplitude while the X axis is the time. In this representation, it's not possible to distinguish the notes that are playing. Because of this, we need a better representation. This is where the Fourier Transformations come in handy. Using the transformation on the data we get a representation over frequency instead of time. This is also known as a spectrum. The spectrum reveals relevant information that's crucial to analyze the audio. (Figure 2.8)

The input MIDI file will be split into $\frac{1}{16} \cdot second$ window frames. For example, a note lasting 1 second will have 16 consecutive windows created. Each window is then turned into a .wav file, using fluidsynth [45], which in turn will be transformed using the Constant-Q transform. One window transformation takes 0.3 seconds making the process slow and punishing if a bug is found in the preprocessing.

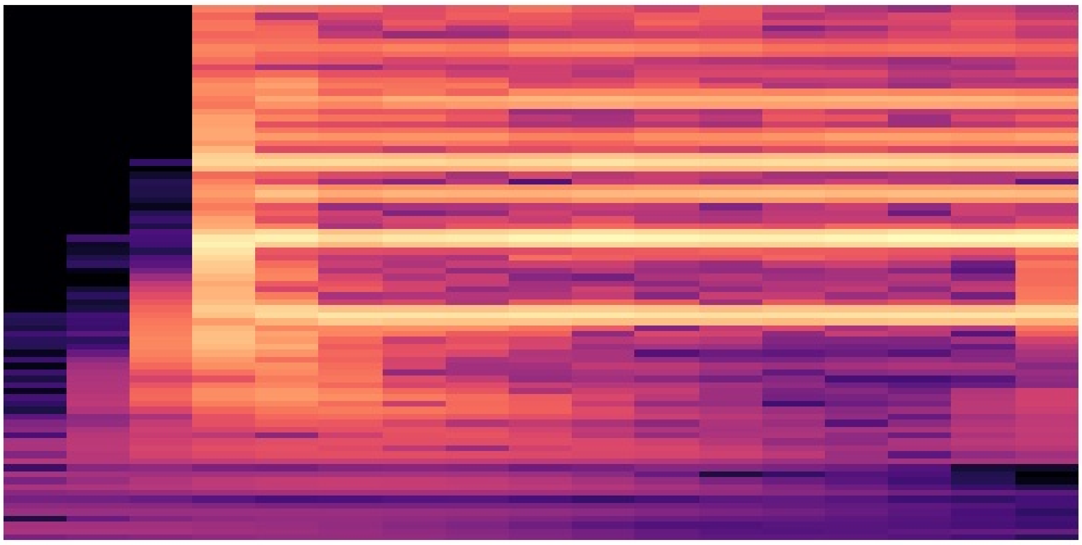
To compute the Constant-Q transform, the library *Librosa* [46] will be used, in particular, the method called *librosa.cqt*. The parameters that we will use are:

- **y**: Audio signal

- **sr**: Sampling rate
- **fmin**: Minimum frequency
- **n_bins**: Number of frequency bins
- **bins_per_octave**: Number of bins per octave
- **hop_length**: Number of samples between successive CQT columns

The result of the Librosa function will be plotted, resulting in a logarithmic scale spectrogram of size 145×49 (Figure 4.2). This way from a song of length x, $x \cdot 16$ spectrograms will be extracted. These are the input of the network.

Figure 4.2: Example of a constant-q spectrogram



For the cross validation, a train-validation-test split approach was used. The learning set was split 80% for training, 20% for testing and 10% of the training set is used for validation. A visual representation can be seen in Figure 4.3

Figure 4.3: Visual representation of the split

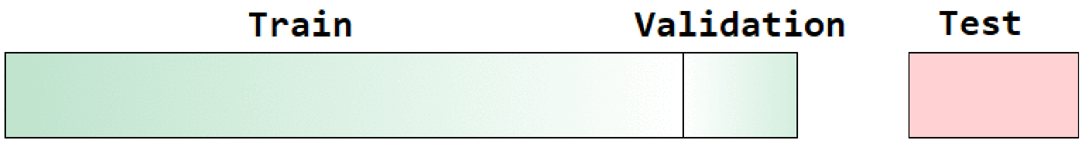


Table 4.1: Example of one hot representation of a MIDI file

Note	Window 1	Window 2	...	Window n
1	0	0	...	0
2	1	0	...	0
3	0	0	...	1
...	...	...	...	...
128	0	1	...	0

