Imperial College
London

Department of Mathematics

Audio-to-MIDI Alignment Using Convolutional Deep Regression

Artemiy Nosov

CID: 02123846

Supervised by Dr. Kevin Webster

2 September 2022

Submitted in partial fulfilment of the requirements for the MSc in Statistics of Imperial College London

Abstract

Audio-to-MIDI alignment is the problem of obtaining an accurate mapping between the MIDI file containing the score and the audio recording containing a performance of the same piece of music. The scheme of all audio-to-MIDI alignment tasks is universal: feature extraction followed by sequence alignment. In this thesis, a novel feature extraction method is proposed using $1-$dimensional MIDI sequences as targets for a deep convolutional regression extraction model. The proposed framework is implicitly robust to minor mistakes in the performance. In addition, 3 methods for audio processing are compared for this task.

This thesis is also a snapshot in time into several contemporary audio-to-MIDI alignment frameworks, along with their methodological backgrounds.

Acknowledgements

First, I would like to thank my supervisor Dr. Kevin Webster for accepting the research proposal on the topic that is close to my heart. Kevin has been easy to work with and has given me good feedback and the advise that I needed.

I would also like to thank my friends and colleagues who I have met during during the year at Imperial, and who have kept me great company during the summer project.

I would like to thank my girlfriend, Anh, who gave me the support that I needed during the writing of the thesis.

Finally, I'd like to thank my parents, Aiza and Mike, for their constant support throughout my years in education.

Contents

1 Introduction

It is often said that music is a universal language. It can exist in multitudes of media, and parallels can be drawn between music and natural language: both can be represented by notation (musical notation and text), and both can be represented by an audio signal (musical recording and speech recording). The notation can be viewed as a set of instructions for the performer or the speaker, and the audio signal is then the finished product of a performance or a speech.

Obtaining an accurate mapping between the set of instructions and the performance is a one of the fundamental parts of audio signal processing. This mapping is known as an *alignment*, and it allows one to better analyse the performance. For example, one can analyse it for mistakes or inaccuracies, deviations from the instructions, and even allows one to compare different performances of the same set of instructions.

An illustration of a typical audio-to-MIDI alignment task is shown in ***Figure 1.1***. From the figure, it can be seen how the alignment bridges the gap between the modalities of the score (represented by the MIDI) and the audio (represented by the waveform). The MIDI file has precise structure whereas the waveform is a 1−dimensional object with no clear structure. By aligning the two, the audio recording can be augmented with the information of the MIDI – such as which notes are played and when.

The field of research devoted to developing computational tools for processing music-related data is called Music Information Retrieval (MIR). Each year, the important advances in the field get presented at a conference curated by the International Society for Music Information Retrieval (ISMIR, 2022).

Musical Instrument Digital Interface (MIDI) is a standard format for storing and transmitting musical information, such as pitches of the notes, their timings and their loudness. A MIDI file to a musical score is what an electronic book is to a paper book: a MIDI file contains the same information as the score, but it is modifiable and analysable.

A large topic of MIR is Automatic Music Transcription (AMT), which is a task of converting an audio signal of a musical performance into musical notation. This is closely related to audio-to-MIDI alignment, and in fact many alignment methods utilise AMT in order to obtain the predictions of the notes, and then this is followed by aligning the predicted notes with the notes of the MIDI score such as (Kwon et al., 2017), (Simonetta et al., 2021).

Both audio-to-MIDI alignment and the AMT have the goal of outputting an accurate representation of the recording in musical notation. The difference between the two approaches stems from the fact that the audio-to-MIDI alignment assumes that a predetermined score exists for a given recording, while the AMT has to pretend as though a musical score does not exist for the given piece of music. Of course, this is true for music that is of the improvisatory nature (and the alignment task would not be possible), but for most musical pieces a score does exist.

In this thesis we propose a novel approach for audio-to-MIDI alignment by framing the audio feature extraction as a regression problem. This feature extraction method allows for an alignment that is more robust to errors in the performance, such as incorrect notes.

In Chapter 2, we will lay out some of the musical theory that is relevant to the thesis, outline some of the most commonly used audio processing techniques, provide the background into convolutional deep neural networks, and discuss the problem of sequence alignment.

After this, Chapter 3 will see the discussion of three distinct audio-to-MIDI alignment methods that cover the three possible approaches to the problem: alignment in the audio domain, in the MIDI domain, and in the middle-ground domain (cross-modal). Each of those methods was the state of the art of its time.

In Chapter 4 we discuss in detail the methodology of our proposed method, including the pre-processing steps, the models proposed, the alignment method and the evaluation methods. Following that, the results are discussed in Chapter 5.

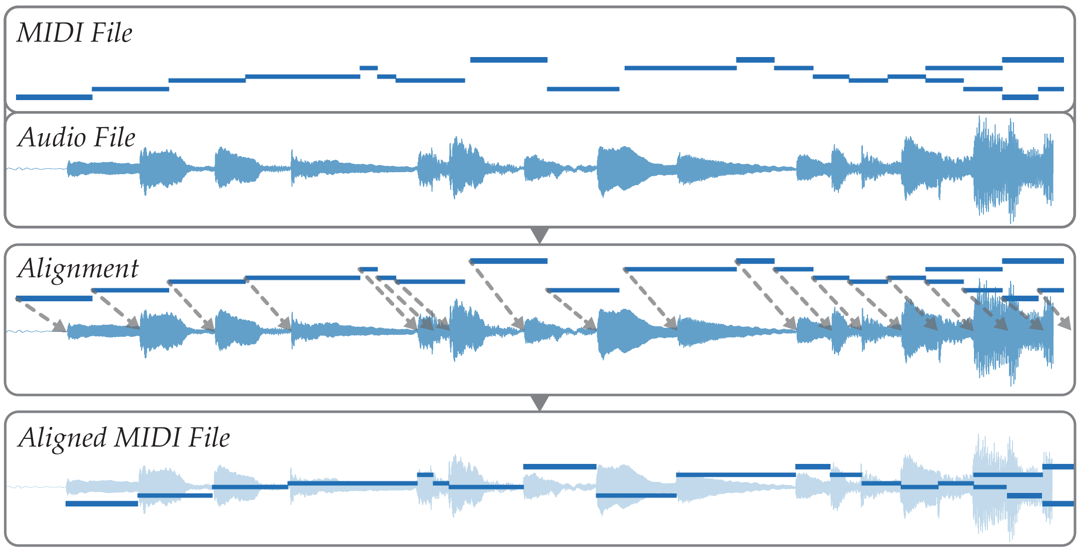

Figure 1.1: *Example of a typical audio-to-MIDI alignment task. The MIDI file is a digital version of the score. By aligning the audio and the MIDI, one is able to augment the audio information of the recording with the structured musical information of the score. Since the MIDI has precise structure while the audio does not, this is useful for a variety of audio analysis problems, most notably audio-to-score transcription.*

2 Background

All audio-to-MIDI alignment methodology shares a universal common thread: feature extraction followed by sequence alignment. The added complication is that the quality of the alignment depends both on the quality of the extracted features as well as on the alignment method. Some authors focus on improving the alignment by better feature extraction methods (Kwon et al., 2017), (Wang et al., 2019), while other authors focus on the alignment methods (Agrawal et al., 2022).

