UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

epl

Deep Learning in Automatic Piano Transcription

Dissertation presented by
Mohamed KARIOUN, Simon TIHON

for obtaining the Master's degree in
Computer Science, Computer Science and Engineering

Supervisor(s)
Christophe DE VLEESCHOUWER, Joachim GANSEMAN

Reader(s)
Laurent JACQUES

Academic year 2017-2018

Abstract

Automatic Piano Transcription (APT) is a branch of Music Information Retrieval (MIR) that focuses on transcribing high-quality piano audio recordings into sheet music. In this thesis, we propose and analyse new methods to improve the state of the art deep learning approaches, basing our experiments on the Onsets and Frames algorithm defined in [1].

First, we proposed the so-called *harmonic layer*, a convolutional layer specifically designed to take into account the harmonics of the notes. However, traditional networks already take these harmonics into account implicitly thanks to the fully connected layers. Even if the results of our experiments are not initially promising, the layer could probably find a better use in a more appropriate architecture.

Second, we balanced the dataset's labels. Given the nature of the datasets (piano recordings of traditional and classical pieces), strong imbalances are observed in the total activation time of the different notes. The performances of Onsets and Frames are slightly improved when balancing the pitch distribution during training through repeated sequences.

Finally, a simplified model of Onsets and Frames is proposed. At the expense of slightly worse performances, the proposed algorithm runs 6 times faster. This model can be used for fast prototyping or as a basis for more complex models.

The different models are compared using ROC curves and F1 scores, but also pitch-wise F1 scores, which reveal some interesting differences in the learning of different pitches.

Acknowledgement

First, we would like to thank our supervisor Christophe de Vleeschouwer who accepted to supervise this thesis on a subject he did not master. His fast answers and great advice helped us all along to take the right decisions in critical moments. We also thank our second supervisor Joachim Ganseman who greatly helped us thanks to its knowledge of the field, including its community. The start would have been harder without the papers and other sources he sent us to begin with and the motivation he helped us to keep.

We also thank Denis Tihon for its careful proofreading and precise comments, despite its tight schedule, and our families and Fiona Bonta Luises for their cheerful support throughout this thesis; the Computing Science Engineering Department for the computer resources they gave us, and its patient studadmins who helped us solve some SSH problems.

Finally, we thank the Google Brain team for the open source algorithm Onsets and Frames and V. Emiya for its great dataset MAPS, both of which are the main bases of this thesis.

Contents

Acronyms

AMT	Automatic Music Transcription
ANN	Artificial Neural Network
APT	Automatic Piano Transcription
BiLSTM	Bidirectional Long Short-Term Memory
CQT	Constant Q-Transform
DFT	Discrete Fourier Transform
DNN	Deep Neural Network
EM	Expectation-Maximisation
FFT	Fast Fourier Transform
FN	False Negative
FP	False Positive
GUI	Graphical User Interface
HMM	Hidden Markov Model
i.i.d.	independent and identically distributed
LSTM	Long Short-Term Memory
MAPS	MIDI Aligned Piano Sounds
MIDI	Musical Instrument Digital Interface
MIR	Music Information Retrieval
MLM	Music Language Model
NMF	Non-negative Matrix Factorisation
PLCA	Probabilistic Latent Component Analysis
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
SI-PLCA	Shift-Invariant Probabilistic Latent Component Analysis
STFT	Short-Time Fourier Transform
TN	True Negative
TP	True Positive

Introduction

Music is deeply rooted in our daily lives. We hear it as soon as we turn on the radio, go shopping, watch a movie or even sing in the shower. Many people also play an instrument, trying to reproduce the music they like. However, doing so is a hard task if all they have is what they hear. This is why music sheet has been invented, making it easier for musicians to read and reproduce the corresponding music. **Chapter 1** is dedicated to the musical knowledge used in the thesis.

This thesis will explore the field of Automatic Piano Transcription (APT), a branch of Music Information Retrieval (MIR) which focuses on transcribing high-quality piano audio recordings into sheet music. The first step of this process generally consists in transforming the audio recording into a piano roll representation, which is less abstract than music sheet. However, some state of the art approaches such as [2] produce directly sheet music, thanks to some simplifying hypotheses. The APT task is considered as being solved for monophonic music. However, for polyphonic music, human experts still outperform computers. This task can be generalised to Automatic Music Transcription (AMT), which does not focus on piano but aims at transcribing any combination of instruments.

Many approaches to solve APT or AMT have been explored in the literature. Non-negative Matrix Factorisation [3] exploits the spectral properties of the musical notes to factorise the input spectral matrix into an atom activity matrix and an atom basis matrix. Probabilistic latent component analysis [4] can be seen as a probabilistic extension of NMF that incorporates priors and has more control over the resulting decomposition. Some machine learning approaches have also been explored, such as Anthem Score [2] and Onsets and Frames [1]. These last approaches are the most promising ones in the literature, even if human experts outperform these significantly. **Chapter 2** goes through widely spread techniques such as Hidden Markov Models and Deep Learning. **Chapter 3** explains how these techniques are used in the state of the art algorithms.

After this literature review, **chapter 4** presents a thorough analysis of the MAPS dataset [5]. We then explore different approaches to improve the Onsets and Frames algorithm. The first one is the use of a new convolutional layer, the *harmonic layer*, which is designed to take into account a very basic principle in music: the harmonics of a musical note. The layer description and its performances are detailed in **section 5.3**. The second one is a simplification of the original model, running faster at the expense of slightly degraded results, described in **section 5.4**. The third one is an analysis of the effect of balancing the pitch distribution in the training dataset, in **chapter 6**. Finally, a pitch-wise analysis of the performances is proposed for deeper assessments of the results and more relevant comparisons of the models.

Part I

Background

Chapter 1

Musical background

Music is a huge part of our cultural heritage. The first known instrument seems to be a Neanderthal flute with four holes sculpted in a bone 45000 years ago [6]. Music has evolved in many different civilisations, along with dances, primarily for rituals, then for pleasure. Many instruments have been invented, many music styles have been explored, and conventions have been agreed for its transcription. This section will introduce the music background and terminology used in this thesis.

1.1 General characteristics of a note

A sound is a vibration of the air (or any material). It can be generated by a vibrating string, some wind in a tube, vibrating membranes, and so on. The main atomic structure of music is the note, which can be described using three elements [7]:

- **The pitch**. It is the **fundamental frequency** of a musical vibration, the one that is the most perceived. A higher pitch means a higher frequency.

- **The velocity** (sometimes called dynamic). It is the strength at which the note is played. A high velocity means more decibels.

- **The timbre**. When an instrument plays a sound, it produces a vibration with a main frequency, its **fundamental frequency**. However, it also produces many other frequencies, in particular the frequencies that are multiples of the main frequency (also called **harmonics**). These compose its timbre. Each instrument has its own timbre, which varies even between two instruments of the same type, for example two pianos.

In other words, roughly speaking, the timbre is related to the repeated form of the sound wave while the pitch is related to its period and the velocity to its amplitude, as shown in figure 1.1.

Each note also has an **onset**, the time at which it starts, as well as an **offset**, the time at which it stops. For some instruments, these boundaries are not clear-cut. In the particular case of a piano, the evolution of a note can be further decomposed. An **attack stage** comes just after the onset, during which the amplitude of the sound is growing, followed by a **decay stage** during which the note loses intensity over time.

When it comes to playing many notes at once, some sets of notes, called **chords**, sound better than others. These generally consist of sets of notes sharing many harmonics (**overlapping harmonics**). The most famous chords are the sets of notes with pitches following a 4:5:6 proportion (major triads, e.g. 440 Hz, 550 Hz and 660 Hz), a 1:2 proportion (octaves), and any proportion described as a ratio of small integers.

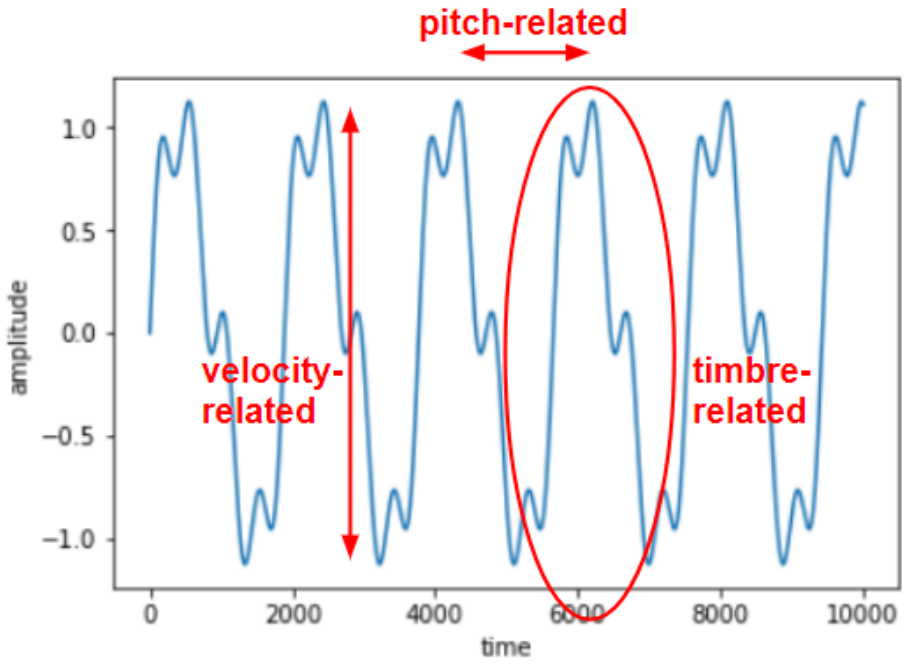

Figure 1.1: Toy example of a sound wave.

1.2 Instruments

1.2.1 Instrument types

Instruments have traditionally been divided in five big families [8]:

- The strings, like violins, which produce music with the vibration of strings. These can be induced by rubbing, striking or plucking these. The pitch produced by a given string is mainly adjusted by changing the length of its vibrating part.

- The woodwinds, like clarinets and saxophones. They produce music through the vibration of an air column by the blow of the musician. The pitch is generally adjusted by blocking different holes (with the fingers or a system of keys) of the instrument.

- The brass instruments, like trumpets, which produce music by the vibration of the air in the instrument caused by the vibration of the lips of the musician. As for woodwinds, the pitch can generally be adjusted by blocking different holes. Some instruments such as the trombone use a varying length to adjust the pitch. Others can do different pitches simply by the way the musician blows in these.

- The keyboard instruments, like pianos, which are played using a keyboard. The sound can be produced by strings (struck or plucked), by some wind in tubes (organ), and so on. This family can use almost any system introduced in the other families.

- The percussion instruments, which can be pitched (e.g. xylophones) or not (e.g. drums). It groups all other instruments, from maracas to vibraphones.

The voice and other human sounds can also be considered as being instruments. As a homonym, a **voice** can also be "a particular musical line, even if this is intended for an instrumentalist and not a singer" [8]. The latter is often a **monophonic** (one note at a time) sequence of musically related notes.

1.2.2 The piano

The piano has been invented in 1698 by Bartolomeo Cristofori [6]. The goal of this instrument was to improve the harpsichord, which was a traditional instrument at this time, to make it possible to play from *piano* (that is, with low velocity) to *forte* (with high velocity), hence the name pianoforte, shortened to piano. Nowadays, many types of pianos exist, such as grand piano, upright piano, electric piano, and so on, all with their own timbre.

The piano is part of the keyboard family with struck strings. It has the interesting property that once a note is played, the only influence the musician can have on it is either to stop it by releasing the key (possibly partially, but it is very rare), or to play it again while it is still active. The piano also has a **sustain pedal** which, when pressed, makes it possible to release the keys without stopping the sounding of the notes. The notes are therefore active until the sustain pedal is released. Generally, when used in classical and traditional pieces, the sustain pedal is pressed during most of the piece. It is regularly released and pressed again immediately just before the next onset to stop all notes on a regular basis. This mixes beautifully the notes without having too many inharmonicities (notes sounding bad when played together).

1.3 Music types

Music comes in many forms, from jazz to heavy metal through disco, varying in the instruments they use, the rhythms, the chords, ... In [9], hundreds of these are linked in a graph (figure 1.2), but these are only the popular music styles, and many more exist. A single style can encompass several types of pieces. For classical music, we can have concertos (for orchestra), duet (two musical voices), etude (initially intended to practice technical aspects of piano playing), and so on.

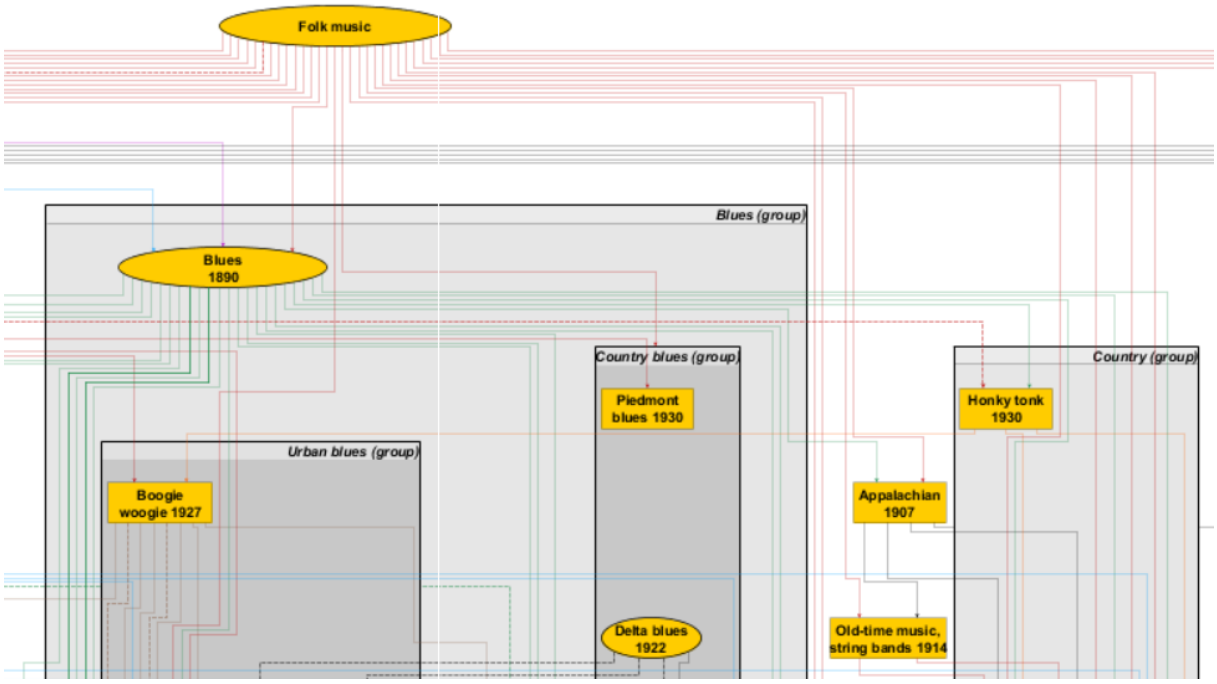

Figure 1.2: Small part of the graph of [9].

One of the most famous types of music for piano is **classical music**, the music from "a period in the history of music, roughly from 1750 to 1830: the age of Haydn, Mozart and Beethoven" [8], and we will therefore focus on this particular style throughout this thesis.

1.4 Writing down music

As for everything, writing music down was at some point necessary to transmit it to future generations. Different formats exist to describe music, some being specific to a particular instrument. Three of them are of particular interest for this thesis, in increasing order of abstraction: the digitised raw waveform, the Musical Instrument Digital Interface (MIDI) format and the more traditional sheet music format. These representations are the musical counterparts of common language representations, namely a recording of a speech, a timed list of pronounced syllables with intonations and a complete well-structured and punctuated text.

1.4.1 Digitised raw waveform

As, physically speaking, music is only an oscillating variation in the pressure of the air, we can play the music and measure the pressure over time for this **musical performance** (a particular execution of a piece of music). This waveform representation is often used in the signal processing field. To be used by a computer, the values have to be discretised and the signal sampled. This will be described more in details in section 2.1.

1.4.2 Musical Instrument Digital Interface (MIDI) format

MIDI [10] is a whole protocol for music communications in which onset, offset, pitch, velocity, instrument, ... of each note can be specified, pedal indications can be precised, and much more. However, in the context of this thesis, we will use only the parts related to the piano roll representation: onsets, offsets and pitches information.

The MIDI format has therefore all the necessary information for a musician to play a piece of music. However, it is not really convenient, hardly human-readable, and it misses the general structure of the music. Musicians who have not learned to read sheet music sometimes use a graphical representation of it such as a piano roll representation.

A **piano roll** is a continuous roll of paper with holes in it, each hole representing a played note whose pitch and timing depends on the position of the hole. It was used for automatic pianos, or pianola. The piano roll representation is the simplest representation of music. The horizontal axis represents time and the vertical axis represents pitch. A played note is, therefore, represented as a line at the appropriate place. A piano roll representation thus contains the information about onsets, offsets and pitches, but not the velocity of each note, the presence of sustain pedal, the general structure of the music, and so on.

1.4.3 Sheet music

This representation is the one taught in music reading lessons. Most of the time, pieces of music are defined through this representation, particularly for classical piano pieces. To have a beautiful representation, the sets of possible pitches and rhythms have been fixed, transforming the continuous possibilities of music into discrete ones. Then, each of these pitches and rhythms has been assigned its own representation.

Set of pitches

As already explained, some pitches sound better together than others, and particularly the pitches whose ratio can be represented with small integers, such as 2:1 (an **octave**) or 3:2 (a **perfect fifth**). The set of possible pitches has been defined from these two ratios, starting at 440Hz. As $(2)^7 \approx (3/2)^{12}$, twelve pitches are defined on a logarithmic scale from 440Hz to 879Hz. Each of these pitches can be multiplied by a power of 2 to obtain the pitches of higher or lower notes. Generally, every note has a pitch $2^{1/12}$ times higher than the previous one (**equal temperament**), preserving a precise octave ratio and approximating all perfect fifth

ratios. Other temperament exists and are described in section 2.1.2 of [11], but this is of minor importance for this thesis.

The octave interval is thus divided in twelve **semitones**. A **natural minor scale** is a set of notes defining intervals following the pattern 2-1-2-2-1-2-2 semitones. When played in increasing order produces a perceptually natural set. Starting from 440Hz, these seven notes are the **natural notes**, the other five being obtained by adding or subtracting a semitone from these. This operation is done via **accidentals**, adding a semitone with **sharps** ♯ and subtracting a semitone with **flats** ♭.

The seven natural notes are named with a letter (one per natural note) and a digit (representing the octave). The 440Hz note is named A_4, and is followed by $A^\sharp{}_4$, B_4, C_5, $C^\sharp{}_5$, D_5, $D^\sharp{}_5$, E_5, F_5, $F^\sharp{}_5$, G_5, $G^\sharp{}_5$, A_5, ..., each separated by a semitone. Note that $A^\sharp{}_4 = B^\flat{}_4$ and $B^\sharp{}_4 = C_5$.

The piano is designed to play notes from this set only, from A_0 to C_8, corresponding to **midi pitches** 21 to 108. It is composed of 52 white keys for the natural notes and 36 black keys for the other ones, for a total of 88 different pitches.

Set of rhythms

As with pitches, a continuous time dimension is not handy, and it has been discretised. The unit is most of the time the **quarter note** ♩, and symbols and names exist for fractions or multiples of it. The actual duration of a quarter note has to be defined for each piece of music or is left to the interpretation of the musician. While some pitches cannot be named using the consensual set of notes described here above, any rhythm can theoretically be approximated as precisely as desired using the appropriate addition of rhythmic notations.

Sheet music

Sheet music resembles the piano roll representation in the sense that the vertical position of a note on the sheet is related to its pitch while the horizontal position is related to the time at which the note begins. However, to ease the reading, five horizontal lines are drawn, called a **staff**. Each line and each inter-line represents a natural note. The staff begins by a clef to indicate which pitch is represented by each line. Piano sheet music is most of the time composed of two staves, one for the high pitches and one for the low pitches. The staff for the high pitches is, therefore, introduced by a **treble clef** and the other by a **bass clef**. If a note is higher than the highest line or lower than the lowest line, additional lines are drawn to improve readability. The rhythm of the notes is defined by their shape. When a piece of music is **polyphonic** (can contain simultaneous notes), the notes that are played at the same time are generally vertically aligned. An example is shown in figure 1.3.

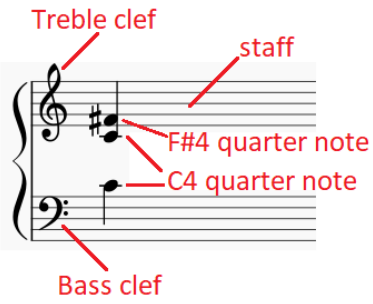

Figure 1.3: Toy example of piano sheet music.

If you observe figure 3.4 from section 3.3.2, you can see five additional musical notations.