4.3 Labeling

A supervised machine learning model needs the data to be labeled. As it was explained in the subsection above, the input song is going to be represented in the frequency domain. The labeling will be provided using the MIDI files associated with the input *.wav* files. They contain information about the notes playing at a certain time. The labels will be arrays showing the played notes represented with *one-hot encoding.*

One-hot encoding is a widely used technique in neural networks. It consists of creating an array of boolean values [47]. In this case, every column symbolizes a possible note that can be played at a certain moment. Because of this, there are as many columns as possible notes (128). This will be done per window, so the goal is to see which notes are played in a certain window (Table 4.1). A value of 1 in a cell means that the specific musical note has been played during that window, while a value of 0 is the opposite.

4.4 First CNN architecture

The first neural network has been created with the aim of obtaining some initial results and it has served to learn how a model developed in Keras [48] behaves.

An architecture similar to the Keras MNIST is evaluated [49]. It has been created to predict handwriting digits, the input and output data are completely different, but this is a standard architecture for image recognition with small patches. No optimization was done on this one as its aim was to perform a first approach that deals with the input data.

The architecture is shown in Table 4.2. It performed poorly but it helped in the preprocessing progress as it learned really fast on a GPU. Making changes to the input sizes and different types of transformations and retesting was really quick.

Table 4.2: Initial CNN architecture

Layer (type)	Output Shape
Conv2D	(None, 72, 24, 32)
Conv2D	(None, 70, 20, 64)
MaxPooling2	(None, 35, 10, 64)
Dropout(0.25)	(None, 64)
Flatten	(None, 2048)
Dense	(None, 128)
Dense	(None, 128)

4.5 Proposed CNN architecture

The proposed architecture is similar to AlexNet [50] (Figure 4.5). This architecture won the Image Classification Challenge in 2012[51]. It was used on high-resolution images into 1000 different classes. This was too big for our input data of $145 \times 49 \times 3$ size images. The kernel and stride sizes were adjusted to fit our data. Some layers were dropped and some dropout layers were added in order to reduce overfitting. The dropout layers have a 0.5 chance to deactivate a neuron. This forces the layer to learn the same concept with different neurons, improving generalization.

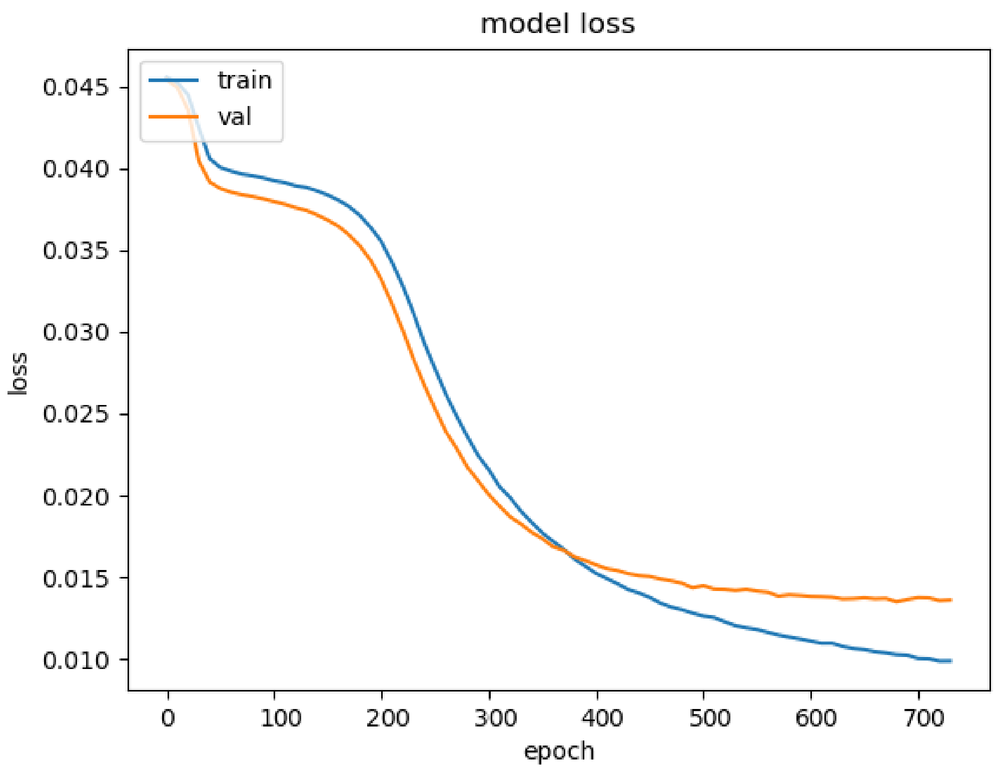
Three activation functions were tested in the hidden layers for the monophonic scenario as this was the first milestone. Without good accuracy here there's no point going further:

- **Sigmoid**: 30% accuracy
- **ReLu**: 83% accuracy
- **Tanh**: 99.9% accuracy

The model performed better using Tanh as the activation function inside the hidden layers. As for the output layer, Softmax was used as this is a multi class classification problem.

The accuracy of the model was tested using custom made song with predetermined note overlap percentage. The results can be seen in Table 4.3. As the amount of overlapping notes increases the accuracy of the model decreases. This happens because the system can't predict with high confidence all the notes from a window when their number increases. It works almost perfectly for monophonic songs, reaching an accuracy of 99.9%. The sweet spot of the system is a combination of one, two and three overlapping notes with no more than 50% and 10%

Figure 4.4: Loss for the final architecture



overlapping time for two and three notes respectively. This combination provides an overall accuracy of about 70%.

Table 4.3: Initial CNN architecture

Max number of overlapping notes	Overlapping Time	Accuracy
1	100%	99.9%
2	10%	95.3%
2	50%	78.1%
2	100%	65.0%
3	10%	90.3%
3	50%	67.2%
3	100%	43%

Figure 4.5: Original AlexNet architecture [50]

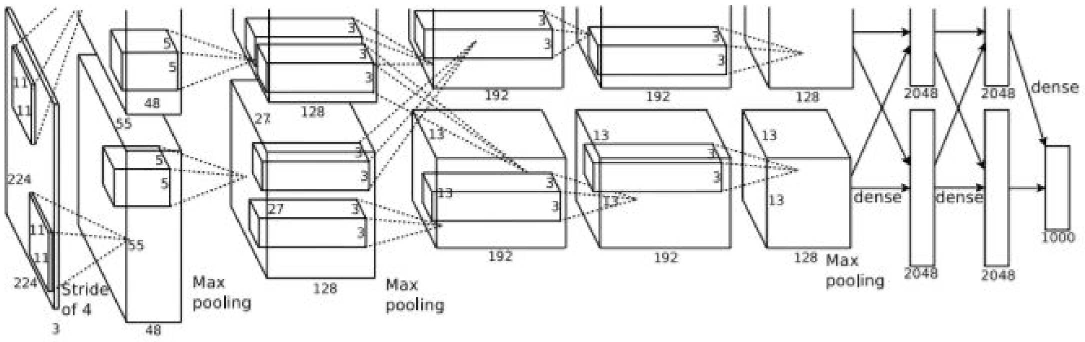
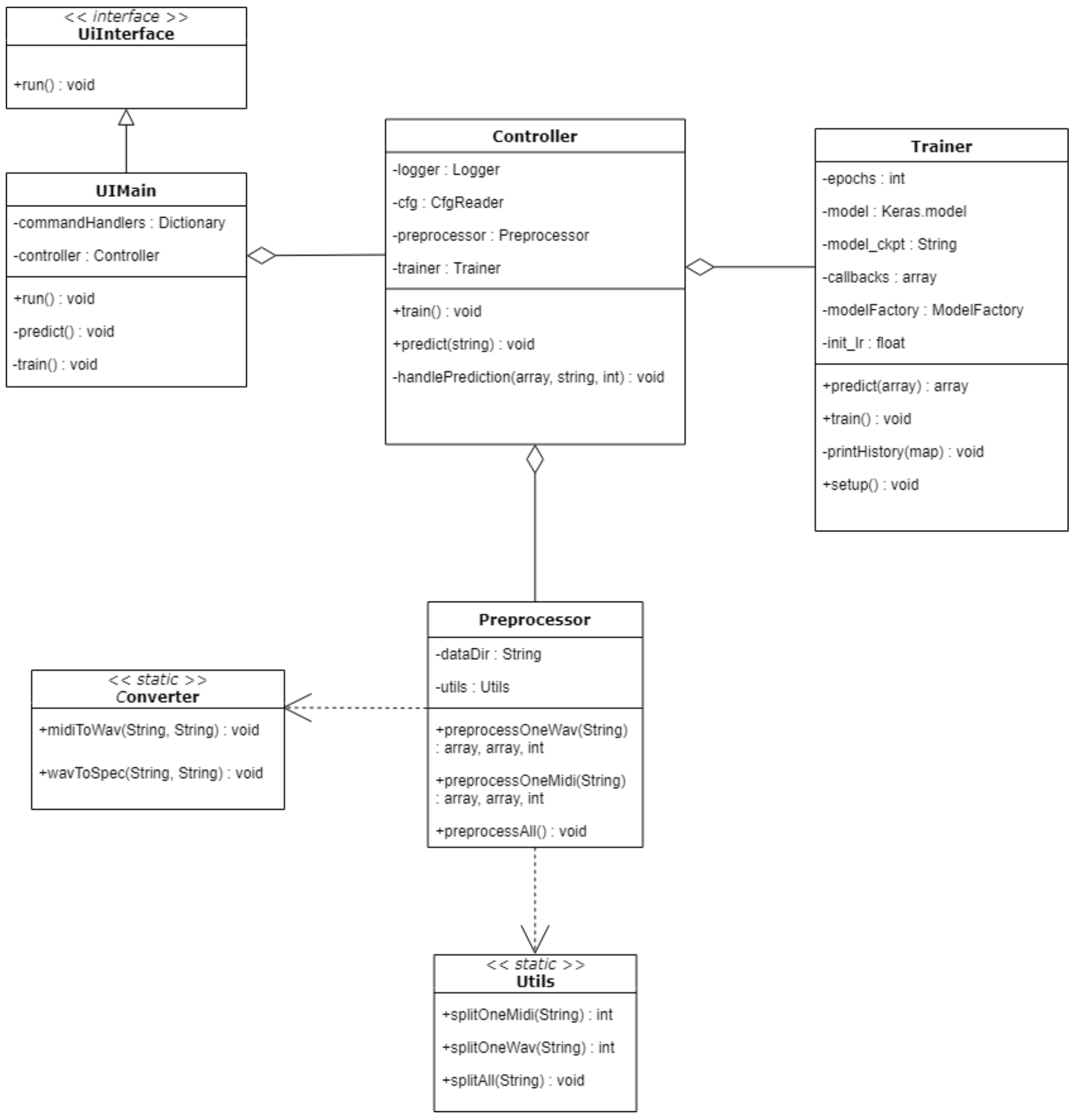


Table 4.4: Proposed CNN architecture

Layer (type)	Output Shape	Param #
Conv2D	(None, 72, 24, 32)	896
MaxPooling2	(None, 36, 12, 32)	0
Conv2D	(None, 17, 5, 64)	18496
Conv2D	(None, 16, 4, 128)	32896
MaxPooling2	(None, 8, 2, 128)	0
Flatten	(None, 2048)	0
Dense	(None, 4096)	8392704
Dropout	(None, 4096)	0
Dense	(None, 4096)	16781312
Dropout	(None, 4096)	0
Dense	(None, 128)	524416