In this chapter, the common methods for audio signal processing, MIDI processing, and machine learning, as well as sequence alignment are defined.

2.0.1 Musical Media

A piece of music can exist in 2 distinct forms: score and audio signal.

The score contains a set of instructions that give the necessary information to the performer in order to execute the author's idea. In Western Classical Music (Finson, 2002), the score contains the pitches of the notes that are to be played and their timings, as well as the dynamics (how loud) and tempo (how fast they are to be played). For a more comprehensive overview of the musical score, see (Müller, 2021).

Four medias of the same piece of music are shown in ***Figure 2.1***: the composer writes a manuscript (A), which is then turned into musical score (B), which is then performed by a pianist (C), and (D) is the musical score as it is accessible to a computer.

2.0.2 MIDI File

The MIDI format is a great success story of the 20th century music production. It was the first standardized interface that allowed electronic musical instruments, such as synthesizers, to communicate with the computers. Nowadays most electronic piano keyboards have the support for MIDI. It is also a standard way to sharing musical notation between different notation software.

A MIDI file is built upon chunks of information known as *events*. An event can be: key press (and release), tempo change (speed of playback), instrument change, channel change, piano pedal press (and release).

The MIDI file also has a resolution, which is the distance between to eighth-notes. This distance is measured in *ticks*. A standard resolution is 96 ticks. This can be thought of as the MIDI equivalent of the Planck length (Mead, 1964), in a sense that distances

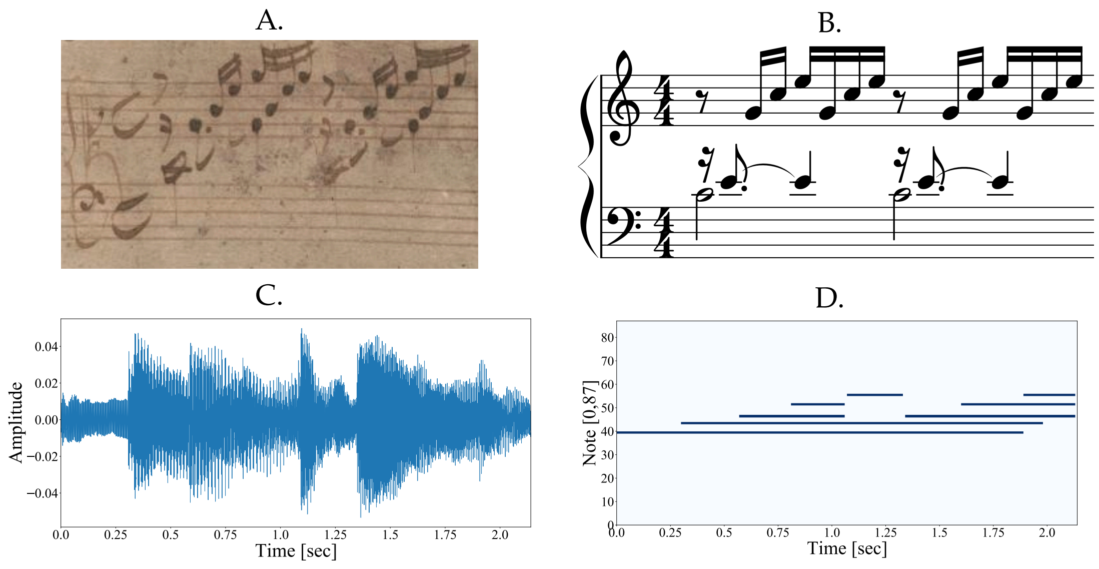

Figure 2.1: *Four medias of the same music: (A) Johann Sebastian Bach's paper manuscript from circa 1721, (B) musical score, (C) audio signal of a performance as a waveform, (D) piano roll MIDI transcription.*

shorter than 1 tick can not be measured by the MIDI standard.

A common way of MIDI processing is a *piano roll*, an example of which is shown in ***Figure 2.1 (D)***. It has the information on what note is pressed ($y-$axis), when it is pressed ($x-$axis), and how loud. Not accounting for the piano's sustain pedal, this is enough information to capture a piano performance in its entirety. As an aside, the phrase "piano roll" comes the the late 19th century – when the microphones were not an advanced technology – a popular way of recording piano recordings was in the paper roll format. These paper rolls could then be played back on a specialised reproducing piano.

For the purposes of this thesis, we define the piano roll as a (sparse) matrix $R \in \mathbb{R}^{88 \times P}$, where P is the length of the MIDI track quantized (rendered) at a some resolution. A common quantize resolution is 10ms, meaning that two consecutive vectors in the piano roll are 10ms apart. A value R_{ij} in the piano roll matrix represents the velocity (the loudness) of the note i at time j. This matrix is usually sparse, because the values are mostly zeroes since each column represents an 88−key piano keyboard and the pianist can only physically press a limited number of keys at a time.

It is very common to omit the velocity information from the piano roll in MIDI processing tasks, which turns the piano roll into a binary matrix. Then, the binary piano roll matrix is defined as:

$$R_{ij} = \begin{cases} 1 & \text{if note } i \text{ is pressed at time } j, \\ 0 & \text{otherwise.} \end{cases}$$

Binary piano rolls are used throughout this thesis. An example of a binary piano roll is shown in ***Figure* 2.2**.