- The first is (♩= 96), which indicates the duration of a quarter note. It specifies that 96 quarter notes are played each minute.
- The second is the three flats at the beginning of each staff, called the **key signature**. It indicates that half a tone will be subtracted from all notes on the staff at these pitches (or octaves of these pitches).
- The third is $\frac{4}{4}$, called the **time signature**. It indicates that the sheet music is divided into measures whose duration is equal to four quarter notes. $\frac{3}{2}$ would have meant a measure

lasts 3 half notes, the 2 indicating the unit of time is the half note instead of the quarter note. These digits are sometimes replaced by a symbol for well-known time signatures.

- The fourth is consisting of vertical lines. These **barlines** are separating the measures, making it easy for the musician to follow the sheet.
- The last is the symbols on the staves which are not notes. In the last measure, we can see three of these. These are **rests**, meaning no note is played for a given duration. They are needed to let the musician know when he has to play the next note. The shape of a rest defines its duration.

Of course, there exist hundreds of other signs, to make sheet music more readable in case of special rhythms, to indicate the velocity of a note, to avoid going too high above the staff, to avoid rewriting the same things when there is a long repetition, and so on.

The careful readers will have noticed that a music piece does not have a unique way to be written, as some notations are equivalent to others. It makes the transcription harder since the transcriber has to choose the best way to write the music.

1.5 Conclusion

Music is a very developed cultural heritage, in many ways similar to the language. Some people even say music *is* a language, the most universal one. Of course, an entire language cannot be completely described in a single chapter of a master thesis, but its basic vocabulary and its main features have been introduced, giving the reader all the necessary tools to understand the musical aspects of this thesis.

The goal of the thesis is to map any digitised raw waveform of traditional and classical piano performances to the corresponding piano roll representation. Some of the difficulties to define this mapping have been introduced, such as the varying timbre from an instrument to the other and the overlapping harmonics.

Chapter 2

Technical background used in APT

Music processing is a small subset of the more general signal processing field. As such, many state of the art techniques for APT use well-known spectral transformations such as the Short-time Fourier transform (STFT) and mathematical models like the Hidden Markov Model (HMM). With the rise of machine learning and its impressive success in other fields, many recent approaches try to apply the successful architectures to the particular case of APT. This section will explore the technical background to help the reader to easily understand the next chapter about state of the art approaches.

2.1 Common signal transforms

As already explained in section 1.4.1, a musical performance can be described entirely through the time-varying air pressure measurement. This description conventionally uses the x-axis for time and the y-axis for pressure difference, or **amplitude**. Since computer cannot handle continuous data, these measurements have to be discretised. Given the human hearing bandwidth goes up to ∼20 kHz, the Nyquist–Shannon sampling theorem says that the sampling must be at least two times higher to avoid losing useful information or having aliasing, that is, 40 kHz. For **CD-quality**, the convention is to sample the music at 44.1kHz with 16 bits per sample to describe the amplitude, and to measure the amplitude in two different places to have a left track and a right track for stereo. The files encoded in the `wav` format use this kind of temporal representations.

2.1.1 Spectral domain

However, the amplitude of a sound at a particular time is only a consequence of all the surrounding vibrations. The sound is fundamentally perceived as a collection of frequencies. To measure a particular frequency, one could want to compute the correlation[1] between the signal and a sine waveform with the good frequency. However, two sine waveforms with the same frequency can have a correlation of 0 if their phases are shifted adequately, as shown in figure 2.1. This is where complex numbers are handy. They make it possible to measure the correlation for every possible shift at once, including the one maximising this correlation.

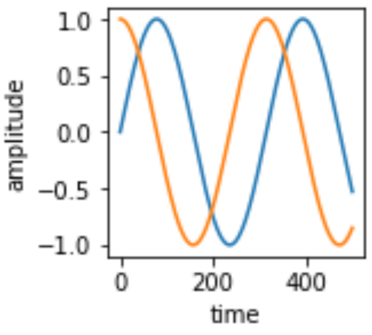

Figure 2.1: Uncorrelated sine waveforms with the same frequency.

The idea is to make a point move on the complex plane. It starts at the origin in eastern direction (positive real)

[1]For centred normalised signals f and g, the correlation is $\sum_n f(n)g(n)$.

and its direction rotates at the wanted frequency while its speed is described by the signal. That is, the speed at which the point moves at any time t is equal to the amplitude of the signal at that time t. At the end of the process, the further it is from the origin, the more the frequency is present in the signal, with the optimal phase for the sine waveform being related to the direction from the origin to the final point. Figure 2.2 shows a discretisation of the process. We can observe as expected that when the frequency of the signal is not the same as the analysed frequency, the result is near the origin, and conversely. Mathematically, this process can be formulated as

$$X(\omega) = \sum_n x(n) e^{i\omega \frac{2\pi}{f_s} n} \tag{2.1}$$

with $X(\omega)$ a complex number whose magnitude is the activation of a particular frequency ω in the signal x, $x(n)$ the value of the n^{th} sample of the signal, $e^{i\omega \frac{2\pi}{f_s} n}$ the direction in the complex plane for frequency ω at time $t = \frac{n}{f_s}$ and f_s the sampling frequency. The final point of the process is the complex number obtained through the formula, with its magnitude proportional to the presence of the frequency.

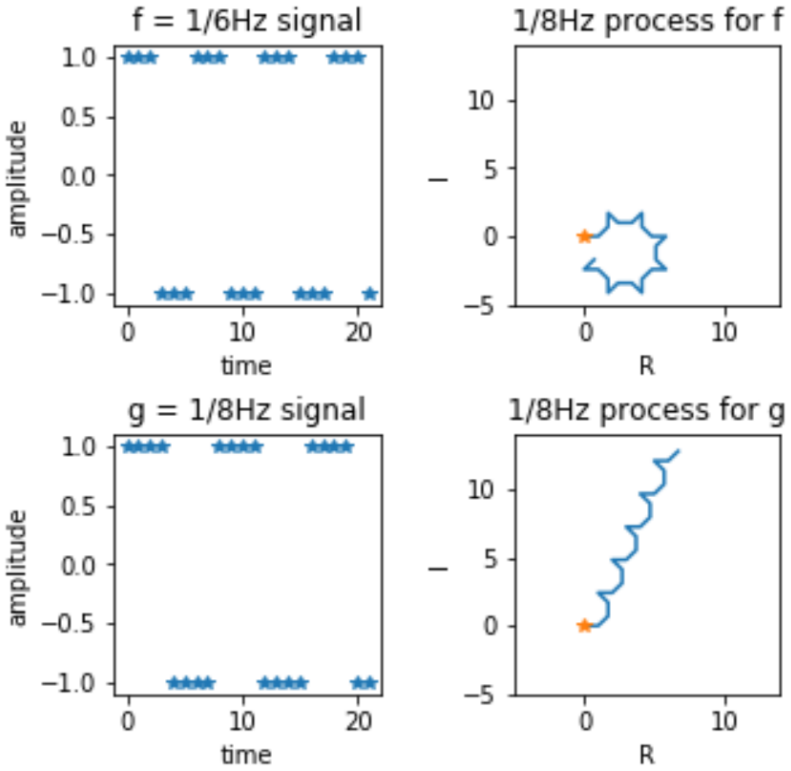

Figure 2.2: Discrete toy example of the process described in section 2.1.1.

This formulation makes it possible to quantify the frequencies activation of a signal, and therefore to express it in the spectral domain. The **spectral domain** describes the strength of each frequency for the whole signal, whereas the **temporal domain** describes the amplitude at each time for the whole range of frequencies.

2.1.2 Fourier Transform

The most known transform from the temporal to the spectral domain is the Fourier Transform, efficiently implemented in $O(n \log(n))$ by the Fast Fourier transform (FFT) algorithm for signals with a length being a power of two[2]. It computes the frequency activation for linearly spaced

[2]An algorithm has also been developed whose efficiency increases with the divisibility of the signal length into the product of smaller integers. [12]

frequencies, such that complete reconstruction of the original signal is possible, transforming it into a sum of sine waves. Details can be found in [13]. As we work with computers, we are only interested in the Discrete Fourier transform (DFT), which is formulated as

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-ik\omega_0 n} \qquad \forall k \in \{0, 1, ..., N-1\} \tag{2.2}$$

with x a signal of length N and $\omega_0 = \frac{2\pi}{N}$. Comparing with formula 2.1, we observe that $\omega \frac{2\pi}{f_s} = k\omega_0 \iff \omega = k * \frac{f_s}{N}$. The analysed frequencies therefore depend on f_s, such that this f_s term disappears from the coefficients formula. The reconstruction is obtained through the inverse Fourier transform

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{ik\omega_0 n} \qquad \forall n \in \{0, 1, ..., N-1\} \tag{2.3}$$

Generally, when analysing the resulting $X[k]$'s, only the magnitudes of the complex numbers are of interest, as they correspond to the correlations with the optimal shifts as introduced in section 2.1.1.

The FFT is widely used in many fields thanks to its speed, partially because convolving in the temporal domain is equivalent to multiplying in the spectral domain. However, in some fields such as music processing, it has a strong weakness: we can know the frequency of e.g. 440Hz is very present in a piece of music, but we do not know *where* it is present. A solution to that is to partition the signal in small time intervals and to do the FFT piecewise (called the Short-time Fourier Transform (STFT)). The smaller the time interval, the higher the temporal resolution, but the lower the spectral resolution. This last transform is 2-D, with a time axis *and* a frequency axis. It therefore results in a **spectro-temporal** representation. In practice, the STFT commonly uses window functions to smooth the frame boundaries (see figure 2.3).

2.1.3 Constant Q-transform

A weakness of the STFT when applied to some musical signal is that the measured frequencies are linearly spaced whereas the traditional pitches are defined as a logarithmic sequence. This means we would have many frequency measures for low pitches, but very few for the high ones.

The Constant Q-transform (CQT) [14] proposes logarithmically spaced frequency measurements. The general idea is to do the process described in section 2.1.1 for chosen logarithmically spaced frequencies, but also at chosen places of the signal thanks to a windowed signal (see figure 2.3).

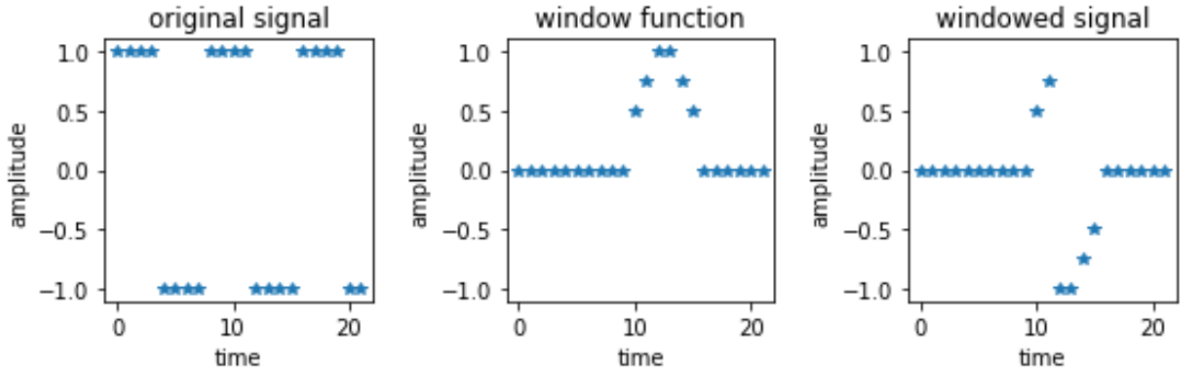

Figure 2.3: Toy example of a windowed signal.

This can be mathematically formulated as

$$X^{CQ}(k,n) = \sum_{j=n-\lfloor N_k/2 \rfloor}^{n+\lfloor N_k/2 \rfloor} x(j) a_k^*(j-n+N_k/2) \quad \forall k \in \{1,2,...,K\} \tag{2.4}$$

$$a_k^*(n) = \frac{1}{N_k} w(\frac{n}{N_k}) e^{i f_k \frac{2\pi}{f_s} n} \tag{2.5}$$

with $X^{CQ}(k,n) \in \mathbb{C}$ the CQT coefficient for frequency f_k around sample n of the discrete temporal signal x, $\lfloor \cdot \rfloor$ the rounding towards $-\infty$, $N_k \in \mathbb{R}$ a parameter, $a_k^*(n)$ the weighted direction in the complex plane at time $t = \frac{n}{f_s}$ for frequency f_k, $w : [0,1] \to \mathbb{R}^+$ a continuous window function, f_k the frequency corresponding to k and f_s the sampling frequency of the signal x. The goal of the $\frac{1}{N_k}$ factor is to have $\sum_n \frac{1}{N_k} w(\frac{n}{N_k}) \approx \int w(t)dt$ for all N_k, in order to have comparable results between different frequencies.

This transform uses many parameters, whose optimal and acceptable values have been studied in [15] with respect to signal reconstruction, signal coverage, fast computations, and so forth. These parameters are:

- f_k's. These are the measured frequencies. As they are logarithmically spaced, we only have to know f_1 (or $\mathbf{f_{min}}$), a constant B - the number of **bins per octave** - and K (or $\mathbf{n_{bin}}$) to have all frequencies: $f_k = f_1 2^{\frac{k-1}{B}} \ \forall k \in \{1,2,...,K\}$. These can be chosen by the user depending on the application.

- N_k's. These are the widths of the temporal windows for the different frequencies. The optimal values are $N_k = \frac{q f_s}{f_k(2^{\frac{1}{B}}-1)} \overset{B \geq 12}{\approx} \frac{q f_s B}{\log(2) f_k}$, with $0 < q \leq 1$ a **scaling factor**. The effective frequency resolution will be equivalent to $q * B$ bins per octave. $q < 1$ therefore leads to an oversampling of the frequency axis.

- n's. Doing the CQT computation for every possible n is not computationally realistic nor interesting. It is more interesting to do it for one every H_k samples (**hop size**). Optimal values are $0 < H_k \overset{<}{\approx} \frac{1}{2} N_k$ to analyse the whole signal and not just regularly spaced parts of it. In practice, it is more convenient to use a fixed hop size $H_k = H$ for all k to have a 2-D matrix with the same number of coefficients for each frequency.

- $w(\cdot)$. It is the window function. The default one is the Hann window function, which is a simple (scaled and translated) sine (see figure 2.4)

- f_s. It is the sampling rate of the original signal. The Nyquist–Shannon sampling theorem imposes $f_s \geq 2 * f_K$.

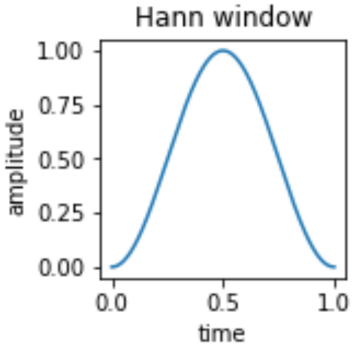

Figure 2.4: Hann window.

2.1.4 MEL spectrogram

As the human perception of sound is not exactly logarithmic, another scale has been invented from experiments, the **MEL scale**, with the **mel** as unit. In fact, many different MEL scales have been proposed, with little variations between each, but only one will be described and used in this thesis, the one used within the Python library `librosa` [16].

Some experiments showed that the human perception of pitches is linear until 1000Hz, then logarithmic. The MEL scale thus follows this, with the mathematical formula being

$$x \text{ Hz} = \begin{cases} \frac{3x}{200} \text{ mel} & when \quad x \leq 1000 \\ 15 + 27\ln(x/1000)/\ln(6.4) \text{ mel} & elsewhere. \end{cases} \tag{2.6}$$

We observe the left-derivative is almost the same as the right derivative at $f_{Hz} = 1000$, being respectively 0.015 and $\frac{27}{1000\ln(6.4)} \approx 0.0146$. Those values are close, as hearing is somewhat continuous, but not exactly the same, as the scale is experimental.

`Librosa` computes the coefficients for a spectrogram using MEL scale as

$$X^{MEL}(k,n) = \sum_{j=0}^{n_{fft}-1} |X_n|^2(j) A_k^*(\frac{f_s}{n_{fft}} j) \tag{2.7}$$

with $|X_n|$ the magnitude of the DFT of signal x around n with n_{fft} samples and $A_k^* : \mathbb{R} \to \mathbb{R}$ a triangular window function over the spectral domain going from 0 at f_{k-1} to V_{max}^k at f_k and back to 0 at f_{k+1}. $V_{max}^k \in \mathbb{R}$ is tuned to have approximately constant energy for each channel, that is, $V_{max}^k = \frac{2}{f_{k+1} - f_{k-1}}$. $\frac{f_s}{n_{fft}} j$ is the frequency corresponding to the j^{th} DFT coefficient, with f_s the sampling frequency of x. The f_k's are linearly spaced on the MEL scale. As for the CQT, the hop size, f_{min}, f_{max} and n_{bin} can be chosen. To compare this with the CQT formula 2.4, it is useful to know that

$$\sum_n x(n)a(n) \propto \sum_j X(j)A(j) \tag{2.8}$$

X and A being the DFT of x and a [15]. This stems from Perceval's theorem [17]. We therefore see that for the same n and f_k, it is the same formula up to the shape of the window function and a magnitude spectral squaring of the signal.

2.1.5 Spectral template of a note

When an atomic signal such as a piano note in our context is expressed in the spectral domain, the result is called the **spectral template** of the atom. The atomic signal can also be expressed in the spectro-temporal domain, leading to a **time-variable spectral template** of this atom.

When analysing the time-variable spectral template of an A4 piano note using a logarithmic frequency axis (see figure 2.5), we see that multiples of the fundamental frequency are very present, as underlined by the white lines. This was expected from the properties of a vibrating string. This is one of the major difficulties of APT, as it is hard to say if these harmonics are themselves played notes or only harmonics.

We also see that just after the beginning of the note, all other frequencies are very present too. This is due to the fact that when the string is struck, it is rather noisy at first before starting to produce the expected frequencies, as it is the sound of the striking hammer and not yet the vibrating one of the string. This is particularly useful to detect onsets.

2.2 Hidden Markov Model

A Hidden Markov Model (HMM) is a statistical model that offers a very rich modelling framework, and that works very well in practice for several important applications [19]. For APT, HMMs are sometimes used as a post-processing step, as detailed at the end of section 3.1. A HMM is based on a Markov process, except that the states are hidden and only the output is observable.

More formally, a HMM can be characterised by the following [19]:

- N: the number of states in the model.

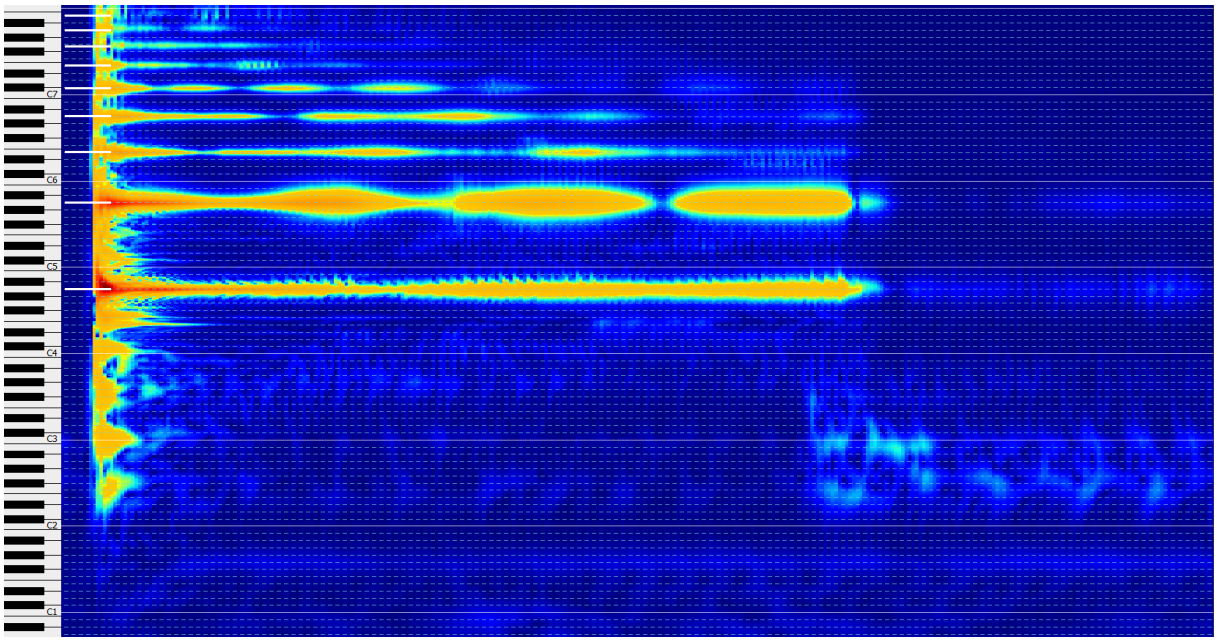

Figure 2.5: Spectrogram of a 2 seconds A4 piano note (from MAPS database MAPS_ISOL_NO_F_S1_M69_ENSTDkAm [18]) produced by the Anthem Score software [2].