4.6 Application architecture

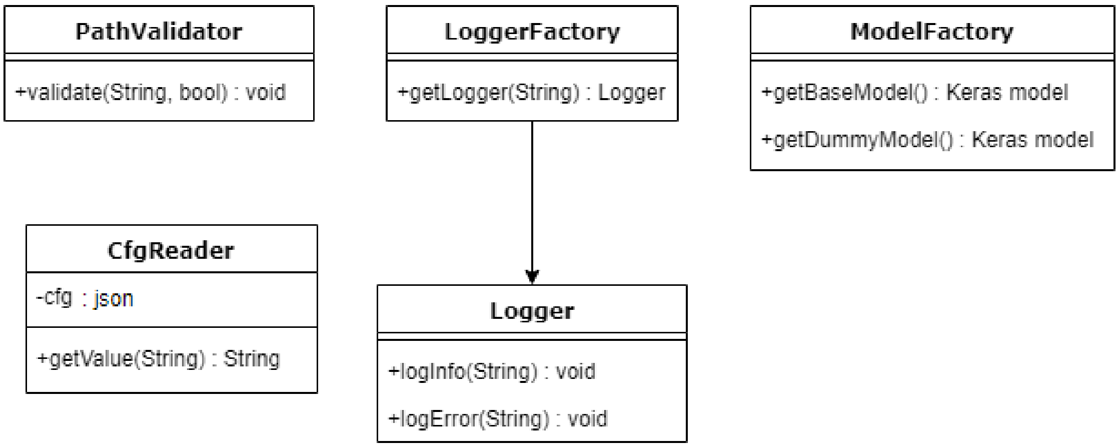
Figure 4.6: Application class diagram



Theia7 has a layered application without the data layer. For the model and loggers creation, the abstract factory design pattern was used in order to facilitate easy creation and scaling for these features. The main business logic happens in the controller with the trainer and preprocessor at its disposal.

The configuration of the application is contained inside a json configuration file. A wrapper over it was created to facilitate reading fields from it inside the classes (Figure 4.7).