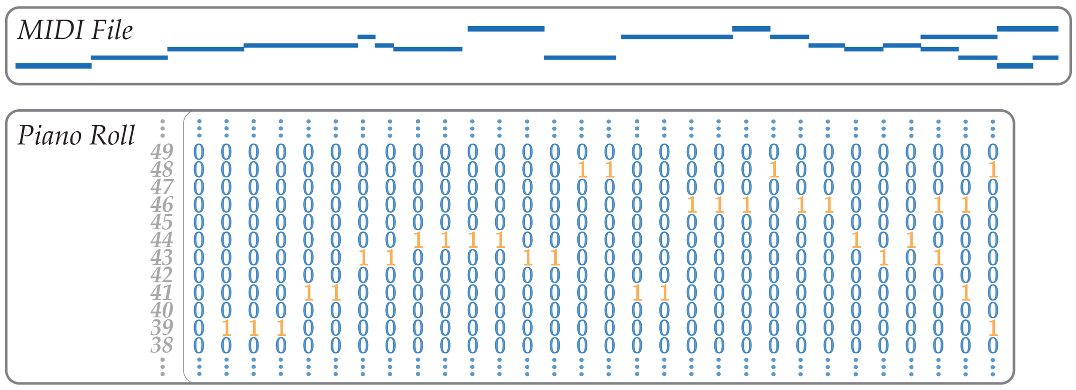

Figure 2.2: *An example of a MIDI piano roll sparse matrix. This is the most common representation of a MIDI file. The $y-$axis corresponds to the* 88 *piano keys, the $x-$axis is the time. The columns are spaced linearly in time. For visualization, a low temporal resolution was chosen to limit the number of columns. In practice, the resolution is a lot finer than shown here.*

2.1 Audio Processing

The audio signal is a 1−dimensional object, containing solely the data about the amplitude of the wave at any time point. This object, however, is rich in information. The single wave might be encoding a variety of different things – from a beautiful piece of music, to a radio weather news report. Our brains are able to pick up this wave signal and decode it into meaningful features.

In this section, various audio processing methods are discussed that allow the computers to transform audio signal into more meaningful objects which can then be used for audio-to-MIDI alignment tasks.

2.1.1 Pitch

A pitch of a single sound is a quality associated with the frequency of the wave producing the sound: the higher the frequency, the higher the pitch, and vice versa.

A standard modern piano contains 88 keys (or notes), each associated with a pitch. To get from one key to the next, the frequency is multiplied by a factor of $2^{1/12}$. The piano is tuned such that the lowest key (note pitch 0) produces a 27.5Hz wave. The formula for the frequency of note pitch n is given by:

$$f(n) = 27.5 \times 2^{n/12}$$

Hence, the highest piano key (note pitch 87) produces a $27.5 \times 2^{87/12} \approx 4186$Hz wave.

12 consecutive notes make up an *octave*, and consequently any two notes that have frequencies differing by a factor of 2 are said to be "an octave apart". This also means that there are 12 *pitch classes*, which are also known as *chroma features*. The frequencies for a range of piano notes which are split by octaves of 12 notes are shown in ***Table 2.1***. Chroma features are often used in the audio-to-MIDI alignment as an augmentation of the 88 piano notes – the pitch class information is introduced by augmenting the 88 note piano roll matrix with an extra 12 notes, for 100 features in total.

In order to avoid having to dive into the rabbit hole of 18th century Western Classical Music Theory (which is outlined in great by Hullah (1876)), we will only define the notes by their pitch numbers: the pitches correspond to the pitches of the piano where 27.5Hz representing pitch 0, and ≈4186Hz representing pitch 87.

	Pitch class											
	A	A#	B	C	C#	D	D#	E	F	F#	G	G#
n	0	1	2	3	4	5	6	7	8	9	10	11
f(n)	27.5	29.14	30.87	32.70	34.65	36.71	38.89	41.20	43.65	46.25	49.00	51.91
n	12	13	14	15	16	17	18	19	20	21	22	23
f(n)	55	58.27	61.74	65.41	69.3	73.42	77.78	82.41	87.31	92.50	98.00	103.83
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
n	84	85	86	87								
f(n)	3520	3729.31	3951.07	4186.01								

Table 2.1: *The frequencies of the* 88 *piano keys, split across* 12 *different notes. The first two octaves, and the last* 4 *notes are shown. Here n denotes the note pitch number, and* $f(n) = 27.5 \times 2^{n/12}$ *denotes the frequency in Hz.*

2.1.2 Overtones

In addition to the main frequency, each key produces harmonic overtones. The nth overtone of a frequency f_0 has frequency nf_0 Hz. The amplitude of the waves of each overtone frequency determines another quality of sound, called timbre. Timbre is the quality that allows one to tell a difference between the same note played at the same loudness on a piano, trumpet or a guitar.

An example of the overtones generated by a single piano note is shown in ***Figure 2.3*** (left). The note that is played in the first note of ***Figure 2.1***, which is note pitch 39. Its main frequency (f_0) is $27.5 \times 2^{39/12} \approx 261.6$Hz, which can be seen as the largest peak in the figure. Furthermore 5 other overtones with frequencies $2f_0, \ldots, 6f_0$ are visible in the plot.

2.1.3 Digital Signal Processing

Timbre is also the quality that makes it difficult to tell multiple notes apart when they are played together. This is where Digital Signal Processing (DSP) comes in to help.

DSP is a domain of electrical engineering focused on analysing and altering electrical signals. Some of its important results and methods allow one to decompose a given waveform signal into simpler waveforms, as well as to convert an audio signal from the amplitude-time domain of the waveform into a frequency-time domain.

2.1.4 Fourier and Discrete Fourier Transforms

The Fourier Transform (FT) (Bracewell and Bracewell, 1986) is arguably the bread-and-butter of audio signal processing. It converts an audio signal from a amplitude-time into amplitude-frequency domain, which allows for a decomposition of a signal into its constituent frequencies. A Fourier Transform of a continuous function h, evaluated at frequency f, is defined as:

$$FT(h)(f) = \int_{-\infty}^{\infty} h(x) \exp\left\{-2i\pi f x\right\} dx, \quad \forall f \in \mathbb{R},\ i = \sqrt{-1}.$$

The absolute value of $\left|FT(h)(f)\right|$ is then the magnitude of the signal of frequency f. If the amplitude is zero, it suggests that such frequency is not present in the signal. Evaluating the integral on a range of frequencies produces a *spectrum* of the wave, and can be used to determine the constituents of the wave.

While the sound itself is a continuous signal, electronic microphones are only able to pick up the signal at a certain rate, due to the nature of an electric current. The recorded signal is then digital. A common sampling frequency is 44.1kHz, or 44,100 samples every second.