- $S = \{S_i\}$ with $1 \leq i \leq N$: the set of states, with the state at time t denoted q_t.
- M: the number of distinct observation symbols.
- $V = \{v_i\}$ with $1 \leq i \leq M$: the set of symbols (also called alphabet).
- $A = \{a_{ij}\}$ with $1 \leq i, j \leq N$: a state transition probability distribution, describing the transition probability from state i to state j, that is:

$$a_{ij} = P[q_{t+1} = S_j | q_t = S_i] \tag{2.9}$$

- $B = \{b_j(k)\}$ with $1 \leq j \leq N$ and $1 \leq k \leq M$: the observation symbol probability distribution in state j, that is:

$$b_j(k) = P[v_k \text{ at } t | q_t = S_j] \tag{2.10}$$

- $\Pi = \{\pi_i\}$ with $1 \leq i \leq N$: the initial state probability distribution, that is:

$$\pi_i = P[q_1 = S_i] \tag{2.11}$$

The probability of an observed sequence $O = O_1 O_2 ... O_T$ for a defined model is, therefore,

$$P(O) = \sum_{\overline{q}} \Big(P(\overline{q}) * P(O|\overline{q}) \Big) \tag{2.12}$$

$$P(\overline{q}) = \pi_{q_1} \prod_{i=1}^{T-1} a_{q_i, q_{i+1}} \tag{2.13}$$

$$P(O|\overline{q}) = \prod_{i=1}^{T} b_{q_i}(O_i) \tag{2.14}$$

with $\overline{q} = q_1 q_2 ... q_T$ a state sequence. The HMM model can be used to find the most probable state sequence given an observation, changing the $\sum_{\overline{q}}$ into an $\text{argmax}_{\overline{q}}$ in the previous formula. For the toy example in figure 2.6, the state sequence $S_2 S_1 S_1$ with observation 'bba' will have a probability of $(\pi_{S_2} * a_{S_2,S_1} * a_{S_1,S_1}) * (b_{S_2}(\text{'b'}) * b_{S_1}(\text{'b'}) * b_{S_1}(\text{'a'})) = (0.8 * 0.5 * 0.9) * (0.6 * 0.7 * 0.3)$

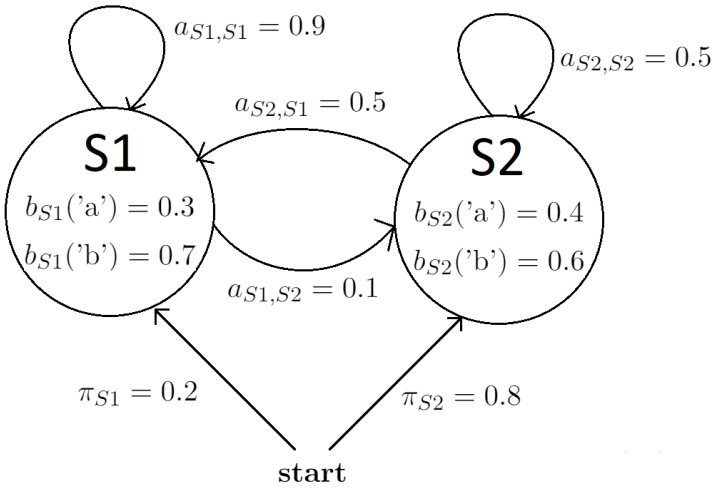

Figure 2.6: Toy example of a HMM.

2.3 Machine learning basis

The rise of machine learning in the past decades has proved to be a revolution in many fields, enabling computers to "learn" how to solve several tasks from examples, without the need for an explicit programming of the solver. In this section, we will describe several components of the machine learning techniques used throughout this thesis.

2.3.1 Definition

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [20]. In other words, machine learning aims at improving its performance at a task by accumulating experience.

2.3.2 Supervised learning

The goal of supervised learning is to learn a function that maps an input to an output. This learning task is based on a labelled dataset, composed of input-output pairs called training examples. After learning this inferred function, it can be used to map new inputs. In this thesis, we use the MAPS dataset described in chapter 4 which is composed of audio recording - MIDI transcription pairs, in order to learn the function that maps audio representation to MIDI representations.

2.3.3 Artificial Neuron

An artificial neuron is a mathematical function inspired by the way biological neurons work. They are the building blocks of Artificial Neural Networks (ANNs), which are defined in section 2.3.6. As illustrated in figure 2.7, an artificial neuron takes as input a vector $\mathbf{x} \in \mathbb{R}^N$ and is characterised by a vector of weights $\mathbf{w} \in \mathbb{R}^N$ and a bias $b \in \mathbb{R}$, with N the size of the vectors. The neuron adds the bias to the dot product of the two vectors and the resulting value goes through a non-linear activation function (see the following section) to get the output $y \in \mathbb{R}$. The mathematical function implemented by artificial neurons can thus be summed up as

$$y = activ(\sum_i \mathbf{x}_i \cdot \mathbf{w}_i + b) \tag{2.15}$$

with $activ(\cdot)$ the activation function. The learning determines the values of $\mathbf{w}$ and b, also known as parameters.

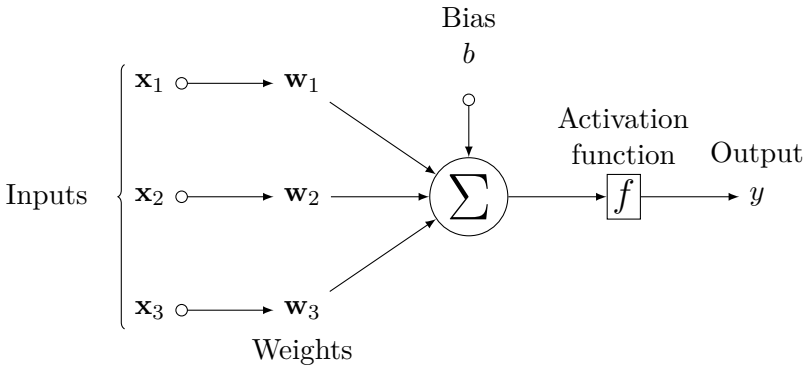

Figure 2.7: Diagram of an artificial neuron. [21]

2.3.4 Activation functions

Without an activation function, artificial neurons would essentially be a linear regression model [22]. Introducing a non-linear function in the model makes it able to learn more complex tasks. It is compatible with the backpropagation technique used during the learning phase, as the activation functions are differentiable, supplying the gradients along with the error. A good explanation for the backpropagation algorithm and its challenges can be found in [23].

Many activation functions exist and are widely used. We will go through a few usual activation functions that were considered or used in the context of this thesis. These functions are shown in figure 2.8.

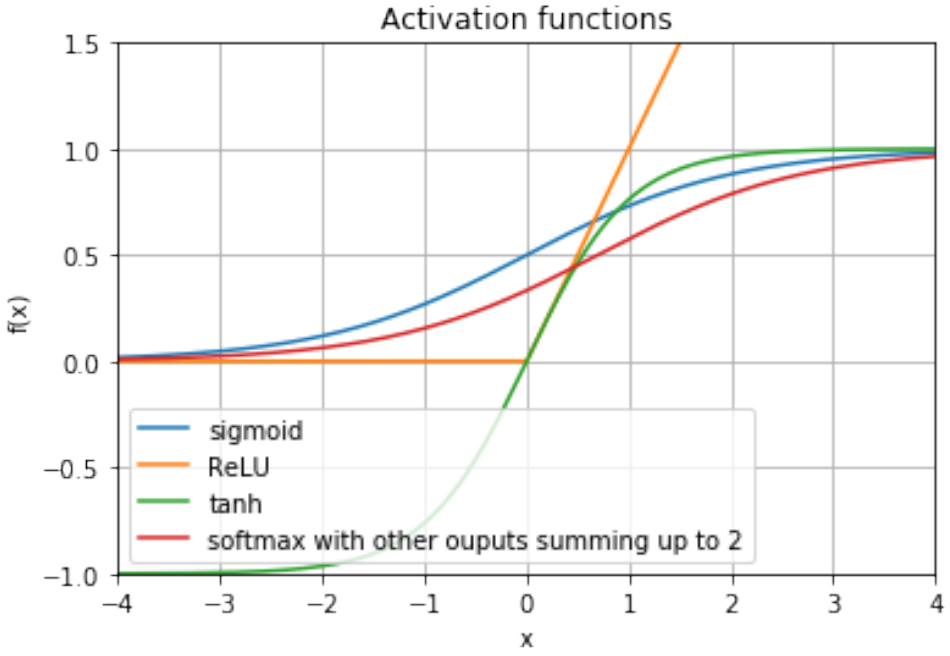

Figure 2.8: Activation functions.

Sigmoid

The sigmoid activation function is widely used since it is continuously differentiable and results in a number between 0 and 1. The sigmoid function is defined as

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{2.16}$$

Hyperbolic tangent

The hyperbolic tangent activation function has the same properties as the sigmoid one but results in a number between -1 and 1. In fact, it is a scaled version of it. The hyperbolic tangent is defined as

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} = 2sigmoid(2x) - 1 \tag{2.17}$$

Rectified linear unit

The rectified linear unit (ReLU) is the most used activation function since it is able to generate sparse models (whenever $x \leq 0$) and it reduces the likelihood of vanishing gradient [24] observed with the sigmoid activation function as its gradient is always 1 or 0. A detailed explanation of the vanishing gradient problem can be found in [23]. The ReLU is defined as

$$relu(x) = max(0, x) \tag{2.18}$$

Softmax

Sigmoid and ReLU are particularly appropriate for binary classification and as intermediate activation functions. The softmax activation function is used for multi-class outputs, assigning a probability to each class, summing up to 1. The normalisation is done by dividing the output associated to each class by the sum of all outputs. For C classes, the softmax function is defined as

$$softmax(x_c) = \frac{e^{x_c}}{\sum_{i=1}^{C} e^{x_i}} \text{ with } 1 \leq c \leq C \tag{2.19}$$

If the outputs associated to all other classes (before normalisation) sum up to 1, the softmax function is the same as the sigmoid.

2.3.5 From neurons to layers

To build further abstraction, artificial neurons are grouped to form **layers**. ANNs are built by stacking these, as explained in section 2.3.6. There are different types of layer, each having advantages and drawbacks.

Fully connected layer

The most basic layer type is the fully connected layer, also called dense layer. Its particularity is that each neuron composing it has as input the outputs of all neurons from the previous layer, as shown in figure 2.9. The number of neurons and the used activation function are hyperparameters to be set by the programmer. The operation implemented by this layer is essentially a simple matrix product.

Convolutional layer

The mathematical operation at the heart of the convolutional layer is the convolution. As seen in figure 2.10, the element ij of the matrix resulting from a convolution of two matrices **I** and **K** is the summed element-wise product of the second matrix, or **filter**, with the sub-matrix from the first matrix centred in ij. It is also noteworthy that two types of padding are possible: either we take only **valid** submatrices and the resulting matrix is smaller than the original one, or we apply a zero padding, meaning that out-of-matrix entries are set to 0, and we obtain a **same** sized matrix in the end.

There are two main characteristics of the convolutional layer that differ from fully connected layers. The input of a neuron of the convolutional layer is composed of the outputs of a constant number of neurons in a local region of the previous layer, and the weights and bias of a neuron are used for other neurons too. An illustration comparing the fully connected layer and the convolutional layer is shown in figure 2.9. The colours of the lines in the right figure illustrate the reused weights. The illustrated convolutional layer has only one filter. The operation implemented by the convolutional layer is essentially one matrix convolution per filter.

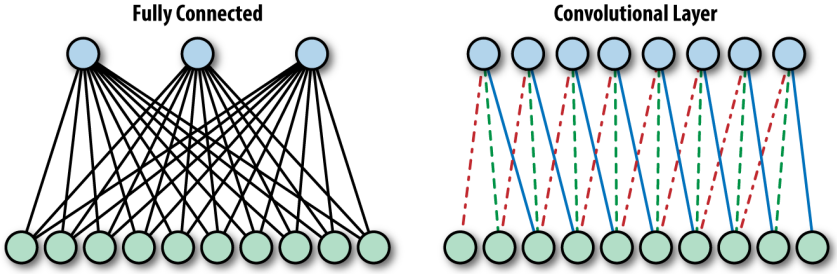

Figure 2.9: "In a fully connected layer (left), each [neuron] is connected to all [neurons] of the previous layers. In a convolutional layer (right), each [neuron] is connected to a constant number of [neurons] in a local region of the previous layer" [25]. In this representation, the information flows from bottom to top.

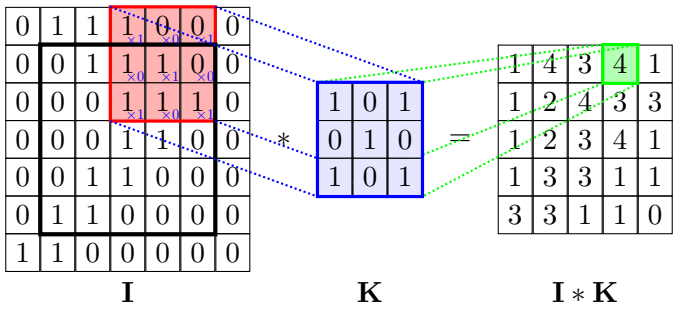

Figure 2.10: Illustration of the convolution operation with valid padding. [26]

Introduced in [27], convolutional layers are thus characterised by a set of filters whose weights are learned. These filters are characterised by their dimensions, also called **shape** (3x3 in the example shown in figure 2.10). If we do not want to have a result for all possible submatrices, we can set a **stride** of e.g. (2,3), meaning we take only one out of 2 submatrices in the first dimension and one out of 3 in the second, for a resulting matrix ∼2x3 times smaller.

This type of layer exploits the locality of information in its inputs. It also assumes a form of shift invariance in the input, as it uses the same weights for each position. It is very useful for images for example, as neighbouring pixels are more likely to carry information about the same object or structure than distant pixels and as the absolute position of the inputs is of small importance. Using one convolutional layer, the learned filters represent simple abstractions,

e.g. edges. Multiple convolutional layers are typically used in order to build up the level of abstraction and detect more complex shapes like eyes and then faces.

Long short-term memory layer

So far, we have only described layers belonging to feed-forward networks, where information flows from input to output, layer after layer, always in the same direction. Recurrent layers have information flowing both ways, by introducing loops in the architecture. The output of certain layers can be reused by said layers as input, giving the layer some kind of state or memory. This property makes recurrent layers particularly fit for sequential data labelling, classification and prediction.

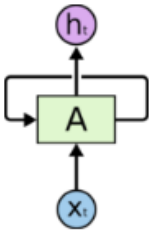

Figure 2.11: Basic representation of a recurrent layer. [28]

The most basic representation of a recurrent layer is shown in figure 2.11, with A a repeating module corresponding to a layer, x_t an input observed at time t and h_t the output at time t. There is also a loop that indicates that an output s_t of the layer is reused. More specifically, the state output s_0 corresponding to the input x_0 is used in combination with x_1 to output h_1 and s_1, as shown on figure 2.12. Because the output depends on the previous input, it can be used to model sequential data. A representation of the standard repeating recurrent layer module is shown in figure 2.13, in which the previous output of the module is concatenated with the current input and fed through a fully connected layer with hyperbolic tangent activation function to produce the current output.

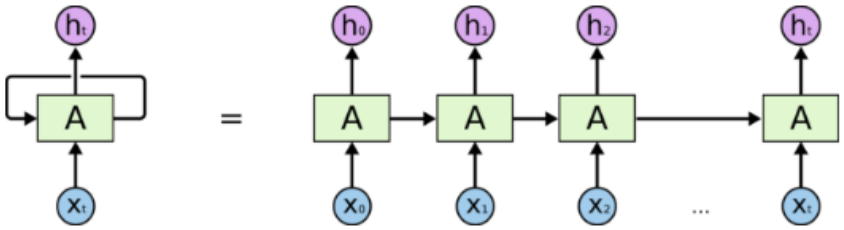

Figure 2.12: Expanded representation of a recurrent layer. [28]

This kind of recurrent layer is good for short-term dependencies. In order to model long-term dependencies, Long Short-Term Memory (LSTM) layers were introduced in [29]. In the modules used in LSTMs, there are three gates that handle the input and previous state, from left to right in figure 2.14:

- the **forget gate** that controls how much of the current state will be kept.
- the **input gate** that controls what is added to the state.
- the **output gate** that controls how much of the updated state will be used in the output.

This kind of architecture can learn short- and long-term dependencies. For more details, we recommend reading Christopher Olah's excellent blog post on the subject [28]. This architecture

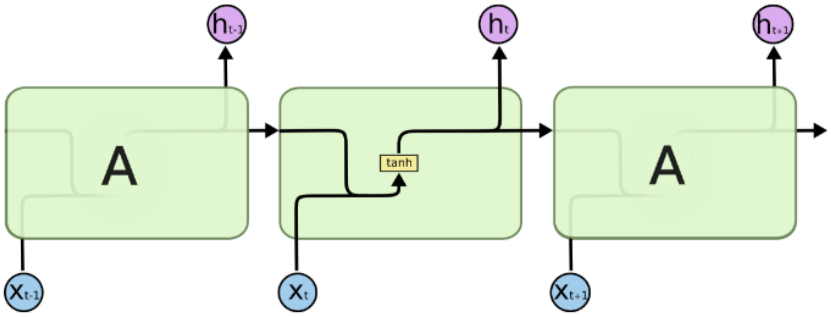

Figure 2.13: Standard recurrent layer repeating module. [28]

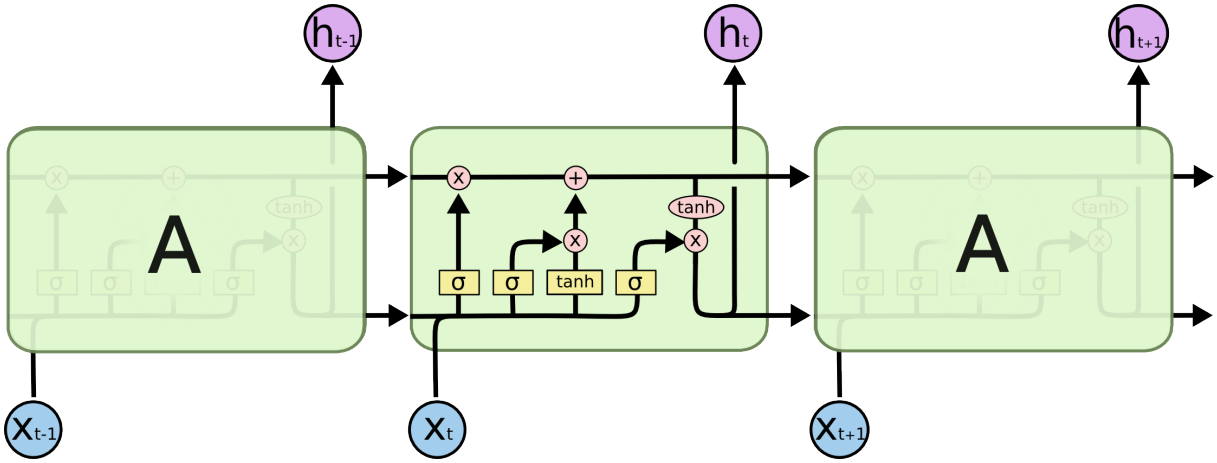

Figure 2.14: Repeating module in an LSTM. [29]

can be upgraded to a bi-directional LSTM (BiLSTM). Originally introduced in [30], BiLSTM are building dependencies with past outputs as well as with future outputs.

Pooling layer

In order to reduce the number of parameters in subsequent layers and reduce overfitting[3] [31], pooling layers are introduced every so often between other layers. It works in the same fashion as the convolutional layer with stride, except it usually takes the maximum (or another statistic) of the inputs, as illustrated in figure 2.15. This layer does not learn anything during training.

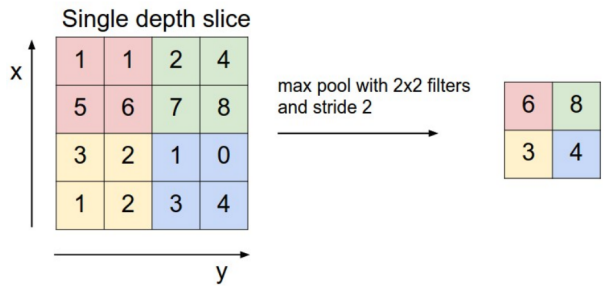

Figure 2.15: Illustration of the max pooling mechanism, with 2x2 filters and a stride of 2. [31]

[3] A model overfits if it learns perfectly its training set but is unable to generalise to unseen inputs.

Dropout layer

Introduced in [32], dropout layers randomly deactivate a fraction of their input during the training phase, following the **keep_rate** hyperparameter. In the testing phase, the neurons' outputs are multiplied by *keep_rate* to keep the same order of magnitude in total. This technique makes for a better generalisation as it reduces co-dependency between the neurons, and in turn reduces overfitting, as demonstrated in [32].

Batch normalisation layer

Introduced in [33], Batch Normalisation is a technique that consists in normalising the input in a hidden layer. The normalisation is done by subtracting the mean of the current training batch and dividing by its variance. The uniform scaling of the inputs makes for a faster convergence during the learning phase and adds a slight regularisation effect, as mentionned in [33]. Some parameters γ and β are learned to let the model scale the inputs as wanted, multiplying these by γ and adding β after the initial normalisation.