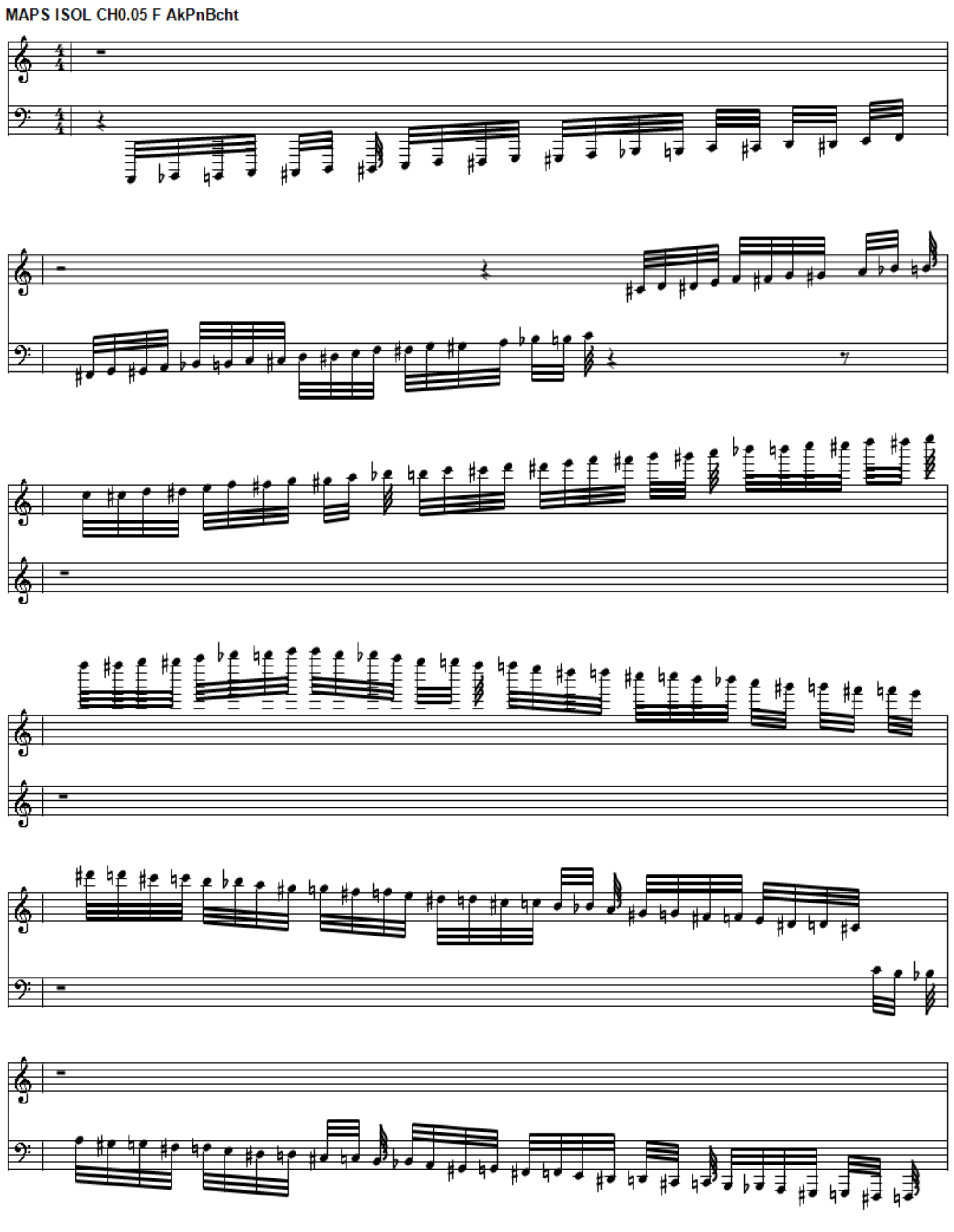
Figure 4.7: Static classes diagram


PathValidator
+validate(String, bool) : void
LoggerFactory
+getLogger(String) : Logger
ModelFactory
+getBaseModel() : Keras model
+getDummyModel() : Keras model
CfgReader
-cfg : json
+getValue(String) : String
Logger
+logInfo(String) : void
+logError(String) : void

4.7 Output

The output of the prediction will be a one-hot encoding of the song as described in Table 4.1. This will be transformed into a MIDI file using the *pretty-MIDI* library [52]. Knowing that each window is a $\frac{1}{16}$ of a second we can find an approximation of the original note length by concatenating all consecutive windows predicted for it. The MIDI is then converted to wav using fluidsynth [45] to have a sound comparison and the spreadsheet is created using the Sheet software [53]. An output example can be found in Figure 4.8.

Figure 4.8: Sheet output example



4.8 Technologies used

List of used technologies:

- Python [54]
- Anaconda [55]
- Librosa [46]
- Keras [48]
- Tensorflow [56]
- Pretty-midi [52]
- Sheet [53]
- Numpy [57]
- Pydub [58]
- Mido [59]
- Cuda [60]

4.9 Testing

The application was tested using unit-testing for various components:

- Path validators were given non-existing, existing paths, directories when testing for normal files and vice versa.
- Preprocessing utils were tested to see if the songs are split into the correct number of windows
- One-hot encoding and decoding utils

Chapter 5

Conclusion

The recognition of musical notes played in a song is a complex problem to deal with because of the polyphonic aspect of music and it's still an unsolved problem. The neural network has to be able to distinguish the notes from different instruments. Because of this, a Constant-Q transform was applied. This removes instruments from the equation, normalizing the data and obtains valuable information from the song. The MAPS dataset [2] provides raw labels for each song but they don't fit the expected output of the network. Because of this, each MIDI provided needs to be analyzed and one-shot encodings of size 128 are extracted for each $\frac{1}{6}$ of a second window of the song to determine what notes are played during this time frame. The windows are then converted into wav format and then into spectrograms using the Librosa [46] library. The spectrograms are then fed to the network's learning process. The learning set is split 80% for training, 20% for testing and 10% of the training set is used for validation.

The output is a one-shot encoding of each window. These are then glued together and a MIDI file is created. From this MIDI file, a wav file is exported for comparison and the music sheet is generated using the Sheet freeware [53].

The proposed model reached a 99.9% accuracy for the monophonic part of the problem. With a combination of one, two and three overlapping notes with no more than 50% and 10% overlapping time for two and three notes respectively the system achieved an accuracy of 70% which is the state of the art method from Google Magenta. That accuracy is achieved for songs way more complicated, but their system is far more complex than the one proposed in this thesis. Going into more complicated songs with the proposed model drops the accuracy significantly as it can't give a strong enough prediction about any playing notes at a certain time if there's too much noise and overlapping sounds.