A discrete Fourier transform (Rao et al., 2010) is a discrete approximation of the Fourier transform. Let $(t_0, \ldots, t_{N-1})$ be a sequence of sampling times such that $x_j = h(t_j)\ \forall j \in \{0, \ldots, N-1\}$. The Discrete Fourier Transform (DFT) of the signal x evaluated at the frequency $f = n \in \{0, \ldots, N-1\}$ is defined as:

$$DFT(x)(f) = \sum_{k=0}^{N-1} x_k \exp\left\{-2\pi i k f\right\}, \quad \forall f \in \mathbb{R},\ i = \sqrt{-1}.$$

From this one can see that computing the DFT on the full range of frequencies $0, \ldots, N-1$ requires $O(N^2)$ computations, which is quite expensive for large N. A very popular and efficient approach for computing the DFT is using the Fast Fourier Transform (FFT), the details of which are given by Cooley and Tukey (1965).

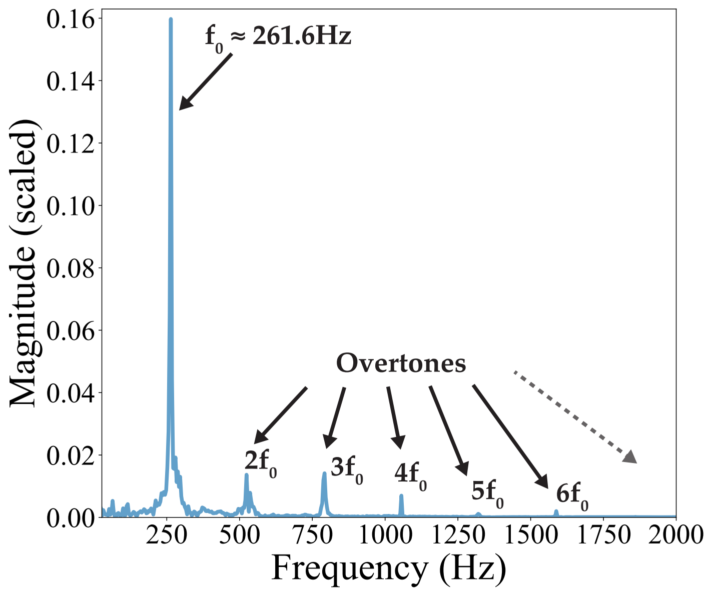

Figure 2.3: *Discrete Fourier Transform (DFT) spectrum of a piano note pitch* 39*, of frequency* 261.6*Hz, from the first* 0.25*s of music from Figure 2.1. The largest peak corresponds to the main frequency f_0, and smaller peaks correspond to the overtones with frequencies $2f_0 = 523$Hz, $3f_0 = 785$Hz, $4f_0 = 1047$Hz, and so on.*

2.1.5 Constant-Q Transform

As we discussed, the pitches of the piano are geometrically spaced, and two consecutive pitches are spaced by a factor of $2^{1/12}$ from each other. A DFT maps the signal into the frequency-magnitude domain where the frequencies are linearly spaced. This makes the DFT not an efficient way of mapping the signal into the frequency domain, since the higher frequencies are spaced wider than the lower frequencies.

The Constant-Q transform (CQT) (Brown, 1991) is a variation of DFT that utilises geometrically-spaced frequency bins instead of the linearly-spaced ones. This makes for a more efficient way of mapping the frequencies to their corresponding notes. CQT also more accurately captures the way humans hear distances between pitches: a distance between the 27.5Hz and 55Hz frequencies are perceived in a similar way as a distance between 2000Hz and 4000Hz.

A computationally-efficient definition of CQT is given by Schörkhuber and Klapuri (2010): the CQT of a discrete signal $(x_1, \dots, x_n)$ evaluated at the frequency bin k can be computed as:

$$CQT(x)(k) = \sum_{j=n-\lfloor N_k/2 \rfloor}^{n+\lfloor N_k/2 \rfloor} x_j \, a_k^* \, (j - n + N_k/2)$$

where N_k is the window length, and a_k^* is the complex conjugate of:

$$a_k(n) = \frac{1}{N_k}\left(\frac{n}{N_k}\right)\exp\left\{-i2\pi n\frac{f_k}{f_s}\right\}$$

The frequency bin $k \in \{0, \ldots, K-1\}$ is centered at frequency:

$$f_k = f_0 \times 2^{\frac{k}{B}}$$

where f_0 is the frequency of the lowest frequency bin, and B is the number of frequency bins per octave.

When processing piano recordings, the number of frequency bins per octave is usually set to a multiple of 12, since there are 12 notes in an octave. The lowest frequency bin is usually set to 27.5Hz, which is the lowest piano note. Additionally, another multiple of 12 is usually added to the total number of frequency bins in order to cover the potential overtones in the high frequencies.

Figure 2.4 shows a comparison in the spectrums produced by DFT (left) and CQT (right). From the figure, we can also see that the overtones are less prominent in the CQT compared to DFT.

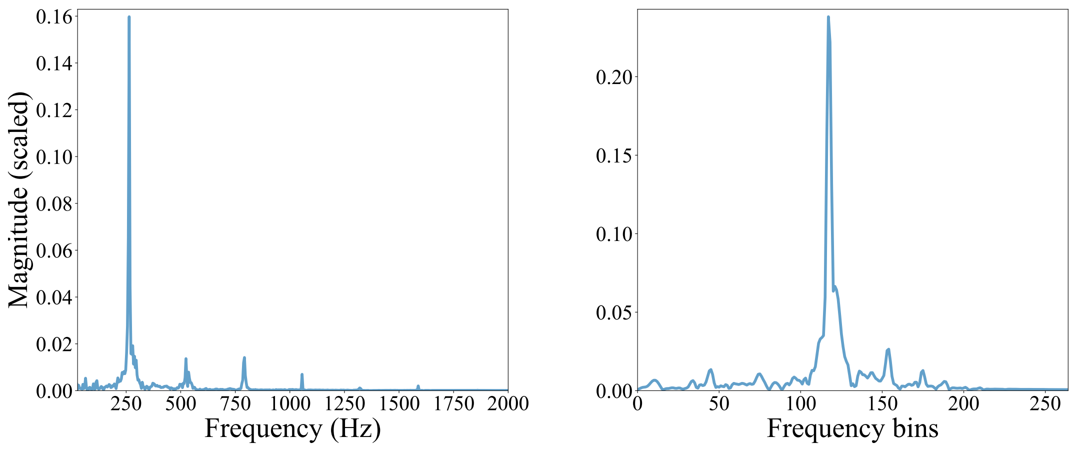

Figure 2.4: *A spectrum of the first note (first 0.25s) of the audio track from **Figure 2.1** computed by the absolute magnitude of the DFT (left), and CQT with 36 bins per octave (right). A single note of frequency* ≈ 261.6Hz *is played. y-axes are shared.*

2.1.6 Short-time DFT and CQT

While the DFT and CQT are useful for determining the frequencies of a single audio signal on their own, an audio recording usually consists of constantly-changing sounds of different frequencies. Therefore one should be able to tell the frequencies of the audio recording at any given timestamp.