2.3.6 Artificial Neural Networks

This brings us to Artificial Neural Networks (ANNs), a family of learning algorithms inspired by the way the brain works. These algorithms are built using, amongst others, the layers previously described. As seen in figure 2.16, the network is built layer by layer. On the left, the input data is fed to an input layer. One or more hidden layers follow the input layer. Finally, an output layer composed of one or several neurons completes the network. The architecture of a model is its network layout (defined by the number of layers, their composition and connections, ...), while its parameters are the learned variables such as the weights associated to each link in the network. The architecture is generally fixed a priori.

The learning is usually done by feeding the network with examples of input-output pairs grouped in **batches**. The size of the batch is a hyperparameter to be tuned. From these batches, we use gradient descent[4] variants on a well-chosen loss function thanks to the backpropagation algorithm (more details in [34]) to learn the parameters, or variables. This learning part will not have a big influence on the thesis and will therefore not be explained further, but curious readers can find more details in the Google course in [35]. The gradient descent is applied on the **parameters**, while the **hyperparameters** have to be tuned manually. ANNs are widely used in computer vision, speech recognition, natural language processing and many other domains.

2.3.7 Architectures

Deep Neural Networks (DNNs) are ANNs with multiple layers between the input and output layers. Deep learning refers to the use of DNNs. There exist many deep learning architectures, combining layers in specific ways tailored to the task they aim to solve. Some important ones are:

- **Dense Neural Networks** composed entirely of fully connected layers after the input.
- **ConvNets** composed of one or more convolutional layers after the input, followed by one or more fully connected layers.
- **AllConv** [36] composed entirely of convolutional layers after the input.
- **Recurrent Neural Networks** (RNNs) composed of recurrent layers only (fully recurrent) or making use of recurrent layers.

[4] See [23] to have an explanation of gradient descent.

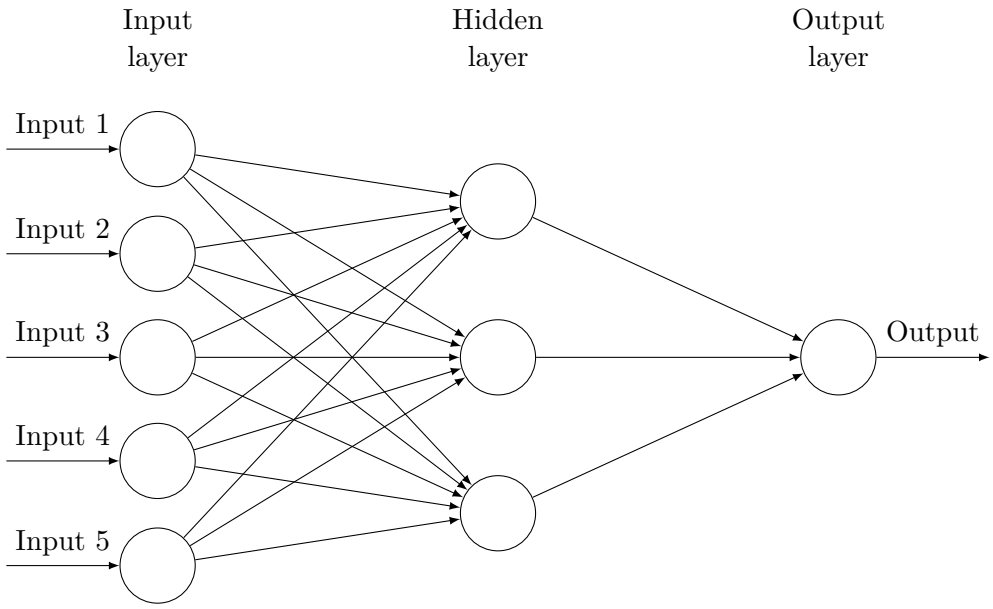

Figure 2.16: Diagram of an Artificial Neural Network. [21]

2.4 Conclusion

Various signal transforms have been introduced which are used to pre-process the input waveforms in most of the state of the art approaches, as described in the next chapter. These transforms are also used for our models in chapter 5 and 6. The HMM framework on the other hand will be used as post-processing for NMF and PLCA as described at the end of section 3.1. Finally, the very developed DNN framework has been superficially described, enough to understand the work done in the thesis. Some great courses on the internet can help the curious readers to learn more about it, such as [35]. All the basic technical background needed for this thesis has now been introduced.

Chapter 3

State of the Art algorithms for APT

Some of the first research on AMT was done in 1977 by J. Moorer for duet with strong restrictions to avoid harmonic ambiguities [37]. Since then, algorithms have evolved, using more and more computing power and complex models. A decade ago, the most promising algorithms were signal processing ones, using Non-negative Matrix Factorisation (NMF) or Probabilistic Latent Component Analysis (PLCA). However, since the rise of machine learning, many neural network architectures have been studied, even though the available labelled datasets are not as abundant as we would like for these approaches. The next chapters of the thesis will only focus on the machine learning algorithms introduced in section 3.3.

3.1 Non-negative Matrix Factorisation

The goal of Non-negative Matrix Factorisation (NMF) [3] is to summarise an audio spectrum by decomposition into a spectral profile of the notes and the temporal information. We do so by reducing the dimensionality of the input, using matrix factorisation.

We start with a non-negative matrix $\mathbf{X} \in \mathbb{R}^{+,\Omega \times T}$ which is a spectro-temporal representation of a music of length T over Ω different frequencies, e.g. a CQT of its temporal signal. The goal is to decompose it into a product of two matrices $\mathbf{W} \in \mathbb{R}^{+,\Omega \times R}$ and $\mathbf{H} \in \mathbb{R}^{+,R \times T}$, such that $\mathbf{X} \approx \mathbf{W} \cdot \mathbf{H}$. $\mathbf{H}$ will then represent the activity of each spectral profile over time, and $\mathbf{W}$ the spectral profiles, as shown in figure 3.1.

To do so, we define a cost function C to minimise, such as

$$C = \|\mathbf{X} - \mathbf{W} \cdot \mathbf{H}\|_F \tag{3.1}$$

with F the Frobenius norm. The minimisation is then done either by computing derivatives of $\mathbf{H}$ and $\mathbf{W}$:

$$\Delta\mathbf{H} \propto (\mathbf{W} \cdot \mathbf{H} - \mathbf{X}) \cdot \mathbf{H}^\top \text{ and } \Delta\mathbf{W} \propto \mathbf{W}^\top \cdot (\mathbf{W} \cdot \mathbf{H} - \mathbf{X})$$

and updating their respective values in that direction iteratively, or by using the algorithm introduced by Lee and Seung [38], with the following update rules:

$$\mathbf{H}_{at} \leftarrow \mathbf{H}_{at} \frac{\sum_i \mathbf{W}_{ia}\mathbf{X}_{it}/(\mathbf{W} \cdot \mathbf{H})_{it}}{\sum_k \mathbf{W}_{ka}} \tag{3.2}$$

$$\mathbf{W}_{ia} \leftarrow \mathbf{W}_{ia} \frac{\sum_t \mathbf{H}_{at}\mathbf{X}_{it}/(\mathbf{W} \cdot \mathbf{H})_{it}}{\sum_v \mathbf{H}_{av}} \tag{3.3}$$

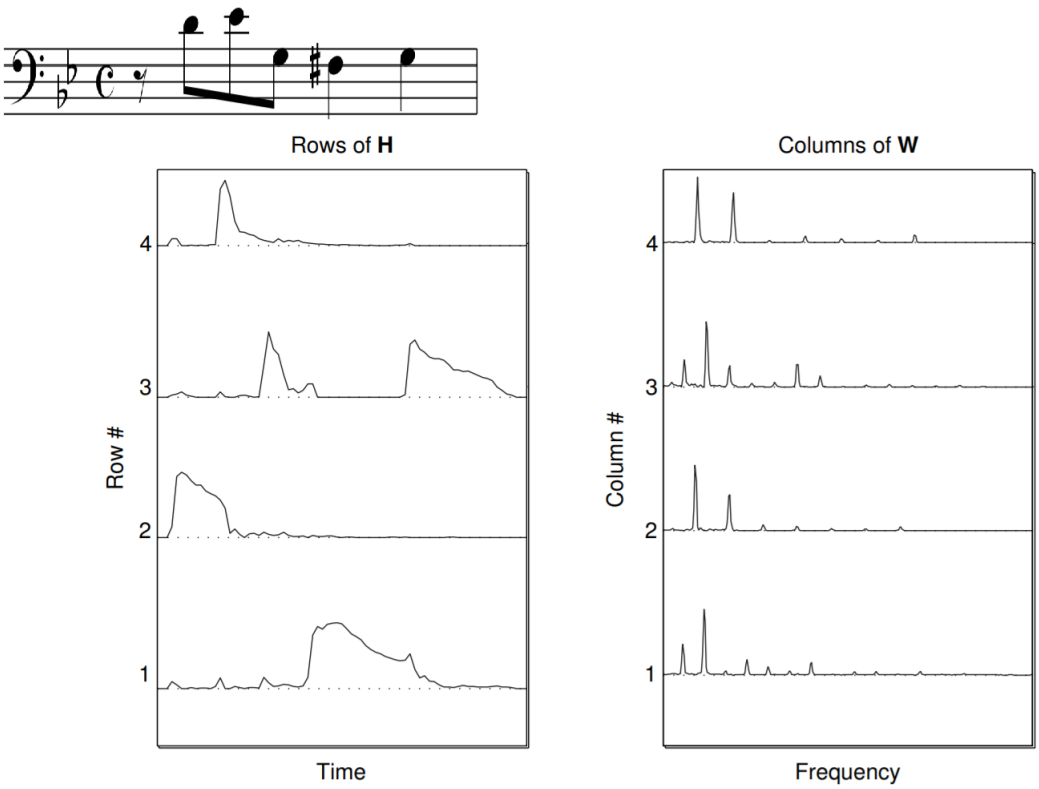

Figure 3.1: Illustration of the decomposition into **H** and **W**. [3]

The level of summarisation is determined by R, the rank of **W**. Having a smaller R forces the decomposition to extract the major elements of the input, whereas having $R = M$ for example would not be very useful as the decomposition would be exact and not very informative.

From the **W** and **H** matrices, we can then do the transcription. If a suitable factorisation was found, analysing the columns of **W** reveals which note is represented by each atom, and which atoms do not represent notes at all. Once this labelling step is done, the activity matrix **H** has to be transformed into a binary matrix - a note is either there or not there, there is no in-between. A simple hard threshold could do the trick, but the best results are obtained through a HMM post-processing.

A simple HMM post-processing introduced in section 3.3.3 of [11] is made of 88 independent HMM models, one per pitch, with two states: active or inactive. The initial state and state transition probability distributions can be estimated from databases containing pieces of music in the MIDI format. The observation symbol probability distribution is a function of the activity value (from the activity matrix). HMMs let us model the hidden states from the observed activity matrix, in order to smooth the predictions and filter out noise, as seen in figure 3.2. This post-processing can be used in other approaches, such as PLCA in the next section.

3.2 Probabilistic Latent Component Analysis

Probabilistic Latent Component Analysis (PLCA) can be seen as a probabilistic extension of NMF [11]. It gives an interpretable framework, which is easily generalisable to take multiple instruments or time-variable spectral templates into account, and much more.

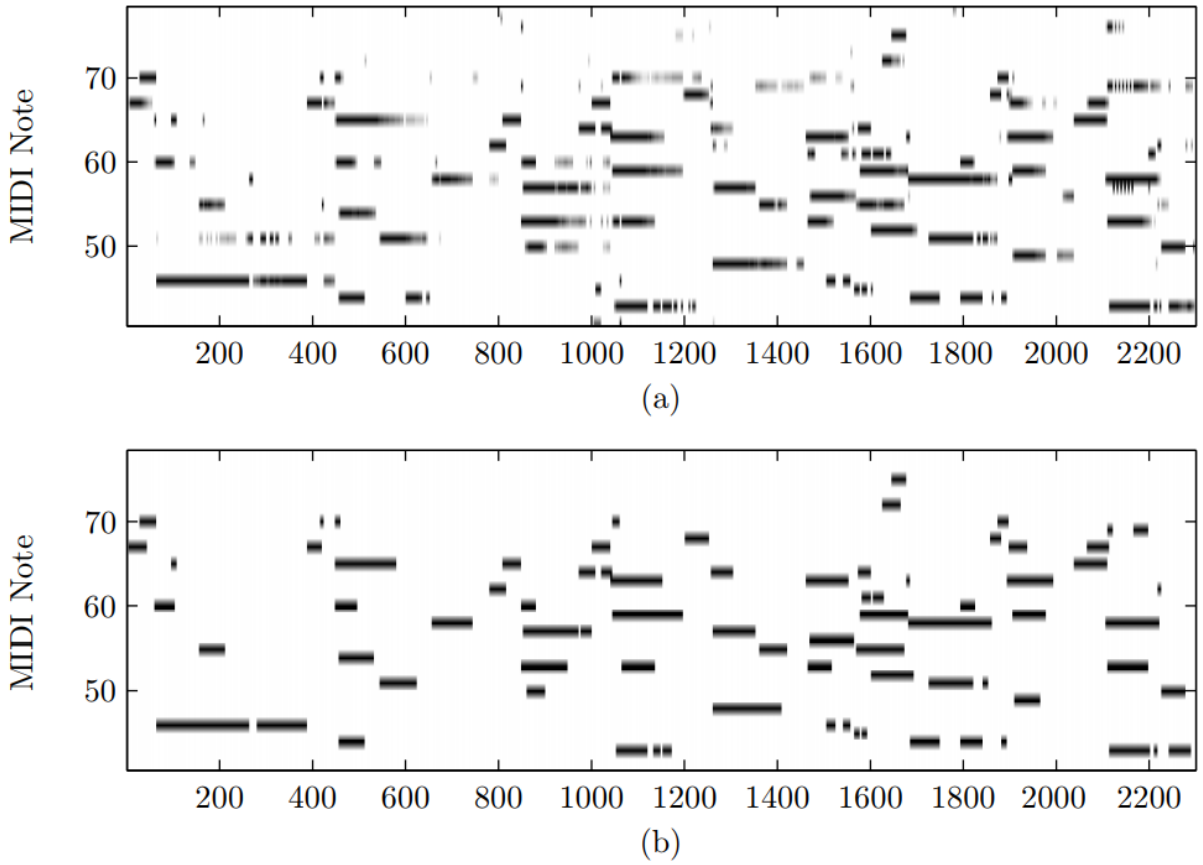

Figure 3.2: Transcription output of an excerpt of 'RWC MDB-J-2001 No. 2' (jazz piano) in a 10 ms time scale (a) Output of the multiple-F0 estimation system (b) Piano-roll transcription after HMM postprocessing. [11]

3.2.1 Basic Principle

It uses as input the signal $\mathbf{V} \in \mathbb{N}^{T \times \Omega}$, which is a spectro-temporal representation of a music of length T over Ω different frequencies, e.g. a CQT of its temporal signal. Each $\mathbf{V}_t$ is considered as being the result of the draw of i.i.d. variables over frequencies according to the distribution F_t. In the most basic approach, F_t is modelled as a weighted sum of a fixed number of atomic spectral templates, that is

$$F_t(\omega) = \sum_z P(\omega|z) A_t(z) \tag{3.4}$$

with z the atom index, $P(\omega|z)$ the spectral template of the z^{th} atom and $A_t(z)$ the activation probability of the z^{th} atom at time t. This model is very similar to the NMF one.

The goal is to find the F_t's (for every t) which maximise the probability to draw $\mathbf{V}$. To do so, the unknown $P(\omega|z)$ and $A_t(z)$ are approximated using the Expectation-Maximisation (EM) algorithm. The expectation is computed using the Bayes' theorem:

$$P_t(z|\omega) = \frac{P(\omega|z) A_t(z)}{\sum_{z'} P(\omega|z') A_t(z')} \tag{3.5}$$

and the maximisation can be computed as

$$P(\omega|z) = \frac{\sum_t \mathbf{V}_{t,\omega} P_t(z|\omega)}{\sum_{t,\omega} \mathbf{V}_{t,\omega} P_t(z|\omega)} \tag{3.6}$$

$$A_t(z) = \frac{\sum_\omega \mathbf{V}_{t,\omega} P_t(z|\omega)}{\sum_{z,\omega} \mathbf{V}_{t,\omega} P_t(z|\omega)}. \tag{3.7}$$

Iterating between 3.5 and 3.6-3.7 is guaranteed to converge to some solution. [4]

3.2.2 Variations

Many variations of this framework exist, some of which are detailed in [11]. We will have a brief overview of some of these.

Shift-Invariant PLCA

When using a logarithmically scaled frequency axis in the spectro-temporal representation of a piece of music, shifting the frequency upwards is essentially the same as playing a higher note (assuming the timbre of the instrument is roughly the same over all frequencies). Introducing this invariant can improve the model by detecting more general structures. The Shift-Invariant PLCA (SI-PLCA) model is defined as

$$P_t(\omega) = \sum_z P(z) \sum_f P(\omega - f|z) A_t(f|z) \tag{3.8}$$

with ω the log-frequency index, z the atom index, and f the shifting factor. We find again the spectral template of each atom in $P(\omega - f|z) = P(\mu|z)$ and activation of these atoms in $P(z)A_t(f|z)$, $P(z)$ being the prior probability of the atom. Even with only one atom, this model can already produce a first draft of transcription, as shown in figure 3.3, whereas the basic PLCA model needs at least one atom per pitch.

HMM-constrained SI-PLCA

As not all frames are independent of each other, particularly from its neighbouring frames, and introducing some memory into the equation can result in some significant improvements. In [11], the SI-PLCA is combined to the HMM post-processing in an HMM-constrained SI-PLCA monophonic model. It uses states with their own spectral template, which can represent for example which note is played and/or which part of the note it is (attack, decay, and so on), in place of the previous atoms. The model is

$$P(\overline{\omega}) = \sum_{\overline{q}} \left(P(q_1) \prod_t P(q_{t+1}|q_t) \right) \left(\prod_t P(\overline{\omega}_t|q_t) \right) \tag{3.9}$$

with $\overline{\omega}$ a draw sequence for all time frames such as $\mathbf{V}$, $\overline{q}$ a state sequence, $P(q_1) \prod_t P(q_{t+1}|q_t)$ the probability of the state sequence (with prior probability $P(q_1)$ and transition probability $P(q_{t+1}|q_t)$), $\overline{\omega}_t$ a sequence of draw at time t and $P(\overline{\omega}_t|q_t)$ the observation probability given the state.

This last probability can be expressed as

$$P(\overline{\omega}_t|q_t) = \prod_{\omega_t} P_t(\omega_t|q_t)^{\mathbf{V}_{t,\omega}} \tag{3.10}$$

$$P_t(\omega_t|q_t) = \sum_{f_t} P(\omega_t - f_t|q_t) P_t(f_t|q_t) \tag{3.11}$$

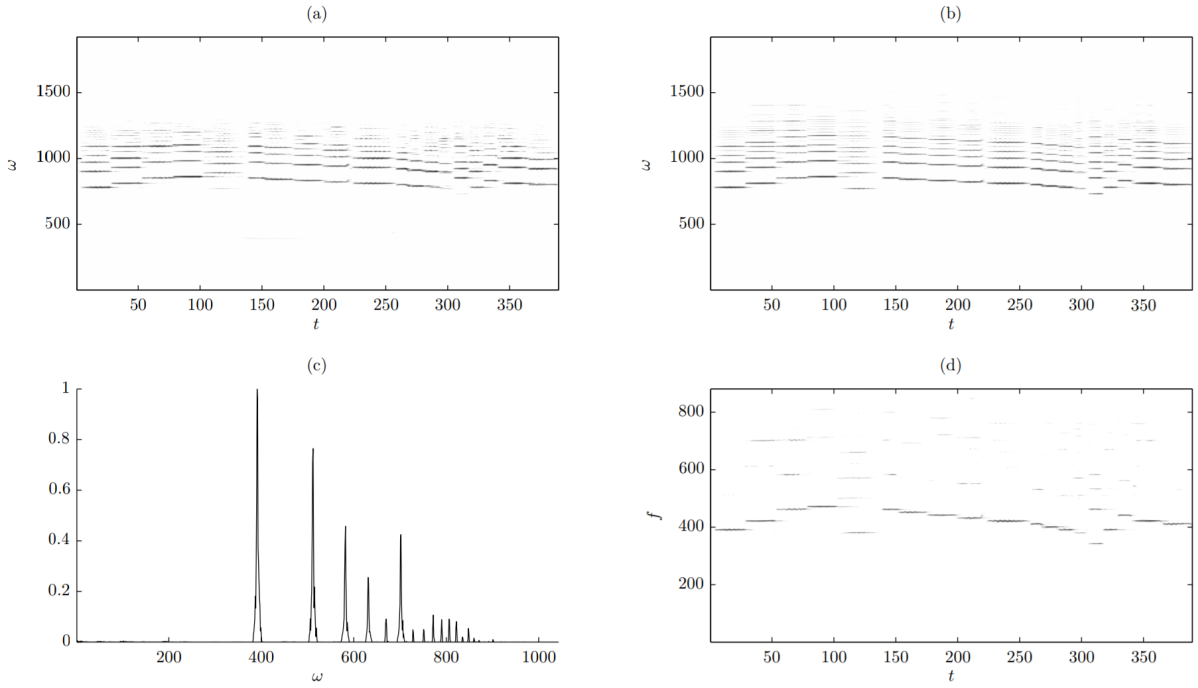

Figure 3.3: (a) The log-frequency spectrogram $P_t(\omega)$ of a cello melody (RWC-MDB-C-2001 No. 12 [39]) (b) Approximation of the spectrogram using SI-PLCA with z = 1 (c) The spectral template $P(\omega|z)$ (d) The pitch distribution $A_t(f|z)$. [11]

with $P(\omega - f|q) = P(\mu|q)$ the spectral template of each state and $P_t(f|q)$ the time-dependant pitch shifting factor for each state ($f \in [1, F]$).