The proposed solution for automatic music transcription consists of creative and heuristic methods to overcome the lack of digital processing knowledge necessary to solve this problem. There are ways to improve accuracy further:

- A long short-term memory network could be added to aid with the classification and detect the length of the notes
- Genre detection could also aid with the classification as it will tell if a note makes sense in a progression
- Further improvements in the preprocessing. By changing the spectrogram from a STFT to a Constant-Q the accuracy improved significantly. There are more ways the make the input clearer by reducing the noise and improve the window splitting. For this task more knowledge of digital signal processing is needed.
- Train on more data. Converting one window takes 0.3 seconds. A song is split into $16 * n$ windows, where n is the song length in seconds. More than two billion MIDI windows were created but only 160 thousand were actually converted into spectrograms due to time constraints.

The final goal for Theia7 is to be integrated into a mobile music teaching and music sheet database application, where people could share their ideas and music sheets with everyone. Theia7 would aid people with no musical theory knowledge by creating the musical sheets for them. This would disrupt the market as many people ask for money in exchange for their music sheets, a common practice for YouTube cover channels.

Bibliography

[1] S. Sigtia, E. Benetos, and S. Dixon, "An end-to-end neural network for polyphonic piano music transcription," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 24, no. 5, pp. 927–939, 2016.

[2] V. Emiya, R. Badeau, and B. David, "Multipitch estimation of piano sounds using a new probabilistic spectral smoothness principle," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 18, no. 6, pp. 1643–1654, 2009.

[3] A. W.A and E. S.M, "Machine learning methods for spam e-mail classification," *International Journal of Computer Science and Information Technology*, vol. 3, 02 2011.

[4] R. Szeliski, *Computer Vision: Algorithms and Applications.* Berlin, Heidelberg: Springer-Verlag, 1st ed., 2010.

[5] "Microsoft azure computer vision." https://azure.microsoft.com/en-in/services/cognitive-services/computer-vision/.

[6] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[7] "Neuron." https://en.wikipedia.org/wiki/Neuron.

[8] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[9] B. C. Bangal, "Automatic generation control of interconnected power systems using artificial neural network techniques," 2009.

[10] *Everything you need to know about Neural Networks.* 2018. https://hackernoon.com/everything-you-need-to-know-about-neural-networks-8988c3ee4491.

[11] “Activation functions.”
http://prog3.com/sbdm/blog/cyh24/article/details/50593400.

[12] “Vitaflux.” vitalflux.com.

[13] “Convolutional neural networks.”
https://en.wikipedia.org/wiki/Convolutional-neural-network.

[14] “Cnn representation.” https://en.wikipedia.org/wiki/File:Typical_cnn.png.

[15] “Convolutional neural networks.”
https://cs231n.github.io/convolutional-networks/.

[16] J. Heckroth, “A tutorial on midi and wavetable music synthesis,” *Application Note, CRYSTAL a division of CIRRUS LOGIC*, 1998.

[17] R. Guérin, *MIDI power!: The comprehensive guide.* Course Technology PTR, 2009.

[18] “Dynamics in music.” https://en.wikipedia.org/wiki/Dynamics(music).

[19] “Digital signal processing.” https://www.analog.com/en/design-center/landing-pages/001/beginners-guide-to-dsp.html.

[20] “Physics of sound.”
https://method-behind-the-music.com/mechanics/physics/.

[21] “Traveling wave.”
https://method-behind-the-music.com/mechanics/physics/.

[22] “Speed of sound.” https://en.wikipedia.org/wiki/Speed-of-sound.

[23] M. Sahidullah and G. Saha, “A novel windowing technique for efficient computation of mfcc for speaker recognition,” *IEEE signal processing letters*, vol. 20, no. 2, pp. 149–152, 2012.

[24] “Fast fourier transform.”
https://en.wikipedia.org/wiki/Fast-Fourier-transform.