The most common method for converting the audio signal into the frequency-time domain is by splitting the audio recording into chunks, known as *frames*, and computing the transforms (e.g. DFT or CQT) on each chunk. This method is then called a *short-time* transform. The audio chunks need to be short enough that they offer high temporal resolution, and long enough that they capture enough of the wave signal that the transforms can be accurately computed. The distance between two consecutive frames is known as a *hop length*.

In order to maximise the temporal resolution, overlapping frames are often used. This way, a reasonably wide frame length can be achieved with a small hop length. The drawback of this is that a frame might contain information that is more relevant to its neighbouring frames.

The rectangular windows have a drawback that they crate discontinuities at each end of the frame. The Fourier transform does not deal with such abrupt changes well, and causes artefacts throughout the spectrum. To prevent this, *window* functions are commonly used. They apply suppression to the audio signal that is away from the frame centre. ***Figure 2.5*** shows a few examples of the window functions. Given an audio signal frame $x = (x_0, \dots, x_{N-1})$, the window function W applied to the signal is defined as a dot-product between the weights $w = (w_0, \dots, w_{N-1})$ and x:

$$W(x) = (w_0 x_0, \dots, w_{N-1} x_{N-1})$$

where $(w_0, \dots, w_{N-1})$ are the weights of a window function. For example, the most common window functions for audio processing for audio-to-MIDI alignment are the Hanning and the Hamming functions (Sangeetha et al., 2021), whose weights are defined as:

$$\begin{aligned} w_i^{\text{Hanning}} &= 0.5 + 0.5 \cos\left[\frac{2\pi n_i}{N}\right] \\ w_i^{\text{Hamming}} &= 0.54 + 0.46 \cos\left[\frac{2\pi n_i}{N}\right] \end{aligned}$$

for all $i \in \{0, \dots, N-1\}$, where $n = \left(-\frac{N}{2}, \dots, \frac{N}{2}\right)$.

The output of a short-time DFT or CQT analysis are a collection of spectrums which correspond to each frame in the audio, known as a *spectrogram*. Spectrograms are used extensively in audio processing, from processing speech (Arias-Vergara et al., 2021) and sound-source separation (Higuchi et al., 2004) to COVID-19 detection (Rodriguez et al., 2020).

An example of an audio processing pipeline using short-time CQT and a Hamming window function is shown in ***Figure 2.6***.

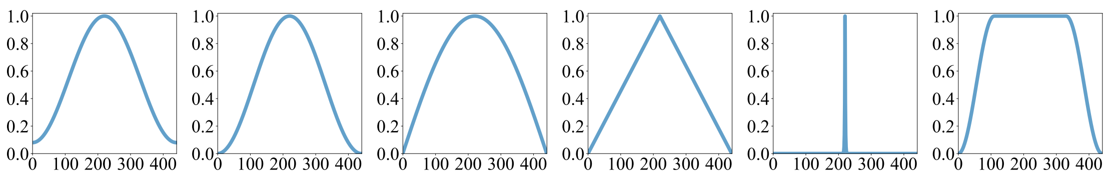

Figure 2.5: *Common window functions. Left to right: Hamming, Hanning, cosine, Bartlett, exponential, Tukey. x-axis represents to the index inside the frame, y-axis is the filtering coefficient.*

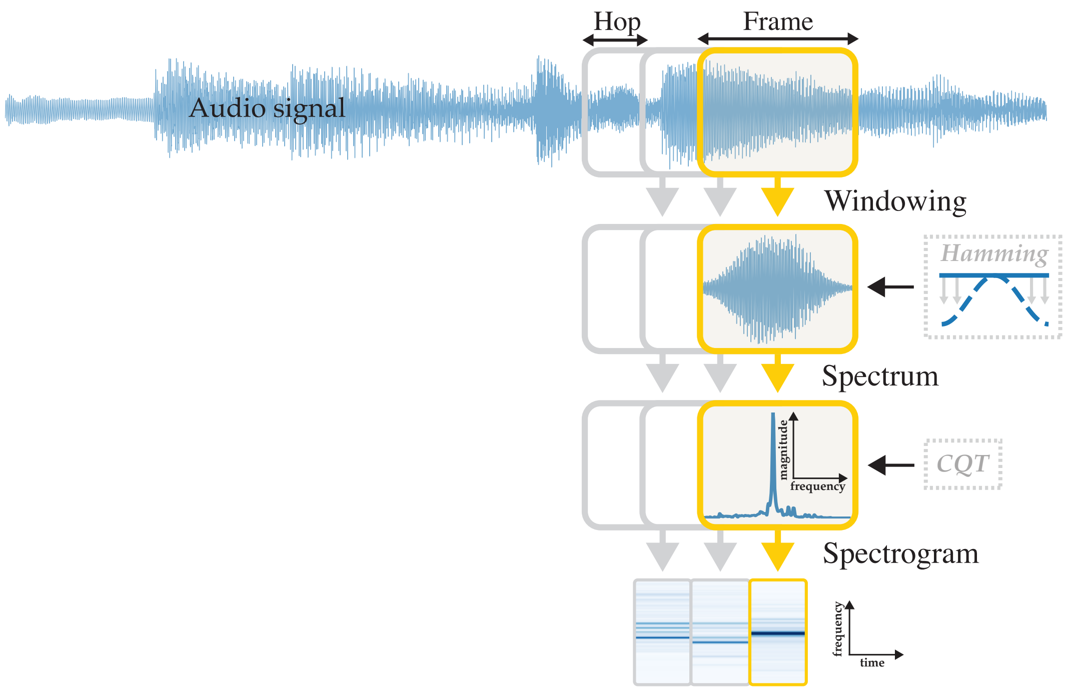

Figure 2.6: *Example pipeline for audio processing depicting a Hamming window function and CQT which are used for windowing as spectrum computation respectively. For illustration purposes, considerably large frame and hop lengths are used.*

2.2 Machine Learning

In the previous section, the common processing methods for the audio recordings were outlined, arriving at the spectrogram. Due to the 2−dimensional nature of a spectrogram (frequency and time), they are most commonly visualised and analysed as images (Brusa et al., 2021), (Khalighifar et al., 2021), (Dennis et al., 2014).