After estimating the parameters through a complicated EM procedure, the activation of each sound state is given by

$$P_t(q_t|\overline{\omega}) \sum_{\omega} \mathbf{V}_{t,\omega} \tag{3.12}$$

This model is a monophonic model, but it can be further extended to take into account multiple instruments and multiple pitches per instrument. This is done by using 88 independent HMM models, one per possible piano pitch and by doing some modifications in the formulas. All the details are in chapter 4 of [11].

3.3 Machine Learning

More recently, the rise of machine learning has interested many AMT researchers. Most of the recent state of the art algorithms use some form of deep learning, some using RNN combined with Music Language Models (MLMs) [40], other exploring also the traditional ConvNets [41]. Two of the most promising architectures have caught our attention: *Anthem Score* [2], which directly produces sheet music instead of piano rolls, and *Onsets and Frames* [1] which extends the *Kelz baseline architecture* introduced in section 3.3.3. Our work is mainly based on this last architecture.

3.3.1 Datasets

A necessary ingredient for supervised machine learning is a huge set of labelled data. To produce these, one can synthesise some MIDI data with complex digital piano synthesisers. However, synthesising piano is a difficult task and the result is not as representative of physical performances

as we could want it. Many datasets for MIR are listed in [42], few of which are useful for the particular task of APT.

Two of the most known datasets are Bach10 [43] and **MAPS** [5] (MIDI Aligned Piano Sounds). The first one is composed of ten pieces of four-part J.S. Bach chorales with one track per instrument as well as the corresponding MIDI and ground-truth alignment. It is useful for AMT with multiple instruments, as it uses violin, clarinet, saxophone and bassoon, but not for APT. The second one is piano only, with some virtual piano software and a Yamaha Disklavier. It contains isolated notes, random and usual chords, as well as complete piano pieces with CD-quality. This last part is randomly taken from 238 different classical and traditional pieces at a rate of 30 per set of conditions (see table 3.1). Everything was automated to produce this dataset by audio synthesis from MIDI files, to have the most reliable ground-truth possible. However, it contains errors, as we will see in section 4.2.

Code	Instrument model	Recording conditions	Real instrument or software
AkPnBcht	Bechstein D 280	concert hall	Akoustik Piano (Native Instruments)
AkPnBsdf	Boesendorfer 290 Imperial	church	Akoustik Piano (Native Instruments)
AkPnCGdD	Concert Grand D	studio	Akoustik Piano (Native Instruments)
AkPnStgb	Steingraeber 130 (upright)	jazz club	Akoustik Piano (Native Instruments)
ENSTDkAm	Yamaha Disklavier Mark III (upright)	“Ambient”	Real piano (Disklavier)
ENSTDkCl	Yamaha Disklavier Mark III (upright)	“Close”	Real piano (Disklavier)
SptkBGAm	Steinway D	“Ambient”	The Black Grand (Sampletekk)
SptkBGCl	Steinway D	“Close”	The Black Grand (Sampletekk)
StbgTGd2	Hybrid	Software default	The Grand 2 (Steinberg)

Table 3.1: MAPS: instruments and recording conditions. [18]

3.3.2 Anthem Score architecture

Anthem Score [2] is a commercial software which attempts to produce a complete transcription from raw audio to playable sheet music. It works best with piano, but can transcribe other instruments too, even if it cannot recognise these and will put the notes of all instruments indistinctly on the same staves. As it is specialised for piano, it separates the sheet music in pitches higher or equal to C4 (Treble clef) and lower to C4 (bass clef). The algorithm to recognise notes focuses on finding the onsets, assuming each note will last until the next one (in the same staff) is played. This is generally not true for all notes in a piece of music but produces readable sheets which are enough for the player to guess the true length of each note at play time. A transcription example is shown in figure 3.4, where we can see that the treble/bass separation hypothesis is too strict to follow the music logic and results in incorrect note lengths. Some experimentation also showed that the transcription is very bad for very high pitches, missing most of these. However, the transcription is overall very acceptable and the mistakes are easily corrected manually thanks to some of the software features.

From a technical point of view, Anthem Score uses deep learning over small time ranges of a CQT-like representation of the music to do the predictions. The CQT is modified to have a better frequency resolution at the cost of a lower time resolution in some critical points, and has its final values passed through a non-linear function to have a final representation closer to the

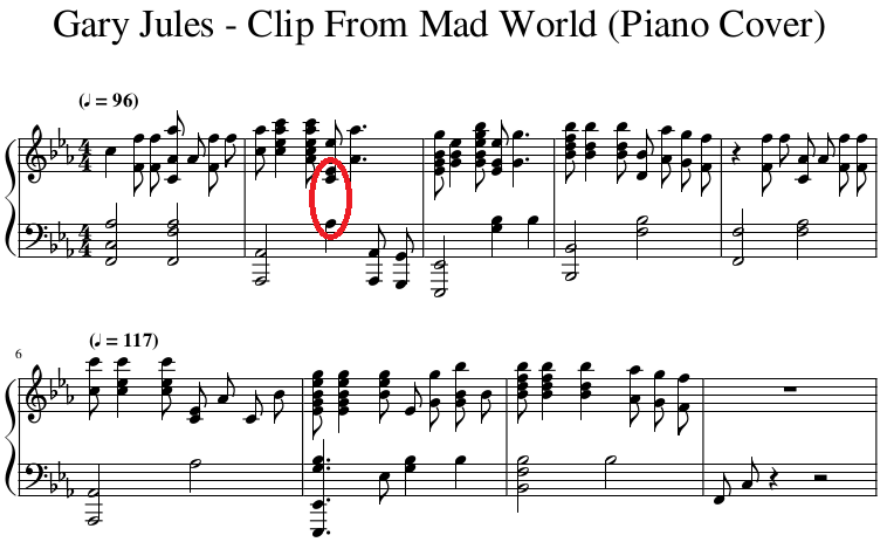

Figure 3.4: Anthem Score transcription. [2]

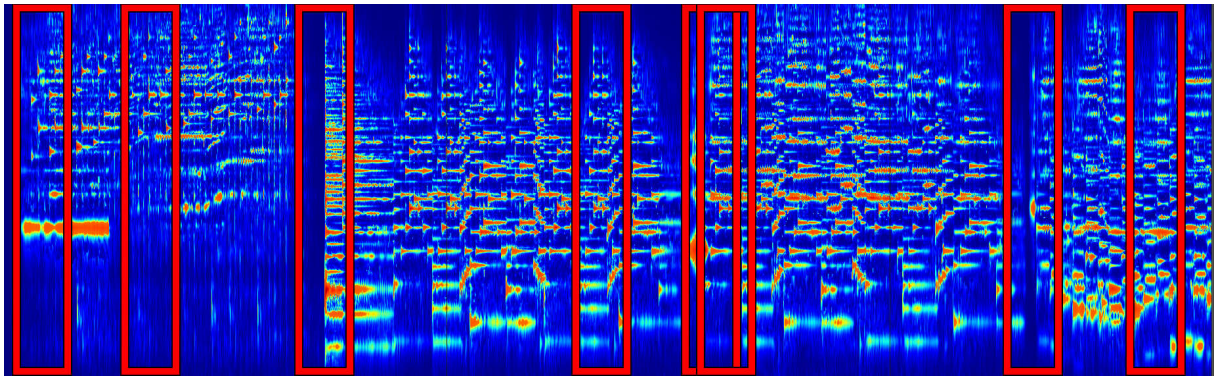

Figure 3.5: Anthem input example (red rectangles). [2]

log of the amplitude. It computes four bins per pitch, seemingly from 26.7 Hz (a little less than A0) to 4308.7 Hz (a little more than C8) as shown by the Graphical User Interface (GUI) of the application.

Then, using this spectrogram, it detects potential onset positions by searching local maxima in the mean (over all frequencies) amplitude change. The goal of this onset detection is to avoid missing any onset (even if that means detecting too many) to make the rest of the algorithm faster without losing any note. If a detected onset is not really an onset, the end of the algorithm will not associate it to any note and it will thus not be seen as a real onset.

Finally, for each time detected as a possible onset position, the whole frequency range and a small time range starting before and ending after this time (cf. figure 3.5) is passed as input to a DNN to predict which notes start at this time. The rest of the algorithm to detect the tempo, the key signature and so on is not publicly available and is out of the scope of this thesis.

The exact structure of the DNN is not publicly available either, but the gist of it is published on their website. It is a residual neural network made only of convolutional layers, pooling layers and addition layers. A part of it can be seen on figure 3.6. Most of the convolution pairs are long and skinny ($M \times 1$ followed by $1 \times N$ convolutions), the goal being to fetch the harmonics, distant from the fundamental. In the end, despite not having any fully connected layer, every output neuron is influenced by every input bin. The skip connections transform the architecture into a sum of (possibly negative) features, which makes sense in music since the expected output is essentially the spectral input without its non-fundamental harmonics.

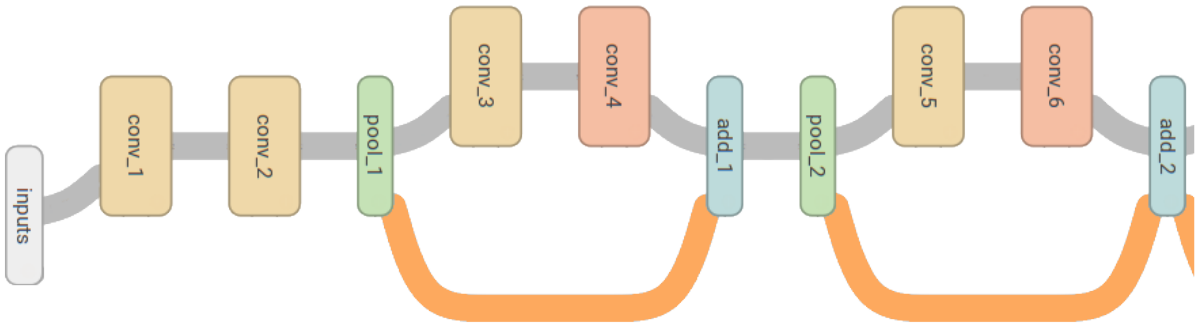

Figure 3.6: Part of the Anthem architecture, with skip connections in orange. [2]

3.3.3 Kelz architecture

A thorough analysis of simple bottom-up frame-wise processing in deep learning is detailed in [44], analysing the performances of dense neural networks, AllConvs and ConvNets for different input representations and hyperparameter tuning. The conclusion is that ConvNets are the best for the APT task and can be used as a baseline for other approaches. We will call this architecture the *Kelz (baseline) architecture* from now on. This architecture is used in the Onsets and Frames model presented in section 3.3.4.

The architecture is composed of convolutional layers directly after the input, followed by a fully connected layer. For readability purpose, this architecture will be presented as an ordered list. Mentioned shapes are of the form $time\ dimension \times frequency\ dimension$.

- the input 7x229 (the paper mentions 5x229, but we need seven frames since we have three convolution layers with a width of 3)
- two convolutional layers (32 3x3 filters each)
- a batch normalisation layer
- a max pooling layer (1x2 pooling)
- a dropout layer with 0.25 keep rate
- one convolutional layer (64 3x3 filters)
- a max pooling layer (1x2 pooling)
- a dropout layer with 0.25 keep rate
- a fully connected layer with 512 neurons
- a dropout layer with 0.5 keep rate
- a fully connected layer with 88 neurons (removed when followed by other layers)

3.3.4 Onsets and Frames architecture

In Onsets and Frames [1], a model was built with the goal of being able to generalise to unseen instruments. The approach uses the Kelz architecture and LSTMs both to predict the onsets, and to condition the frame predictions on these onset predictions. This results in a transcription that correlates better with human musical perception, rather than predicting either separately.

Model

An onset detector was trained, and its raw output was used as additional input for the frame predictor. Moreover, a thresholded version of this output is used during the inference, so that a frame is predicted only if the onset detector agrees.

For both parts of the architecture, as shown in figure 3.7, the input is a MEL spectrogram. Also, instead of having only a few frames as input, the whole sequence is input. This way, the output of the convolution layers can be used as input for the LSTM.

The onset detector (right column in figure 3.7) is based on an acoustic model, which is composed of the Kelz architecture (section 3.3.3), followed by a BiLSTM layer. Finally, there is a fully connected layer with sigmoid activation, representing the probability of activation for each of the 88 possible piano keys.

The frame detector also uses the Kelz architecture, but is followed directly by a fully connected layer (88 neurons) with sigmoid activation. The output of the fully connected layer is then concatenated with the output of the onset detector, and goes through a BiLSTM layer, followed by another fully connected layer (88 neurons) with sigmoid activation.

For training, a pre-processing of the dataset is needed since training LSTMs over long sequences can be very memory intensive. The input audio files are thus split into smaller files. To minimise note splitting between the files, 20 seconds splits were deemed appropriate. The split point is the maximum time below 20 seconds where no note is played, when starting from the last split point. When active notes must be split, a zero-crossing of the audio signal is chosen. Finally, the loss function is the sum of the onset detector loss and the frame detector loss.

$$L_{total} = L_{onset} + L_{frame} \tag{3.13}$$

with

$$L_{onset}(l,p) = \sum_i -l(i)\log p(i) - (1-l(i))\log(1-p(i)) \tag{3.14}$$

where $l(i)$ represents the presence of the onset of note i and $p(i)$ represents the predicted probability of said onset. The frame loss is weighted in order to ensure an accurate onset. Each note has three frames associated to specific events, namely: t_1 for the note start, t_2 for the end of the onset and t_3 for the end of the note. The raw frame loss is defined as follows:

$$L'_{frame}(l,p) = \sum_i -l(i)\log p(i) - (1-l(i))\log(1-p(i)) \tag{3.15}$$

where $l(i)$ represents the presence of a frame i and $p(i)$ represents the predicted probability of said frame. This enables us to define our frame loss as:

$$L_{frame}(l,p) = \begin{cases} cL'_{frame}(l,p) & when \quad t_1 \leq t \leq t_2 \\ \frac{c}{t-t_2}L'_{frame}(l,p) & when \quad t_2 < t \leq t_3 \\ L'_{frame}(l,p) & elsewhere \end{cases} \tag{3.16}$$

with c being a hyperparameter, empirically chosen to be 5.

Both are thus computed as the weighted sum of cross-entropy between the prediction and the ground-truth, with log function being capped at $\ln(10^{-7}) \approx -16$ to avoid exploding loss. During the inference, a hard threshold of 0.5 is used for both detectors, in order to determine the activation.

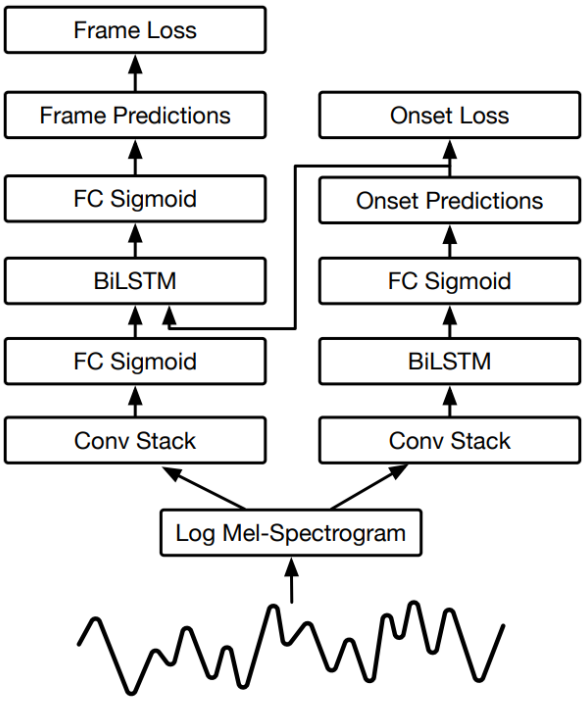

Figure 3.7: Architecture of the model used in Onsets and Frames. [1]

3.4 Conclusion

The field of AMT and more particularly of APT has already been explored in various ways for many years, from basic signal processing and matrix factorisation techniques to HMM post-processing and machine learning approaches, but the performances are not as good as those of human experts yet. The lack of big labelled databases is a huge drawback for this task, particularly for the machine learning approaches which have known their success from big datasets. Nonetheless, these latter obtain the best results amongst other techniques, and more particularly the Onsets and Frames approach which this thesis will analyse and try to improve.

Part II

Thesis contributions to the Automatic Piano Transcription task

Chapter 4

Dataset analysis

As introduced in the previous chapter, Onsets and Frames [1] is a state of the art APT algorithm using deep learning. This thesis will focus on analysing and improving this particular algorithm through small modifications. The dataset used by Onsets and Frames to train and test its model is the MAPS dataset [5], more particularly the part composed of complete piano pieces. It has been introduced in section 3.3.1. From now on, the terms *MAPS dataset* will refer only to this particular part of it, unless stated otherwise. This chapter will analyse its content more in details.

4.1 Content of the dataset

The MAPS dataset is composed of nine parts, each being a particular piano setting (see table 3.1). These parts are each composed of 30 piano performances of pieces randomly taken from 238 different classical and traditional ones (160 of which are finally used). Therefore, some pieces are played only by a single piano while others are played by many. Onsets and Frames splits the dataset in a test set composed of the 60 performances of the real pianos (ENSTDk's in table 3.1) and a train set with all the other performances which are not already performed in the test set. In other words, a piece played by a digital piano is unused if the same piece is played by a real piano. In the end, the train set is as described in table 4.1.

Piano	Pieces
AkPnBcht	23
AkPnBsdf	20
AkPnCGdD	21
AkPnStgb	25
SptkBGAm	25
SptkBGCl	25
StbgTGd2	0
total	139 (from 107 distinct pieces)

Table 4.1: Onsets and Frames train set composition.

The StbgTGd2 has exactly the same pieces played as the ENSTDkCl, which seems to be an error in the randomisation of MAPS. Therefore, the StbgTGd2 part is completely unused by Onsets and Frames. To have comparable results, we used the same splitting for our models.

For each piano performance, there is a wav file in CD-quality (reduced to single-track 16000Hz sampling by Onsets and Frames), the piano performance, as well as a MIDI file, the ground truth. This MIDI file contains the onset and offset times of each note, corresponding to pressed keys on the piano, as well as a sustain pedal indication. Therefore, in practice, the notes can last for longer than indicated, stopping only when the sustain pedal is released. Onsets and Frames modifies each offset to take this into account, setting it at the time the sustain pedal is released

instead of the time the key is released when applicable. However, when the note is supposed to be played for too long, it could stop being perceptible before it theoretically ends. This is a major problem to solve APT, as the ground truth can be different for two identical frames, depending on its context. Thankfully, deep learning solutions exist to deal with this problem, such as the LSTM layers used by the Onsets and Frames algorithm, as they can take the context into account and could, for example, implicitly detect the sustain pedal.

4.2 Ground truth errors

The process to build the MAPS dataset was highly automated to avoid mistakes, but some remain, particularly in the real piano performances, as reported in [1]. This can be a huge problem for learning algorithms as they will try to learn something wrong, possibly preventing them from generalising well and converging fast. As Onsets and Frames chose not to train on the real piano files, this is not a problem for the learning part, but the evaluation part will generally underestimate the performances of the model as missed notes will be wrongly reported.

4.3 Pitch distribution

As the MAPS dataset is composed of traditional and classical pieces, each pitch is not equally present. Using the Onsets and Frames loss weighting (equation 3.16) to compute the presence of each pitch in the Onsets and Frames pre-processed train set and in its test set, we obtain the plot shown in figure 4.1. As we can see, the training pitch distribution is close to a Gaussian, with a mean of 63.89 ($\approx E4$) and a standard deviation of 13.20 ($\approx$ an octave). The test distribution is quite similar too, as expected. The spiky look of the plot is due to the natural notes being more present than the accidental ones (except for the $F\sharp$ and $B\flat$ which are the most common accidentals in traditional and classical music). Some pitches are not present at all, both in the train and in the test set. Chapter 6 will analyse the effect of balancing this distribution for the training phase.