[25] “Short time fourier transform example.”
https://en.wikipedia.org/wikishort-time_fourier_transform.

[26] “Constant-q transform.”
https://en.wikipedia.org/wiki/Constant-Q-transform.

[27] Z. Duan and E. Benetos, “Automatic music transcription,” in *ISMIR conference*, 2015.

[28] J. S. Downie, “Music information retrieval evaluation exchange (mirex),” *http://www. music-ir. org/mirex/2009*, 2009.

[29] “Transcription.” https://en.wikipedia.org/wiki/Transcription(music).

[30] E. Benetos, S. Dixon, D. Giannoulis, H. Kirchhoff, and A. Klapuri, “Automatic music transcription: Breaking the glass ceiling,” 2012.

[31] A. Livshin and X. Rodet, “Musical instrument identification in continuous recordings,” in *Digital Audio Effects 2004*, 2004.

[32] “Sound waves of different musical instruments.” http://eyuzdaosivadare.changeip.com/Sound-waves-and-musical-instruments.html.

[33] H. Kirchhoff, S. Dixon, and A. Klapuri, “Multi-template shift-variant non-negative matrix deconvolution for semi-automatic music transcription.,” in *ISMIR*, 2012.

[34] D. G. Morin, “Deep neural networks for piano music transcription,” 2017.

[35] G. E. Poliner and D. P. Ellis, “A discriminative model for polyphonic piano transcription,” *EURASIP Journal on Advances in Signal Processing*, vol. 2007, no. 1, p. 048317, 2006.

[36] J. Nam, J. Ngiam, H. Lee, and M. Slaney, “A classification-based polyphonic piano transcription approach using learned feature representations.,” in *Ismir*, pp. 175–180, 2011.

[37] P. Smaragdis and J. C. Brown, “Non-negative matrix factorization for polyphonic music transcription,” in *2003 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (IEEE Cat. No. 03TH8684)*, pp. 177–180, IEEE, 2003.

[38] V. Emiya, R. Badeau, and B. David, “Multipitch estimation of piano sounds using a new probabilistic spectral smoothness principle,” *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 18, no. 6, pp. 1643–1654, 2009.

[39] “Non-negative matrix factorization.” https://en.wikipedia.org/wiki/Non-negative-matrix-factorization.

[40] S. Ewert and M. Müller, "Score-informed source separation for music signals," in *Dagstuhl Follow-Ups*, vol. 3, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2012.

[41] B. L. Sturm, J. F. Santos, O. Ben-Tal, and I. Korshunova, "Music transcription modelling and composition using deep learning," *arXiv preprint arXiv:1604.08723*, 2016.

[42] J. Sleep, "Automatic music transcription with convolutional neural networks using intuitive filter shapes," 2017.

[43] E. Benetos and S. Dixon, "A shift-invariant latent variable model for automatic music transcription," *Computer Music Journal*, vol. 36, no. 4, pp. 81–94, 2012.

[44] "Music genre classification." https://towardsdatascience.com/music-genre-classification-with-python-c714d032f0d8.

[45] "Fluidsynth." Software available from http://www.fluidsynth.org/.

[46] "Librosa library." Software available from https://librosa.github.io/librosa/.

[47] "One-hot encoding." https://en.wikipedia.org/wiki/One-hot.

[48] "Keras." Software available from https://keras.io/.

[49] "Keras mnist dataset." https://keras.io/examples/mnist-cnn/.

[50] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[51] "Imagenet 2012 challenge." http://www.image-net.org/challenges/LSVRC/2012/.

[52] "Pretty midi." Software available from https://github.com/craffel/pretty-midi.

[53] "Midi to sheet converter." Software available from https://github.com/BYVoid/MidiToSheetMusic.

[54] "Python software foundation. python language reference, version 3.6." Available at http://www.python.org.

[55] "Anaconda." Available at https://www.anaconda.com/.

[56] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[57] T. Oliphant, *Guide to NumPy.* 01 2006.

[58] "Pydub." Software available from http://pydub.com/.

[59] "Mido." Software available from https://mido.readthedocs.io/en/latest/.

[60] "Cuda." Software available from https://developer.nvidia.com/about-cuda.