When it comes to image processing, few benchmarks have established themselves as competitive as the ImageNet's Large Scale Visual Recognition Challenge (ILSVRC) (Russakovsky et al., 2014). The ImageNet challenge consists of a database of over 14 million hand-annotated images, with over 20,000 potential categories. Since 2012, all of the winning models have been deep neural networks; and since 2015 have surpassed the human classification accuracy of 95% (He et al., 2015).

In particular, the winning model in 2012 AlexNet (Krizhevsky et al., 2012) had outperformed its competition by a margin so wide that it sparked a revolution in artificial intelligence research (Liu et al., 2018).

This provides a good morale for using deep learning methods for the purpose of extracting information from the audio spectrogram.

2.2.1 Deep Neural Networks

Deep learning is a subdivision of machine learning that is aimed at capturing and modelling high-level, or abstract, features in the data (Hinton, 2006). As was mentioned earlier, deep learning models are currently the state-of-the-art for a range of image classification and recognition tasks, and most modern audio-to-MIDI alignment methods use deep neural networks as part of their feature extraction.

In this section, the feedforward artificial neural network (FNN) is defined, as well as the model class known as the *Multi-Layer Perceptron* (MLP) (Rosenblatt, 1958).

We also define another type of networks known as the convolution neural network, which are essential in the computer vision tasks.

2.2.1.1 Multi-Layer Perceptron

A fully-connected FNN consists of an input layer, a number of fully-connected hidden layers, and an output layer. Each neuron in a layer i takes every output from the layer $i-1$, computes their weighted sum, adds bias, and passes the sum through an activation function to obtain an output. The output then serves as an input to the next hidden layer. It can therefore be seen that the FNNs are composed of functions where each layer serves as a linear function.

The network is called *fully-connected* when every output of the layer $i-1$ serves as an input to every input in layer i. The layers in these networks are called *dense* layers. An example of a fully-connected neural network is shown in ***Figure 2.7***.

Let $x = (x_0, \dots, x_{N-1})$ be the output vector of layer $i-1$, and $h = (h_0, \dots, h_{M-1})$ be a layer of M neurons in the layer i. The *weights* between the two layers is a matrix $w \in \mathbb{R}^{N \times M}$ such that $w_{j,k}^{(i)}$ is the weight of the connection between x_j and h_k. The output of the ith layer h, is then computed as follows:

$$h_k = \sigma\left(b_k + \sum_{j=0}^{N-1} w_{jk} x_j\right) = \left(g^{(i)}(x)\right)_k, \quad \forall\, k \in \{0, \dots, M-1\} \tag{2.1}$$

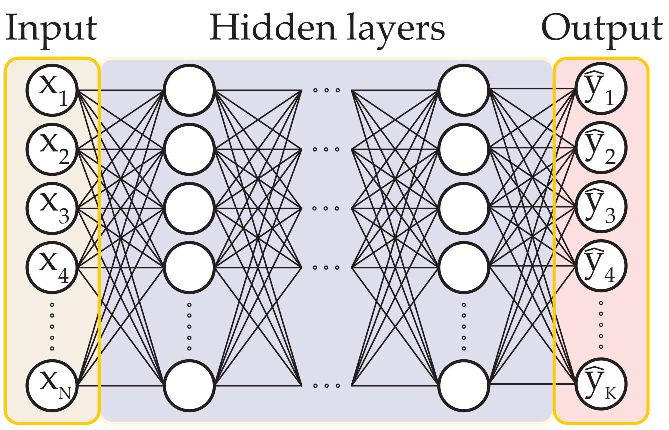

Figure 2.7: *A sketch of a fully-connected neural network with dense hidden layers, taking a vector X of length N as an input and outputting a vector $\hat{Y}$ of length K. Each circle represents a neuron, and each line represents a weight. Each hidden layer can contain any number of neurons, and each neuron acts as a function of all the weights and neurons of the previous layer according to* (2.1).

where $b = (b_0, \ldots, b_{N-1})$ is a vector of biases, σ is an activation function, and k represents the kth element in the vector. The parameters to be estimated are the biases b and the weights w.

From the above computation , it can be seen that the output after two layers is $g^{(i+1)}(h) = g^{(i+1)}(g^{(i)}(x))$. Hence for an MLP network with N layers, the output $\hat{y}$ of the entire network is given by the composition of all functions $g^{(i)}$, $i \in \{0, \ldots N-1\}$:

$$\hat{y} = f(x^{(0)}) = \left(g^{(N-1)} \circ g^{(N-2)} \circ \cdots \circ g^{(i)} \circ \cdots \circ g^{(0)}\right)(x^{(0)})$$

where $x^{(0)}$ is the input vector.

An example of a simple MLP network with a single hidden layer of two neurons is shown in ***Figure 2.8***.

2.2.1.2 Activation function

From (2.1), it can be seen that the outputs of each layer get passed through an activation function. Typically, a single activation function is chosen for a given layer. Activation functions play a similar role in the neural networks as the link functions in generalized linear models (GLMs): they allow a linear combination of the inputs to be related to the output through some function σ.

A list of common activation functions is shown in ***Table 2.2***, which are then plotted in ***Figure 2.9***.

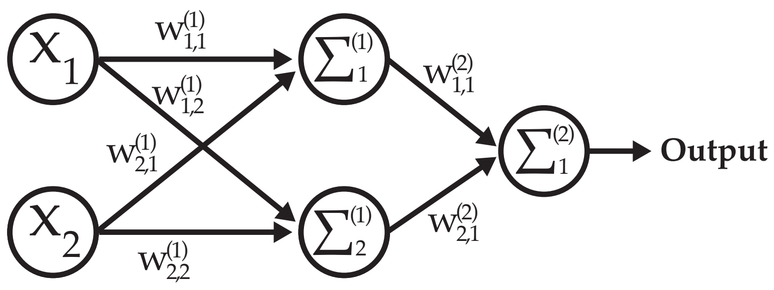

Figure 2.8: *An example of a multi-layer perceptron with two layers of two neurons each, followed by a single output neuron. The symbol $\Sigma^{(i)}$ represents the function $g^{(i)}$ from* (2.1) *applied to the layer $i-1$ with weights $w^{(i)}$. The output of this particular network is given by $\Sigma^{(2)} = g^{(1)}\left(g^{(0)}(x)\right)$.*

In supervised learning – when the targets are given – the choice of the activation functions usually depends on the domain of the targets.