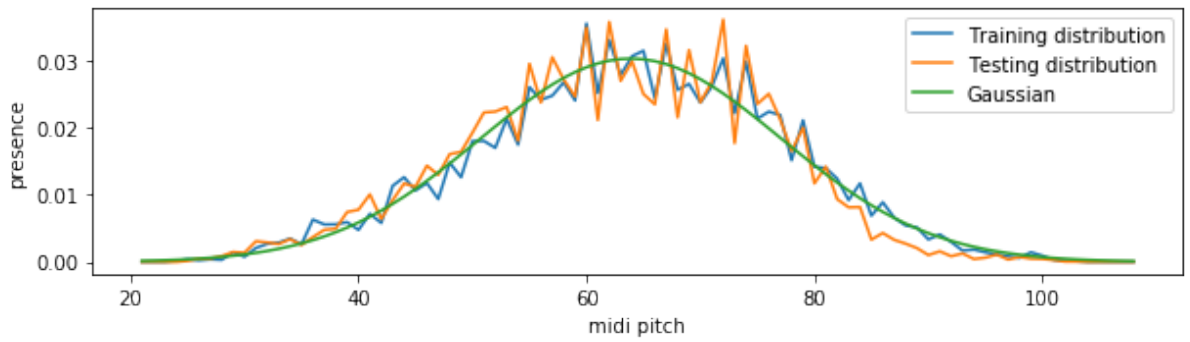

Figure 4.1: Pitch distribution in the Onsets and Frames train and test set.

4.4 Shifted harmonics

Using the "isolated notes" part of the complete MAPS dataset, we analysed the spectral template of different notes from the different pianos. For the highest note of the dataset (figure 4.2), we can observe that the pitch is half a semitone higher than its theoretical value for most of the pianos. It is even worse when looking at the second harmonic, where the ENSTDkCl piano one (brown) is spread over the whole semitone while the AkPnStgb one has only a very little activation an entire semitone higher than expected. The worst part of it is that the shift is different for each

piano (but similar for the same pianos in different conditions, such as ENSTDkAm-ENSTDkCl and SptkBGAm-SptkBGCl). This shift is due to the non-null diameters of the piano strings which modify their harmonic frequencies. The pitch of higher notes is, therefore, tuned to sound well with these wrong harmonics, propagating the error up to the highest and down to the lowest pitches.

Thankfully, the phenomenon is less extreme for middle pitches, which are the most used. For as little as an octave below, the practical frequencies are almost the theoretical ones, as shown in figure 4.3. This phenomenon has been studied and is explained in detail in [45]. This shift could be taken into account to define a proper APT model, as lightly developed in section 5.3.3.

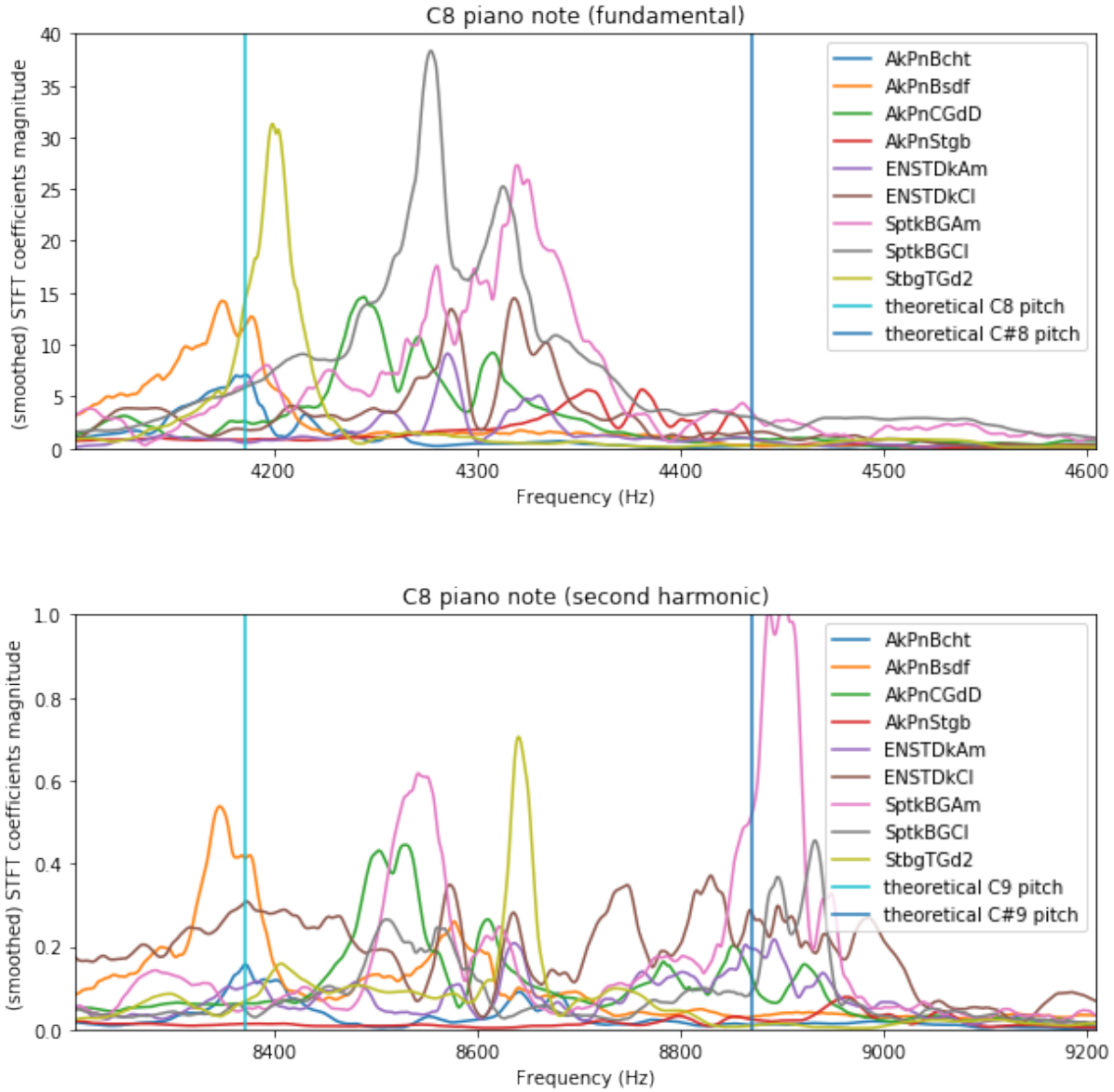

Figure 4.2: C8 piano note spectral template from the MAPS dataset (MAPS_ISOL_NO_F_S*_M108_*.wav).

4.5 Conclusion

In this chapter, we explored the MAPS dataset in order to get more familiar with the data used in this thesis.

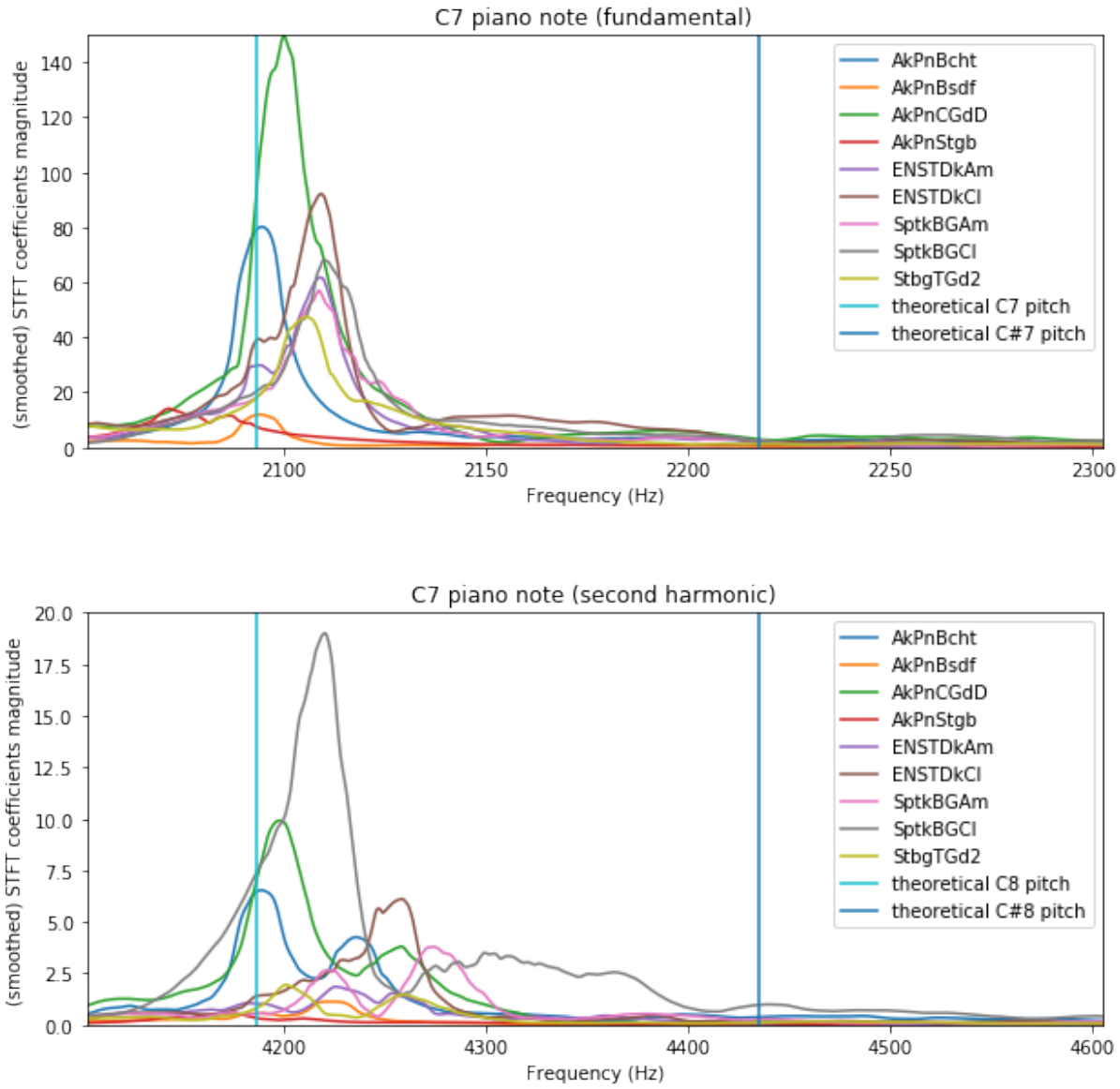

Figure 4.3: C7 piano note spectral template from the MAPS dataset (MAPS_ISOL_NO_F_S*_M96_*.wav).

We identified several problems in the dataset, making the transcription task more difficult. Some of these are due to the very nature of the problem to be solved, namely, the notes' sounding stopping before the theoretical end for very long notes, the unbalanced pitch distribution and the shifted harmonics. Others are real errors which should be corrected in the dataset, namely, wrong randomisation of the pieces played by the StbgTGd2 piano and missed low velocity notes on the real pianos.

While some of these are already addressed by the original Onsets and Frames algorithm, others are still to be explored, particularly the pitch distribution explored in chapter 6.

Chapter 5

Modifying the Onsets and Frames architecture

In this chapter, we test some small changes in the Onsets and Frames [1] architecture (see section 3.3.4). The goal of these modifications goes from analysis of the original model to acceleration and improvement of its performances for the APT task. Due to the insufficient computing power to perform hyperparameter tuning for each model, the hyperparameters of the tested models are set to the values in the code mentioned in section 5.1, unless stated otherwise. This limitation could have a significant impact on the performances.

This chapter will describe the testing conditions, two of the models we tested as well as their performances and analysis.

5.1 Testing conditions

All the models have been trained on the jabba server of the Computing Science Engineering Department, using an average of approximately 20 CPUs under ubuntu 16.04. The code used to estimate the original model performances and modified for the tested models is the version c9962ca of Magenta on Github[1], using the Bazel and TensorFlow CPU installation. It is important to note that the weighting of the loss (equation 3.16) is (erroneously) deactivated in this version. This error has been spotted too late to recompute all the results of this chapter, but these are presumably representative enough to draw our conclusions.

For technical reasons, the training may have been interrupted regularly (every 10-20 000 steps) and restarted soon after, without any important effect on it. The original Onsets and Frames model will be named the *Kelz* model as it uses the Kelz architecture for its Conv Stack. As for the dataset split into training and testing, we used the one described in section 4.1.

5.2 Metrics

In order to compare the different models proposed, we need to define a metric. In this thesis, we chose the F1 score (also called F-score or F-measure) to measure the performances of each model. The F1 score is a widely used measure of a classifier's performances and is thus suitable for measuring the performances in APT. It uses 4 basic sub-metrics, computed by comparing the prediction and the ground truth (in an element-wise comparison of frame-pitch matrices), namely:

- **TP** (True Positive), the number of positive predictions with a ground truth positive as well. The **True Positive Rate** is defined as the ratio between TP and the total number of positive labels in the ground truth. Therefore, it takes a value between 0 and 1.

[1] `https://github.com/tensorflow/magenta/tree/c9962ca9cf4b8e7125641e6af46279c85614d0af`

- **TN** (True Negative), the number of negative predictions with a ground truth negative as well.

- **FP** (False Positive), the number of positive predictions with a negative ground truth. The **False Positive Rate** is defined as the ratio between FP and the total number of negative labels in the ground truth. Therefore, it takes a value between 0 and 1.

- **FN** (False Negative), the number of negative predictions with a positive ground truth.

The F1 score is a particular case of the more general F_β score (where $\beta = 1$). The F_β score is defined as

$$F_\beta \text{ score} = (1 + \beta^2) * \frac{P * R}{\beta^2 * P + R} \text{ with } \beta \in \mathbb{R}_0^+ \tag{5.1}$$

with P, the *precision*, defined as

$$P = \frac{TP}{TP + FP} \tag{5.2}$$

and R, the *recall*, defined as

$$R = \frac{TP}{TP + FN} \tag{5.3}$$

In this thesis, the F1 score is computed for the whole test set at once. An alternative would have been to compute a mean of the F1 scores of each of the test samples, as in the Onsets and Frames paper [1]. However, this alternative can lead to biased results, as suggested by [46]. We also measured the pitch-wise F1 scores of our models, comparing it with the original Kelz model in figures 5.3, 5.7 and 6.3. These graphs use the grey colour for the baseline (Kelz model) and the colours ranging from red to green for our models performances, depending on the delta with the baseline. A delta of less than -0.1 is in red while a delta of more than $+0.1$ is in green. The naming convention in the legend is *name_of_the_model number_of_training_iterations (total_F1_score)*.

Another interesting metric is the receiver operating characteristic (ROC) curve. It is based on the fact that most classification algorithms use a threshold to go from a continuous score to a binary classification as a final step. Changing the threshold can increase (resp. decrease) the True Positive Rate at the expense of an increased (resp. decreased) False Positive Rate. A ROC curve shows these rates for different thresholds. For the Onsets and Frames algorithm, two thresholds are defined, th_{frame} for the frames and th_{onset} for the onsets. We chose to vary these simultaneously to plot the ROC curves in figures 5.2, 5.6 and 6.2 , keeping $th_{frame} = th_{onset}$. A star on the curve indicates the result for intended model parameters, that is, $th_{frame} = th_{onset} = 0.5$. The naming convention in the legend of the ROC curves is *name_of_the_model number_of_training_iterations*.

5.3 The harmonic layer

As a note is essentially composed of a fundamental frequency and its harmonics, we designed a new type of layer: the *harmonic layer*. Where the traditional convolutional layer takes the neighbouring neurons' outputs for the convolution, the harmonic layer will take as input the output of neurons corresponding to its harmonics and to frequencies of which it could itself be a harmonic.

5.3.1 Definition

The conventional convolutional layer takes as inputs the $3 * 3 = 9$ outputs of the neurons corresponding to previous, current and next frames and frequencies, as introduced in section 2.3.5.

The harmonic layer tested in this section is a convolutional layer with $n_{filters} * (3 * 5 + 1)$ parameters to be learned, that is, 15 weights and 1 bias per filter. As a comparison, the classical 3×3 convolutional layer has $n_{filters} * ((3 * 3) + 1)$ parameters to be learned. This layer is theoretically 1.6 times slower at evaluation time than the classical one, given the number of parameters compared to the conventional layer. It takes as input the outputs of a neuron $\eta_{f,t}$, its two frame-neighbouring neurons $\eta_{f,t\pm1}$ and the neurons corresponding to a frequency two and three times bigger and smaller than the one corresponding to $\eta_{f,t}$. The input is therefore the outputs of neurons $\eta_{\{1/3,1/2,1,2,3\}*f,t+\{-1,0,1\}}$, as shown in figure 5.1. A zero-padding is applied for out-of-range neurons.

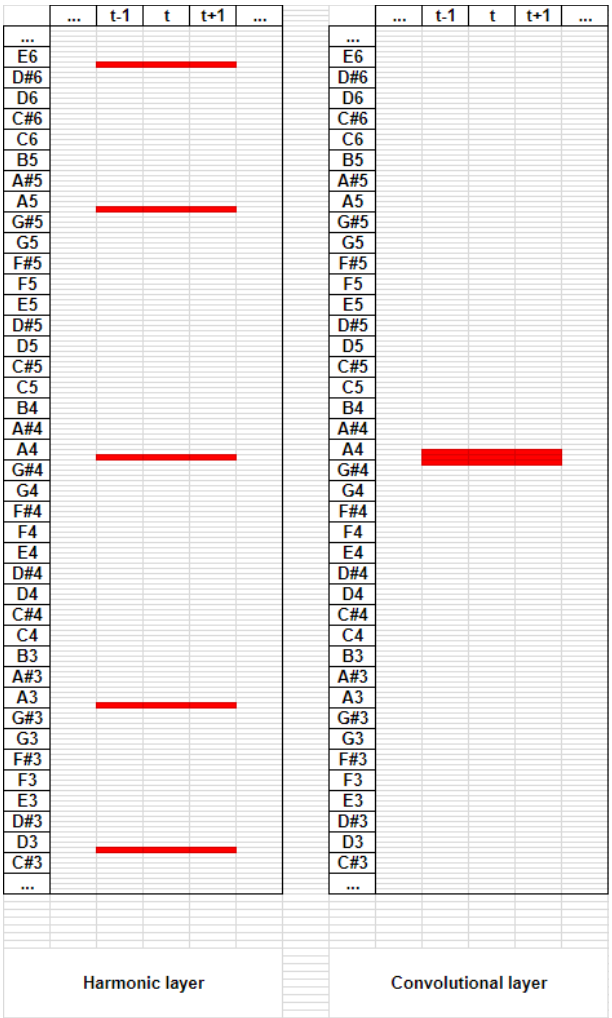

Figure 5.1: Used neurons for the harmonic and convolutional layers.

This layer could also be used as a pure frequential convolution, with as input the outputs of neurons $\eta_{\{1/7,1/5,1/3,1/2,1,2,3,5,7\}*f,t}$, the seventh harmonic having the interesting particularity that it falls between two traditional pitches ($33 < log\ _{\sqrt[12]{2}}(7) \approx 33.69 < 34$).

5.3.2 Results

To test this layer, we introduced it in the original Kelz model shown in figure 3.7. We replaced the two last convolutional layers of the Conv Stack by this one (both in frame and in onset

parts), but keeping the first convolutional layer a classical one. As this model mixes both types of convolutional layer, it has been named the *Hybrid* model. Other architectures have been tested, including one replacing every traditional convolutional layer by a harmonic layer, obtaining worse results. The input spectrogram of the Hybrid model is a CQT with 4 bins per pitch instead of the MEL spectrogram used for the original model.

As the replaced layers are the bottleneck of the model, the new architecture should be around 1.6 times slower for evaluation than the original one, which corresponds to our empirical estimation. The ROC curves are shown in figure 5.2, and the pitch-wise F1 scores are shown in figure 5.3. We observe that even after training for 90000 iterations, the original model performs slightly better the proposed model. However, it is worth mentioning that in the pitch-wise F1 scores comparison graph, we observe that the Hybrid model has better performances in the higher pitches than the original model.

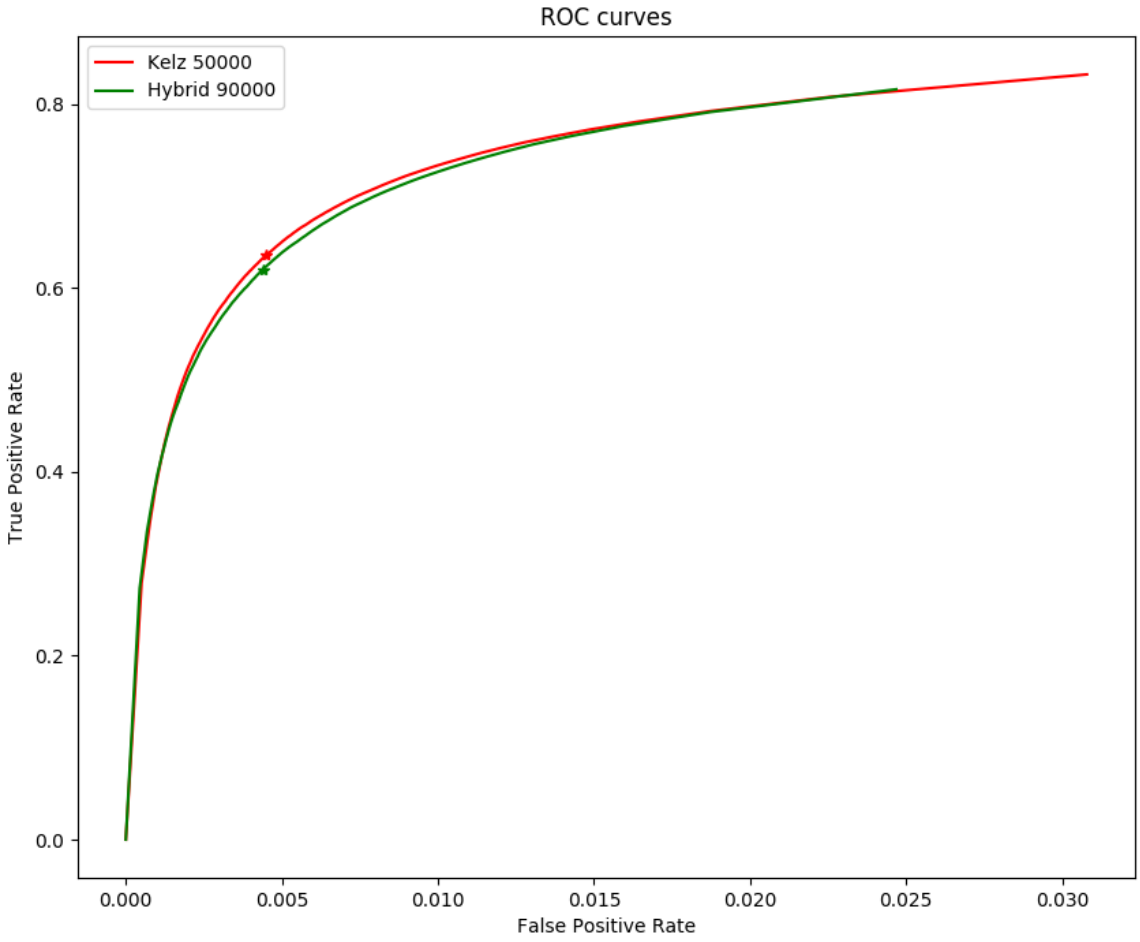

Figure 5.2: ROC curves of the original Kelz model and the Hybrid model proposed.

5.3.3 Analysis

Two reasons have been found for this result. The first one is the harmonic shifting explained in section 4.4, which makes our chosen bins the wrong ones for some of the pitches. As we cannot know in advance what will be the exact shift of the harmonics, we should feed the layer with the outputs of neurons whose values depend on more frequencies (for examples, the outputs of neurons of a pure-frequential convolutional layer with 1×9 shaped filters). Another solution

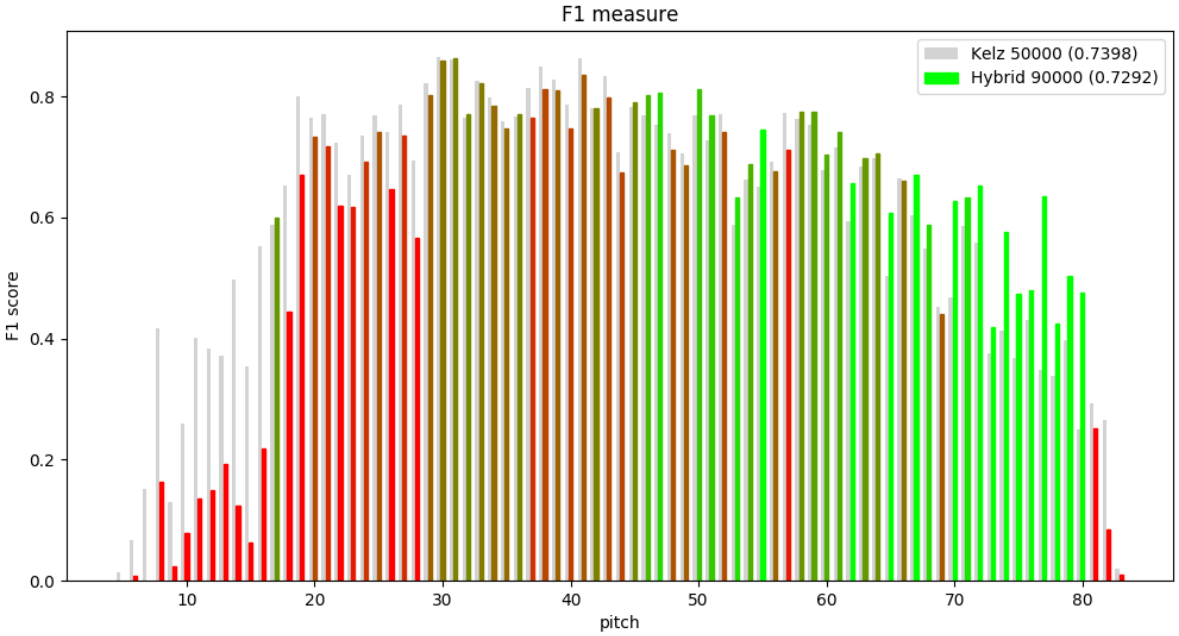

Figure 5.3: Pitch-wise comparison of the original Kelz model and the Hybrid model proposed.

could be to detect these shifts and adapt the model at evaluation time through a pre-processing step, for example using the MEL approach of section 2.1.4 to correct the bin positions of the CQT. This last approach should have better generalisation properties as it would not have to learn each different existing shift.

The second reason is that the fully connected layers of the original architecture already take harmonics into account implicitly, as shown in figure 5.4. To fully understand this figure, some reminders about the architecture could be useful. The input of Onsets and Frames is a MEL spectrogram of the signal with 229 bins, $f_{min} = 30Hz$ and $f_{max} = 8000Hz$. The input of the fully connected layer analysed in figure 5.4 is composed of $57 * 64$ values, that is, 57 groups of 64 values, each value in a group depending on the same input but differently transformed. These groups each depend on different frequency ranges, with some overlaps, as shown in figure 5.5. The graph shown in figure 5.4 shows for each of the 57 groups $\sum_{i=0}^{63} |w_i|$, with $|\cdot|$ the absolute value operation and w_i the weight given to the i^{th} value of this group. The frequency chosen to plot a group is the MEL scaled mean of its frequency range. We can clearly see a high correlation between the high values and the harmonics of the G5 note.

5.4 A faster simplified model

As the fully connected layers seem to do the job, we analysed the results of removing the convolutional layers to assess the convolutional layers' impact on the performances. This model has the advantage of being around 6 times faster, while having approximately the same performances as the original one. This model can thus be used for quick prototyping or as a fast basis for more complex algorithms.

5.4.1 The model

The model is very close to the original Kelz model shown in figure 3.7. The only difference is the Conv Stack that was replaced by a fully connected layer with batch normalisation. We will thus name this model the *Fully batched* model. This layer takes 7 full frames as input and is

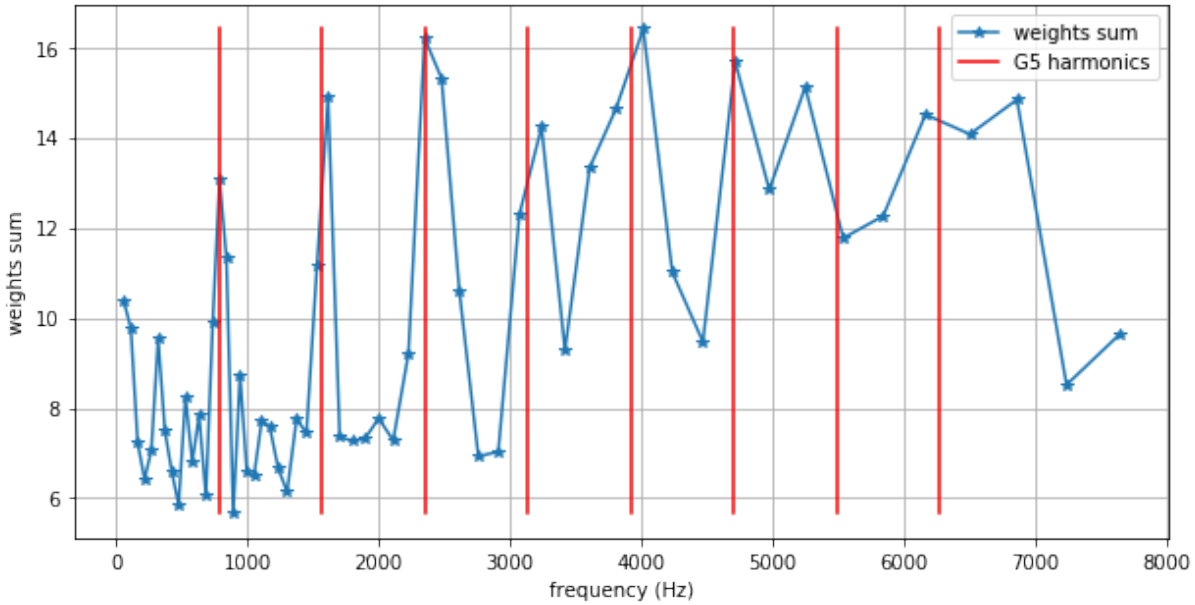

Figure 5.4: Sum of the absolute values of the weights of one of the 512 neurons of the first fully connected layer in the frame part of the original Onsets and Frames architecture, grouped by input frequency range. This illustrates the implicit learning of harmonics by the fully connected layer.

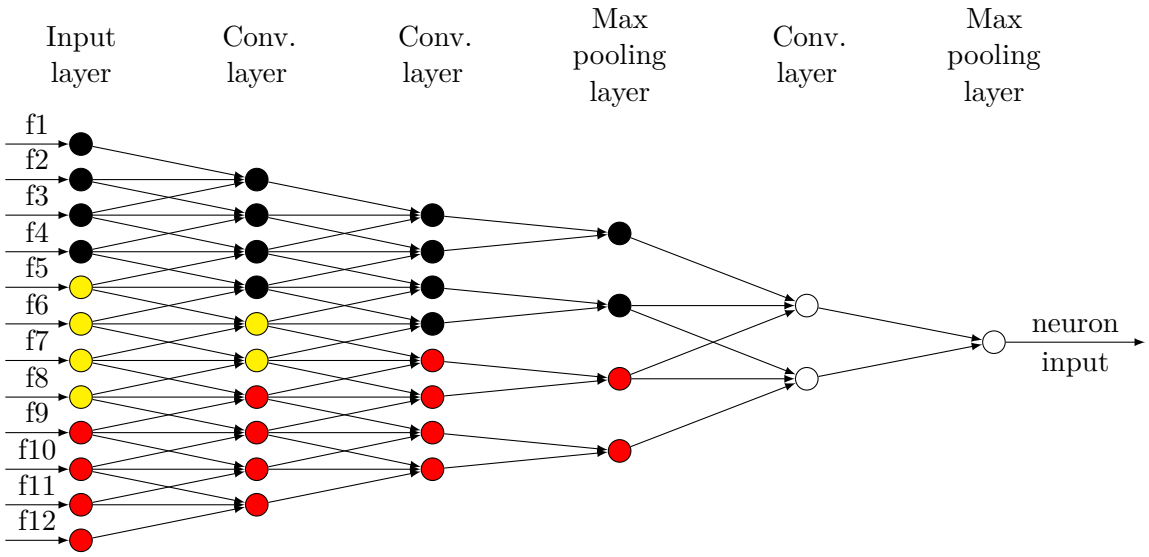

Figure 5.5: Frequency range of a group, black/red/yellow neurons being also dependencies of the previous/next/both group(s).

composed of 512 batch normalised neurons with ReLU activation, followed by a 0.5 keep rate dropout layer. We replaced the Conv Stack for both the frame side and the onset side.

5.4.2 Results

With the same computer resources as for the original model (∼20 CPUs), this simplified model is 6 times faster. In terms of performances, we achieve a **0.7225** F1 score, compared to Kelz's **0.7398** F1 score. A comparison of both ROC curves can be found in figure 5.6. The F1 score and

the ROC curve indicate that the model's performances is slightly worse, but certainly comparable. It can thus be used for prototyping, testing hypothesises, etc. The pitch-wise F1 scores are shown in figure 5.7.

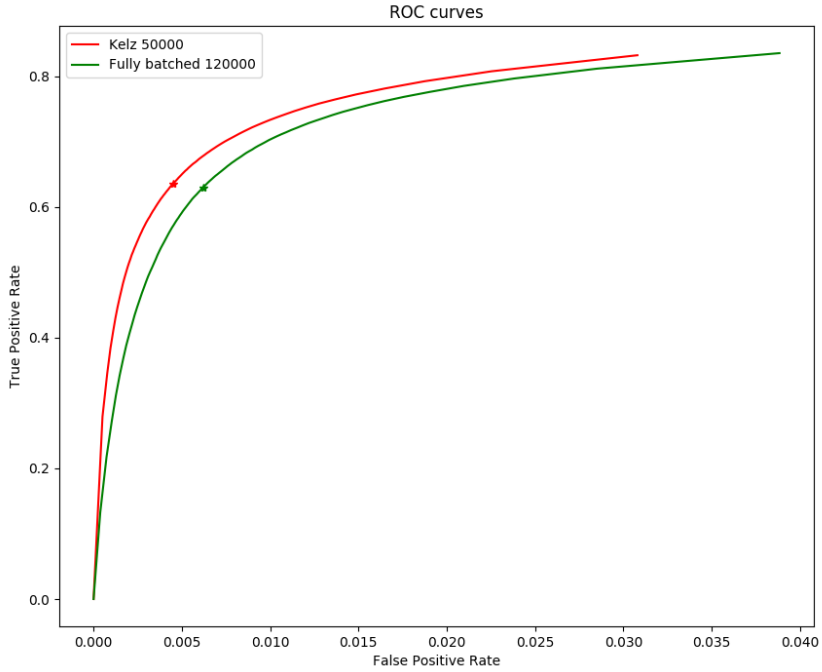

Figure 5.6: ROC curves of the original Kelz model and the Fully batched model proposed.

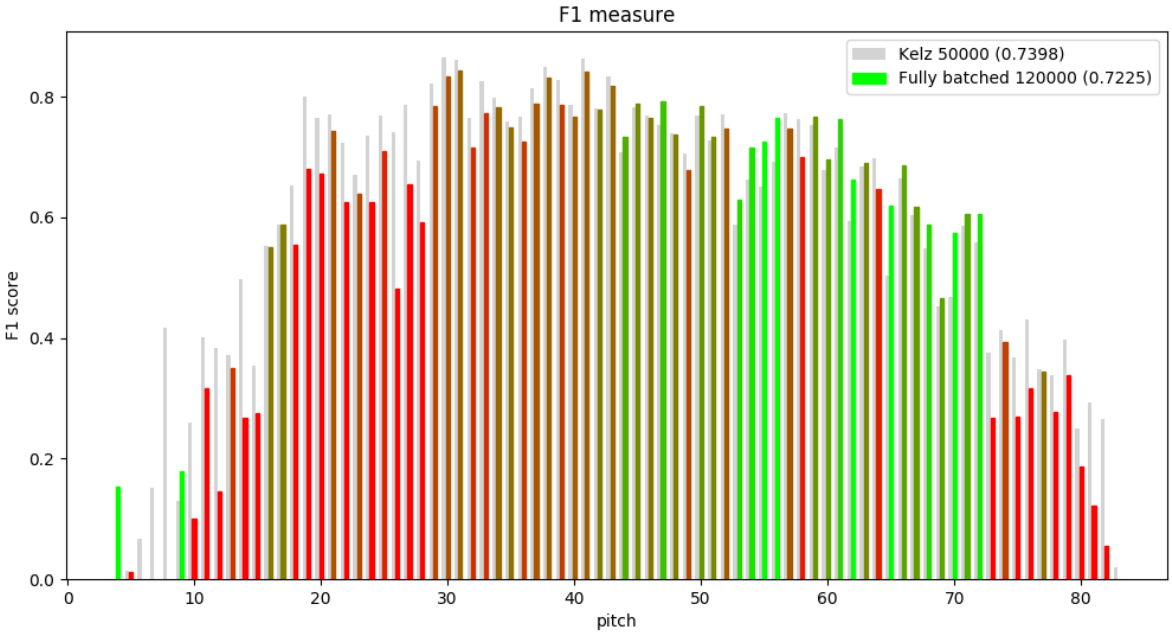

Figure 5.7: Pitch-wise comparison of the original Kelz model and the Fully batched model for the F1 score.

5.4.3 Analysis

The results show that, even if just a little, the convolutional layers have a positive impact on the predictions. Moreover, even if it takes more time in total, the model needs less training steps to reach the same performances. A possible reason is that it *softens* the dropout of the input of the fully connected layer: the convolutional layers cannot predict the notes, as it does not have the harmonics information. It can only save the most important local information in the different filters, resulting in more filters for more important information (for example, the mean, important edges, and so on). The dropout layers then remove some of these filters and the fully connected layers have to be more general. This is not the case in the simplified model, as the fully connected layers have the complete input to predict the notes before being dropped out. This first frame-wise prediction (before LSTM layers) could therefore more easily overfit on the piano settings. When applied to the harmonic layer model, this hypothesis holds, as the harmonic layers have the harmonic information and could thus overfit on the piano settings before any dropout.

Another interesting observation is the pitch-wise F1 scores which are better for a lot of high pitches. We have no hypothesis for why this is, but this could be an important research direction to improve the model.

5.5 Conclusion

This chapter clarifies some aspects of how the original Kelz model works, in particular how the architecture takes into account the harmonics. It also introduces a new type of convolutional layer, which however has not found its usefulness yet. Lastly, it proposes a faster and only slightly worse model which can be used for quick testing purposes or as a basis of new models. This simplified model shows that the convolutional layers of the original model bring only small improvements while being very time consuming. Interestingly, the tested models seem to perform better than the original model for high pitches. This could be an important direction for future research.

Chapter 6

Balancing the training pitch distribution

As introduced in section 4.3, the pitches in the training set are highly unbalanced, approximating a Gaussian distribution instead of a uniform one. In this chapter, we will introduce a balancing technique and analyse its impact on the trained model accuracy of the original Onsets and Frames [1] model (also named *Kelz* model in this thesis) introduced in section 3.3.4. The testing conditions are the same as in the previous chapter (see section 5.1), with this time the erroneous boolean corrected, meaning the loss will be weighted as in equation 3.16. The analysis will be done using the metrics introduced in section 5.2.

6.1 Balancing technique

The easiest way to balance the dataset is to modify the weights of the loss proportionally to pitch rarity, that is, to give a weight e.g. ten times bigger for pitches ten times less present. However, as some pitches are hundreds of times more present than others, this could either significantly slow down the training, as the gradients will be smaller for the very present pitches, or completely ruin the learning as the gradient will be too big for the very rare pitches. This last problem could however be prevented by the `clip_gradient_norm` hyperparameter, clipping the gradient to a maximum norm. This, in turn, would prevent the training from being really balanced, in a way that would be hardly quantifiable.

This is why we chose another technique. The idea is to use as input some sequences more often than others. In details, the score vector $\mathbf{S}_i \in \mathbb{R}^{88\times 1}$ of each sequence σ_i is computed. Each of its elements $\mathbf{S}_{ij}$ corresponds to the piano note j and is the sum of the weights attributed to this note activation during the sequence, thus following the weighting of equation 3.16 but with a score of 0 for the frames where the note is not active. Each sequence will be given a probability $\mathbf{P}_i$ based on its score vector. Each time the sequence is candidate to be the next input, it will be discarded with probability $1 - \mathbf{P}_i$. This will eventually probabilistically balance the dataset.

To compute $\mathbf{P} \in \mathbb{R}^{N\times 1}$ with $N = 1670$ the number of sequences, a convex optimisation problem is solved through classical gradient descent. We want the final distribution $\mathbf{S} \cdot \mathbf{P}$, with $\mathbf{S} \in \mathbb{R}^{88\times N}$ the matrix composed of the score vectors, to be balanced. This constraint can be modelled as a minimisation of

$$||\mathbf{S} \cdot \mathbf{P} - \mathbf{I}||^2 \tag{6.1}$$

with $||\cdot||^2$ the sum of the squares of the elements of a vector and $\mathbf{I} \in \mathbb{R}^{88\times 1}$ a vector of 1.

We also want the input to be as varied as possible, to avoid overfitting on a small number of

sequences. This can be modelled as a minimisation of

$$\frac{||\mathbf{P}||^2}{N} - \left(\frac{\sum_i \mathbf{P}_i}{N}\right)^2 \tag{6.2}$$

that is, the variance of $\mathbf{P}$. Both minimisation objectives are combined with a hyperparameter $\lambda \in \mathbb{R}^+$ to give more or less importance to one or the other, resulting in the minimisation of

$$||\mathbf{S} \cdot \mathbf{P} - \mathbf{I}||^2 + \lambda \left(\frac{||\mathbf{P}||^2}{N} - \left(\frac{\sum_i \mathbf{P}_i}{N}\right)^2\right) \tag{6.3}$$

As this minimisation problem alone gives a probability of 0 to half the dataset before having a reasonably balanced dataset, we add a last constraint, being

$$\mathbf{P}_i \geq \pi_{min} \ \forall \ i \tag{6.4}$$

with π_{min} a hyperparameter to be chosen.