For classification tasks, such as those in the AMT models for classifying which notes are played at any given time, a sigmoid function is commonly used to output probabilities. In this case $g^{(i)}$ equivalent to a logistic regression.

When the function is the identity $\sigma(x) = x$, also known as *linear* activation, then $g^{(i)}$ is equivalent to a linear regression.

The ReLU activation function is particularly widely used in neural networks as it is found to speed up the convergence of the network during training (Krizhevsky et al., 2012). Due to the gradient-based approach of the training, the ReLU also implicitly induces a regularization (Glorot et al., 2011) onto the model by mapping some of the neurons to 0, which then remain at 0 for the duration of the training (known as "dead neurons").

Name	$\sigma(x)$	Domain
Linear	x	$(-\infty, \infty)$
ReLU	$\max(0, x)$	$(0, \infty)$
ELU	$\max(\alpha(e^x - 1), x)$	$(-\alpha, \infty)$
Sigmoid	$1/(1 + e^{-x})$	$(0, 1)$
Hyperbolic-tangent	$(e^x - e^{-x})/(e^x + e^{-x})$	$(-1, 1)$
Softplus	$\ln[1 + e^x]$	$(0, \infty)$

Table 2.2: *Equations of the common activation functions along with their output domain. The choice of the activation functions depends on the prediction task. ReLU is usually used for positive outputs, while sigmoid is usually used for predicting the probabilities. Linear activation is used for linear regression problems.*

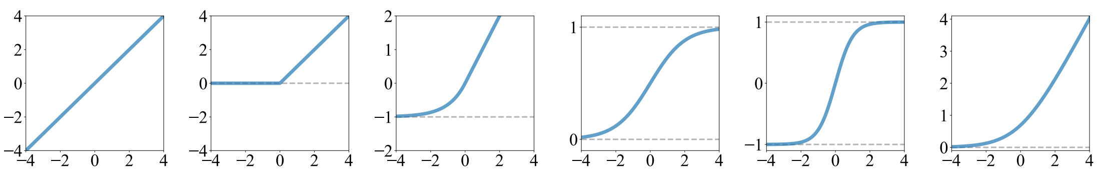

Figure 2.9: *Plots of the activation functions from Table 2.2. Left to right: linear, ReLU, ELU, sigmoid, hyperbolic tangent (tanh), and softplus.*

2.2.1.3 Convolutional Neural Networks

A very effective way of image feature extraction are *Convolutional Neural Networks* (CNNs). The idea was first introduced by Fukushima (1980) as a method for modelling hierarchical structures in image data. CNNs are now used extensively in image processing and computer vision (Yamashita, 2018), including the audio feature extraction from the spectrogram (Dörfler et al., 2017). The current state-of-the-art feature extraction system of Wang et al. (2019) is a convolutional neural network.

A convolutional layer takes an input image and processes it into a number of feature maps using a kernel convolution. The purpose of the kernel convolution is to compress the information of the image and to smooth-out the features.

The *convolution transform* (Hirschman and Widder, 1955) between two integrable functions f and g, written $f \star g$, is a function such that:

$$(f \star g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau,$$

and it is a way of combining, or "blending", two functions.

For discrete functions, the definition is similar:

$$(f \star g)(t) = \sum_{i=-\infty}^{\infty} f(i)g(i-t).$$

In modern neural networks, the 2−dimensional convolution C between a (black-and-white) image $M \in \mathbb{R}^{N\times M}$ and a kernel $k \in \mathbb{R}^{n\times m}$ is often computed as:

$$C(M,k)_{ij} = \sum_{a=0}^{m-1}\sum_{b=0}^{n-1} k_{ab}M_{i+a,j+b}, \quad \forall i \in \{0,\dots,N-1\}, j \in \{0,\dots,M-1\} \tag{2.2}$$

which is also known as the *cross-correlation* between M and k, since it differs from the convolution definition but serves a similar purpose of "blending" the image with the kernel.

An example of a single cross-correlation transform of a small 3×3 image and a 2×2

kernel with 1×1 stride is shown in ***Figure 2.10***. A stride is simply the distance between consecutive kernels. Since this particular kernel is positive, the output is positively-correlated with the input – in a sense that it contains high and low values in similar locations to the corresponding high/low values in the input. If the input was a larger image of an object, the output would still be recognizable.

It can be seen that several choices can be made with respect to the padding, which then determines the dimension of the output. If the cross-correlation is computed without any padding (known as "valid" padding), the dimension of the output is at most the dimension of the input; if the zero-padding is added (known as "full" padding), the dimension of the output is at least the dimension of the input. Another popular choice is to preserve the output dimension by computing the cross-correlation and only keeping the first N rows and M columns, which is known as "same" padding.

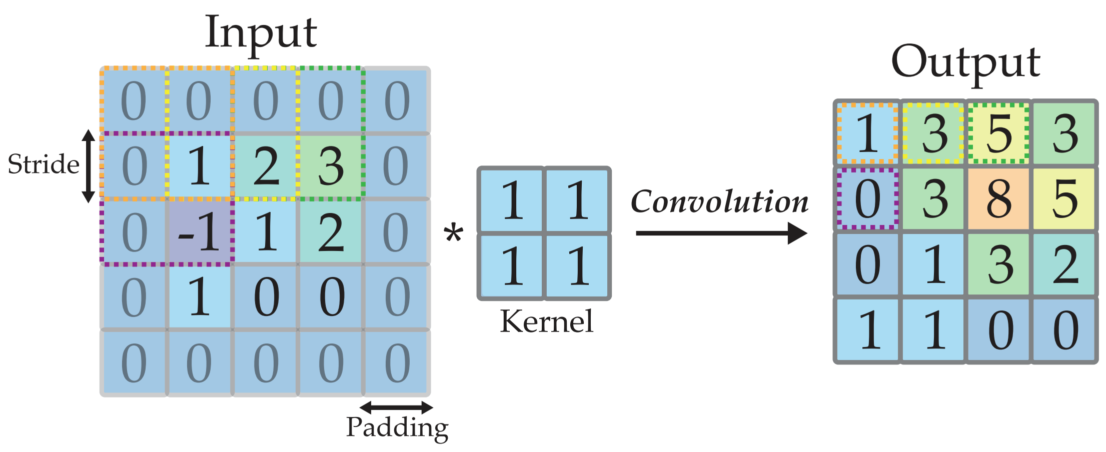

Figure 2.10: *Example of a convolution transformation of a 3×3 image with a 2×2 kernel of ones, and a 1×1 stride, and a full padding. Outlined in dashed squares are a few selected input features and their corresponding location in the output.*

The job of the neural network during training is to optimize the values of the kernel (weights) and one bias value, for each feature map. So if there are F feature maps, the number of trainable parameters is $(n \times m + 1) \times F$, where n and m are the dimensions of the kernel as before.