After hyperparameter tuning and algebraic transformations, the solved optimisation problem is the following:

$$minimise \quad ||\mathbf{S} \cdot \mathbf{P} - 30 * \mathbf{I}||^2 + 10^7 \left(||\mathbf{P}||^2 - \frac{(\sum_i \mathbf{P}_i)^2}{N}\right) \tag{6.5}$$

$$subject\ to \quad \mathbf{P}_i \geq \frac{1}{5N} \ \forall i \tag{6.6}$$

The factor 30 introduced is useful to have a more interpretable π_{min}, as $\pi_{min} * N$ is therefore approximately the ratio between the initial probability and the new minimum probability.

This gives the distribution shown in figure 6.1, with $0.355 < \mathbf{P}_i < 12.414 \ \forall i$. We can see that the pitch distribution has really improved towards a uniform distribution while the dataset remains reasonably varied.

The last step of the process is to scale $\mathbf{P}$ such that $\mathbf{P}_i \leq 1 \ \forall i$ with $\mathbf{P}_k = 1$ for at least one k. This equality minimises the randomness of the process, but also the impact on the computing time, as less inputs will be filtered out.

6.2 Results

The optimisation problem is solved in a pre-processing step, solved in few minutes on the jabba server introduced in section 5.1. The effect of the filtering on the computing speed of a training step is small enough to be neglected.

When comparing the ROC curves of the original training and the balanced training, shown in figure 6.2, we can see the result is slightly better with balanced input, with a F1 score of **0.7627** for the balanced training and **0.7621** for the original one. A pitch-wise analysis is shown in figure 6.3.

6.3 Analysis

Even when the balancing is done by repeated sequences, the result seems to improve a little. The reason is probably that many neurons are used for middle pitches in the original training, leaving only a few for the more extreme pitches. Balancing the dataset also balances the use of the neurons as all pitches will have the same importance. This could lower the score for middle pitches, but it can overall improve the generalisation of the model and its behaviour with extreme pitches.

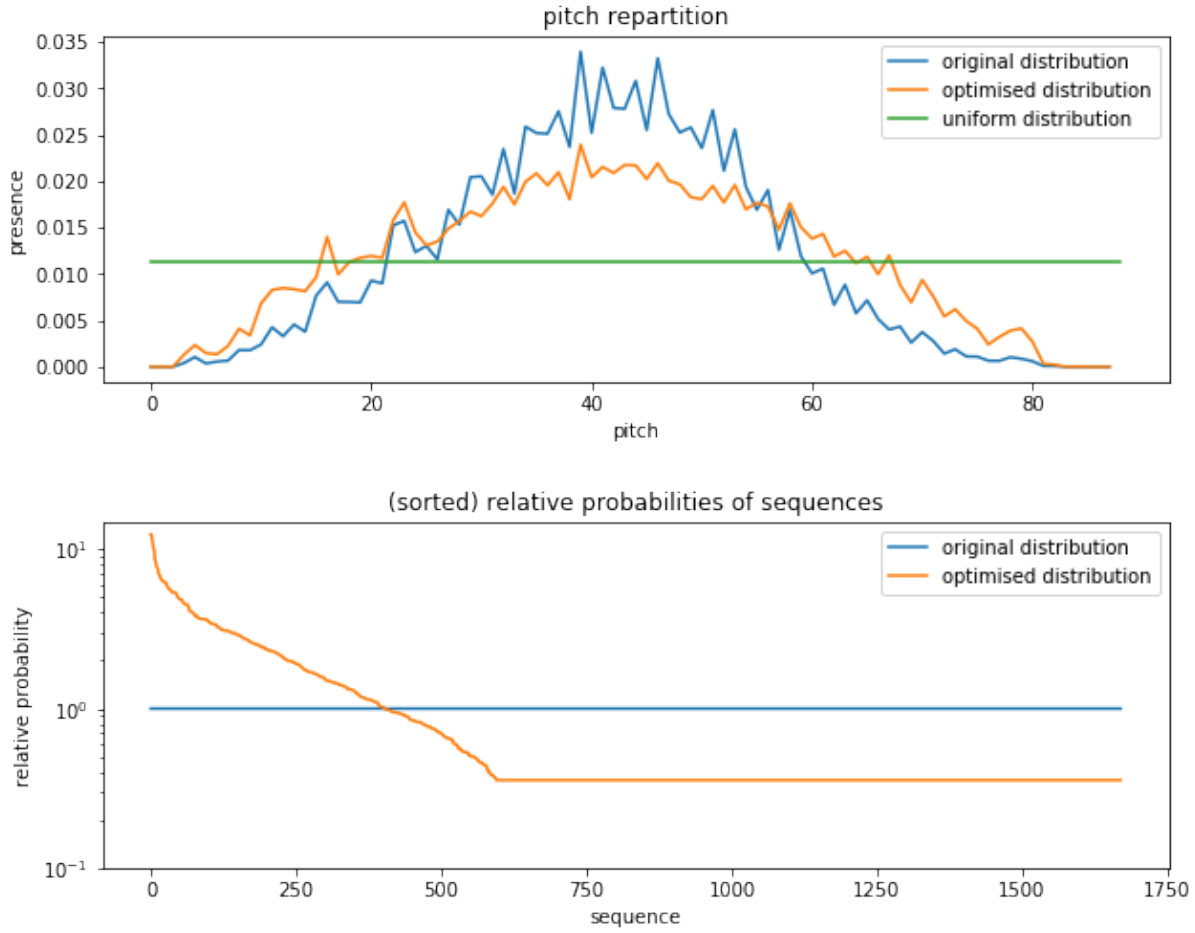

Figure 6.1: Analysis of the optimal $\mathbf{P}$ scaled such that $\sum_i \mathbf{P}_i = 1670$ (below) and the resulting pitch distribution (above).

6.4 Conclusion

The improvement is too low to conclude it is an important improvement in itself, but this could potentially become a lot better by modifying the dataset instead of the way we use it. It can be balanced by playing whole pieces some pitches higher or lower than the expected pitches, but keeping similar the relative intervals between the notes[1]. Balancing the pitch distribution with different musical performances instead of the same repeated should lead to better generalisation of these higher and lower pitches, improving the training of the model overall.

[1]This even has a name: to **transpose** a piece.

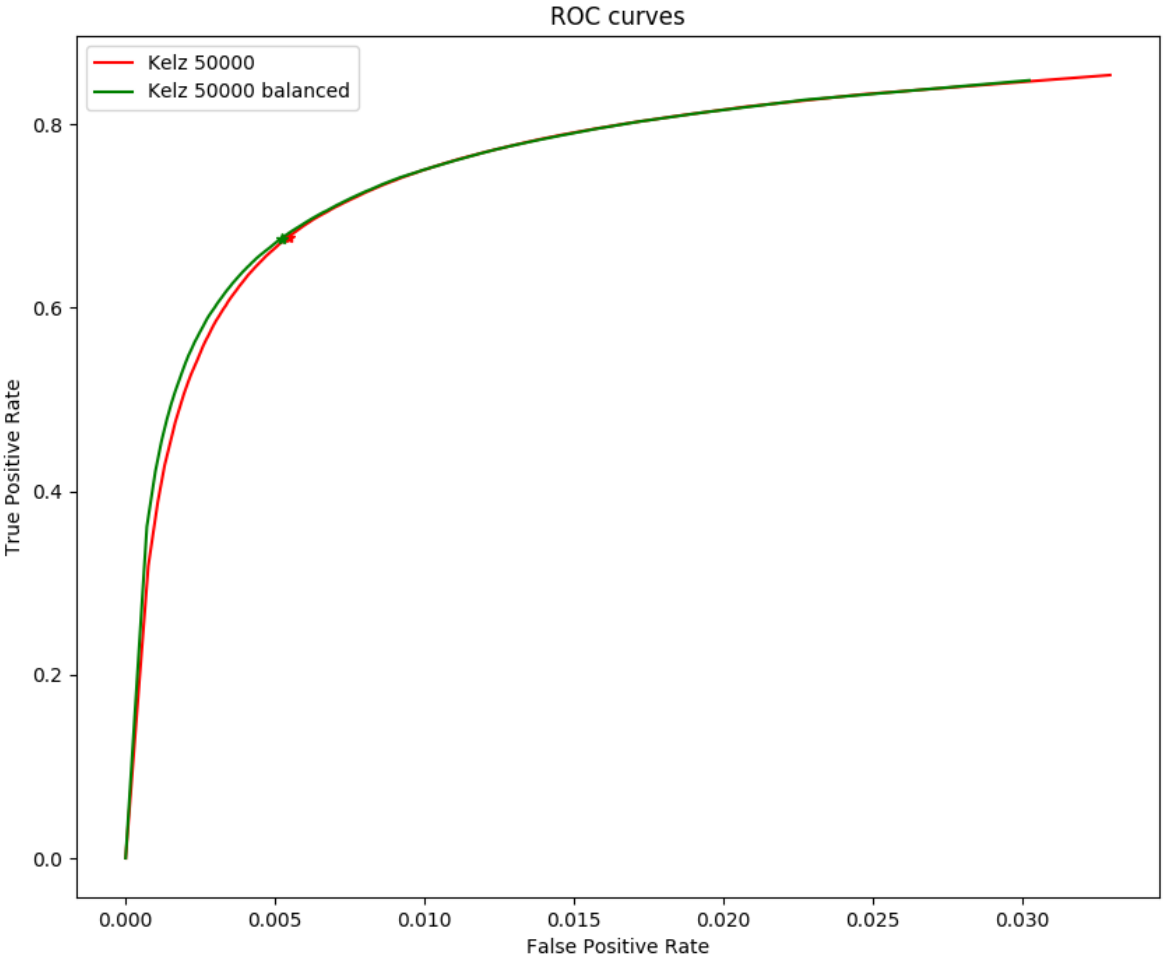

Figure 6.2: ROC curves of original and balanced training of the Kelz model.

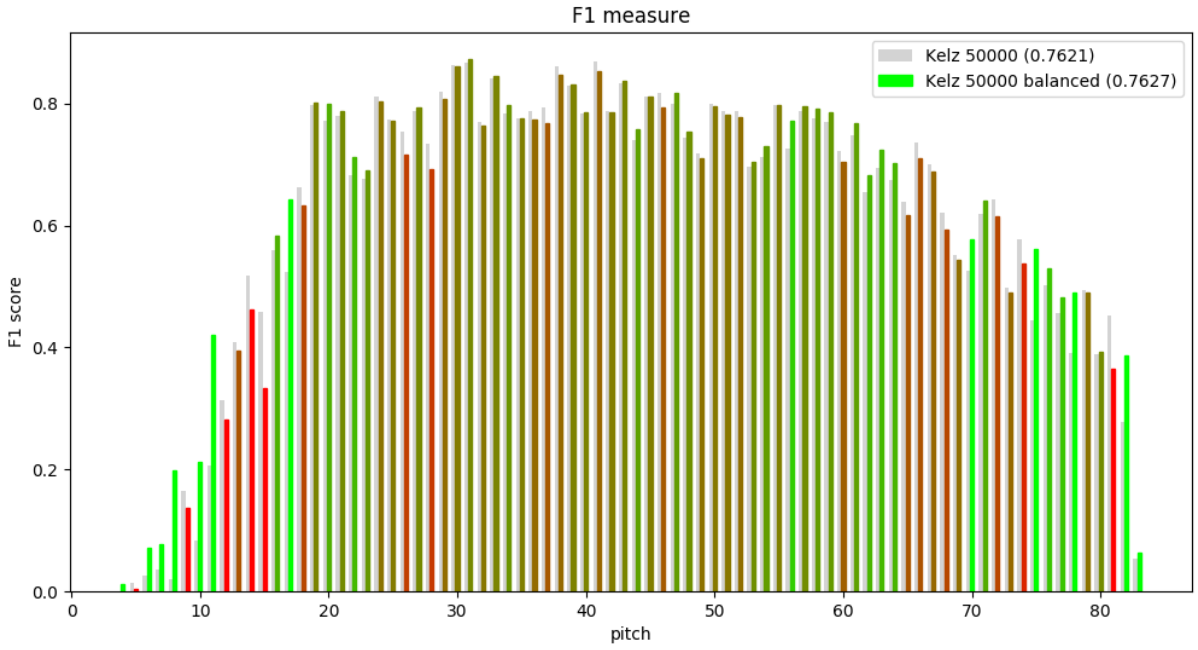

Figure 6.3: Pitch-wise comparison of the original and balanced training for the F1 score.

Conclusion

Automatic Piano Transcription is a difficult task, especially when the available labelled datasets are quite small. Some of the main drawbacks are overlapping harmonics, shifted harmonics and long-lasting notes that do not produce sound anymore. However, even with limited dataset resources, some algorithms perform reasonably well, while still being far from human expert level. One of these algorithms is the Onsets and Frames deep learning algorithm, which was analysed and improved throughout this thesis. To the best of our knowledge, the ideas proposed in this thesis are novel in the field. Due to the limited computer processing power, only *best guess models* have been trained, without proper hyperparameter tuning.

The first approach, the *harmonic layer*, takes harmonics into account for the convolution parts. When introduced in the Onsets and Frames architecture, no improvement is observed. It is probably because this architecture has its own means of taking these harmonics into account. The use of the harmonic layer is therefore redundant. However, the layer could certainly be useful in other architectures, taking advantage of its low number of parameters compared to the fully connected layer alternative.

The second approach, the *Fully batched* simplification, speeds up the predictions. The F1 score is only slightly worse and could certainly be improved with proper hyperparameter tuning, such as the number of neurons in the fully connected layer or the keep rate of the dropout layer. Adding a dropout layer before the fully connected one is also an option to be explored.

The third approach, the *balanced training*, is the most promising one, as its performances seem slightly better than the ones of the original algorithm and as greater improvement can be expected from the use of better resources. Moreover, this third approach is not specific to the APT domain and can find applications in other domains since we observed that modifying the label distribution during training can improve the model.

Finally, we introduced the pitch-wise F1 scores analysis, which provides interesting insights. State of the art approaches can probably be improved by combining models that perform better for lower pitches with models that perform better for higher pitches. To go even further, one could train models specifically for some pitch ranges.

To conclude, the Automatic Piano Transcription task is still an open problem, with many interesting research directions to be explored, including the ones proposed in this thesis.

Bibliography

[1] C. Hawthorne, E. Elsen, J. Song, A. Roberts, I. Simon, C. Raffel, J. Engel, S. Oore, and D. Eck, "Onsets and frames: Dual-objective piano transcription," *CoRR*, vol. abs/1710.11153, 2017.

[2] Lunaverus, "Anthemscore." `https://www.lunaverus.com/`, visited on Jun. 2018.

[3] 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, Oct 2003.

[4] P. Smaragdis, B. Raj, and M. Shashanka, "A probabilistic latent variable model for acoustic modeling," in *Advances in Neural Information Processing Systems (NIPS)*, Dec. 2006.

[5] 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, pp. 1643–1654, Aug. 2010.

[6] B. Gascoigne, "History of Music." HistoryWorld, `http://www.historyworld.net/wrldhis/PlainTextHistories.asp?historyid=ab42`, visited on Jun. 2018.

[7] E. Estrella, "An introduction to the elements of music." `https://www.thoughtco.com/the-elements-of-music-2455913`, Mar. 2018.

[8] Naxos Records, "Musical instruments." `https://www.naxos.com/education/music_instruments.asp`, visited on Jun. 2018.

[9] Domogalla, "Genres of popular music." `http://www.wido-software.de/popgenres.html`, visited on Jul. 2018.

[10] D. Back, "Standard midi-file format spec. 1.1, updated." `http://www.music.mcgill.ca/~ich/classes/mumt306/StandardMIDIfileformat.html`, 1999.

[11] E. Benetos, *Automatic transcription of polyphonic music exploiting temporal evolution*. PhD thesis, School of Electronic Engineering and Computer Science, Queen Mary University of London, UK, Dec. 2012.

[12] H. Stone, "R66-50 an algorithm for the machine calculation of complex fourier series," *IEEE Transactions on Electronic Computers*, vol. EC-15, no. 4, pp. 680 – 681, 1966.

[13] R. Kulkarni, "Chapter 4: Frequency domain and fourier transforms." Lecture Notes for ELE201 Introduction to Electrical Signals and Systems, Princeton University, 2000, 2001, 2002.

[14] J. Brown, "Calculation of a constant q spectral transform," *Journal of the Acoustical Society of America*, vol. 89, pp. 425–434, January 1991.

[15] C. Schörkhuber and A. Klapuri, "Constant-q transform toolbox for music processing," *Proc. 7th Sound and Music Computing Conf.*, Jan. 2010.

[16] Brian McFee, Colin Raffel, Dawen Liang, Daniel P.W. Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto, "librosa: Audio and Music Signal Analysis in Python," in *Proceedings of the 14th Python in Science Conference* (Kathryn Huff and James Bergstra, eds.), pp. 18 – 25, 2015.

[17] S. Mallat, *A wavelet tour of signal processing.* Academic Press, third ed., 2009.

[18] V. Emiya, N. Bertin, B. David, R. Badeau, "MAPS - A piano database for multipitch estimation and automatic transcription of music," Jul. 2010.

[19] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," in *Proceedings of the IEEE*, pp. 257–286, 1989.

[20] T. M. Mitchell, *Machine learning.* McGraw Hill series in computer science, McGraw-Hill, 1997.

[21] phuong, Gonzalo Medina and Torbjørn T., "Diagram of an artificial neural network." `https://tex.stackexchange.com/questions/132444`.

[22] D. Gupta, "Fundamentals of deep learning – activation functions and when to use them?." `https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use-them/`, Oct. 2017.

[23] R. Kapur, "Rohan #4: The vanishing gradient problem." `https://ayearofai.com/rohan-4-the-vanishing-gradient-problem-ec68f76ffb9b`, March 2016.

[24] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions.," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 2, pp. 107–116, 1998.

[25] T. H. Itay Lieder, Yehezkel S. Resheff, "Learning Tensorflow." `https://www.safaribooksonline.com/library/view/learning-tensorflow/9781491978504/ch04.html`, 2017.

[26] P. Veličković, "TikZ/2D Convolution." `https://github.com/PetarV-/TikZ/tree/master/2D%20Convolution`, 2016.

[27] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, pp. 2278–2324, 1998.

[28] C. Olah, "Understanding lstm networks." `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`, August 2015.

[29] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput. 9*, vol. 1735–1780, 1997.

[30] M. Schuster and K. Paliwal, "Bidirectional recurrent neural networks," *Trans. Sig. Proc.*, vol. 45, pp. 2673–2681, Nov. 1997.

[31] A. Karpathy, "Convolutional neural networks (cnns / convnets)." `https://cs231n.github.io/convolutional-networks/`, 2018.

[32] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, Jan. 2014.

[33] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift.," in *ICML* (F. R. Bach and D. M. Blei, eds.), vol. 37 of *JMLR Workshop and Conference Proceedings*, pp. 448–456, JMLR.org, 2015.

[34] C. Olah, "Calculus on computational graphs: Backpropagation." `https://colah.github.io/posts/2015-08-Backprop/`, August 2015.

[35] Google, "Machine learning crash course with tensorflow apis." `https://developers.google.com/machine-learning/crash-course/`, visited on Jul. 2018.

[36] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller, "Striving for simplicity: The all convolutional net.," *CoRR*, vol. abs/1412.6806, 2014.

[37] J. A. Moorer, "On the transcription of musical sound by computer," *Computer Music Journal*, vol. 1, no. 4, pp. 32–38, 1977.

[38] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, pp. 788–791, Oct 1999.

[39] M. Goto and T. Nishimura, "RWC Music Database: Music Genre Database and Musical Instrument Sound Database," in *ISMIR*, pp. 229–230, 2003.

[40] S. Sigtia, E. Benetos, N. Boulanger-Lewandowski, T. Weyde, A. S. d'Avila Garcez, and S. Dixon, "A hybrid recurrent neural network for music transcription," *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2061–2065, 2015.

[41] S. Sigtia, E. Benetos, and S. Dixon, "An end-to-end neural network for polyphonic piano music transcription," *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, vol. 24, pp. 927–939, May 2016.

[42] A. Lerch, "Audio content analysis: datasets." `https://www.audiocontentanalysis.org/data-sets/`, Mar. 2018.

[43] Z. Duan, B. Pardo, and C. Zhang, "Multiple fundamental frequency estimation by modeling spectral peaks and non-peak regions," *IEEE Transactions on Speech and Audio Processing*, vol. 18, pp. 2121–2133, 9 2010.

[44] R. Kelz, M. Dorfer, F. Korzeniowski, S. Böck, A. Arzt, and G. Widmer, "On the potential of simple framewise approaches to piano transcription," *CoRR*, vol. abs/1612.05153, 2016.

[45] R. W. Young, "Inharmonicity of plain wire piano strings," *The Journal of the Acoustical Society of America*, vol. 24, no. 3, pp. 267–273, 1952.

[46] G. Forman and M. Scholz, "Apples-to-apples in cross-validation studies: Pitfalls in classifier performance measurement," *SIGKDD Explor. Newsl.*, vol. 12, pp. 49–57, Nov. 2010.

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**