The number of feature maps in a convolutional layer plays a similar role as the number neurons in a dense layer – they control the number of representations an input image will have in the output, – while the kernel weights control what the representations will qualitatively look like.

In deep neural networks for image classification, the convolutional layers are typically increasing in the number of feature maps with each layer and decreasing in kernel size. For example, the AlexNet Krizhevsky et al. (2012) architecture has five convolutional layers – the first layer has a 11×11 kernel and 96 feature maps, while the last two layer have a 3×3 kernel and 384 and 256 feature maps respectively. This way, the represented features get more abstract as an input move through the layers.

One can deduce that the dimension of the output after a convolution with "same" padding and F feature maps is $N \times M \times F$, which gets large with increasing F. An effective way of reducing the dimensionality is by reducing N and M by means of a *pooling* operation.

Given an aggregating function f and an input $C \in \mathbb{R}^{N \times M \times F}$ (e.g. from a convolution), the pooling transform is defined as:

$$\begin{aligned} p(C,k)_{ijk} &= f_{\substack{a \in \{0,\dots,n-1\} \\ b \in \{0,\dots,m-1\}}} \left(C_{i+a,j+b,k} \right), \\ &\forall i \in \{0,\dots,N-1\}, j \in \{0,\dots,M-1\}, k \in \{0,\dots,F-1\}, \end{aligned} \tag{2.3}$$

which is just a convoluted (*ha!*) way of saying that the input is split into kernel-sized frames $h \in \mathbb{R}^{n \times m}$, and f is applied onto each frame. Typical choices of f include $f(h) = \max(h)$, known as "max-pooling"; and $f(h) = \text{mean}(h)$, known as "average-pooling". An example of both poolings is shown in ***Figure 2.11***.

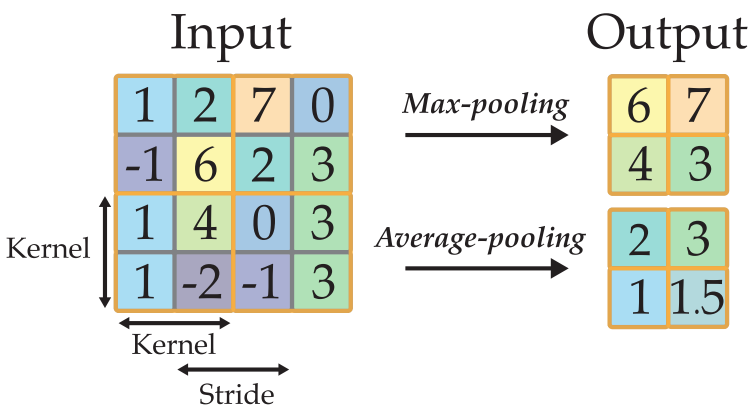

Figure 2.11: *Examples of a pooling layer with a* 2 × 2 *kernel and a* 2 × 2 *stride: max-pooling and average-pooling.*

Notice that in (2.3), the pooling is not done along the F dimension. If instead the pooling includes the feature maps dimension F, it is known as a "global" pooling. This is useful for completely eliminating the dimension F, leaving just a 2−dimensional output, and is sometimes used as the last layer of the convolutional network.

Alternatively, a flattening layer is frequently used as the final convolutional layer. The flattening layer simply reshapes a higher-dimensional object into a 1−dimensional vector. The 1−dimensional output of the convolutional network can then serve as the input to the fully-connected network.

An example of a complete convolutional network with two convolutional layers and a pooling layer is shown in ***Figure 2.12***. The output of this can then serve as the input x in the network from ***Figure 2.7***.

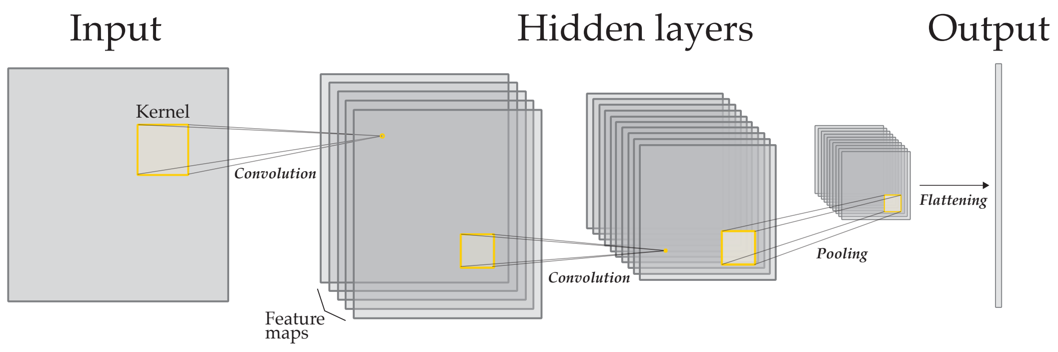

Figure 2.12: *A sketch of a 2-dimensional convolutional network consisting of two convolution layers and a pooling layer, followed by the flattening of the feature maps into a 1-dimensional vector output.*

2.2.2 Network training

The goal of network training is to select the weights and biases of the network, as well as all other hyperparameters, such that the loss is minimised.

This is efficiently done using backpropagation (Rumelhart et al., 1986), where the network adjusts the weights according to an *optimizer*. An outline of backpropagation is: at each training step, a random sample ("batch") is taken from the training set and passed through the model ("forward pass") to obtain predictions. A loss is computed, and the optimizer adjusts the weights of the layers in the reverse order during the backward pass.

The optimizer specifies the algorithm by which the weights are updated. The most basic one is the Stochastic Gradient Descent (SGD), and is defined as:

$$w_{\text{new}} = w - \eta \boldsymbol{\nabla} L(w; B)$$

where B is a randomly-sampled batch from the training data, w are the weights, L a minibatch loss function and η is the learning rate. The symbol $\boldsymbol{\nabla}$ represents the gradient of the loss function, which is taken with respect to the weights. The *minibatch loss* $L(w; B)$ is given by:

$$L(w, B) = \frac{1}{|B|} \sum_{(x,y)\in B} l(y, f_w(x))$$

where l is a loss function and $f_w(x)$ is the output after a forward pass through the model with weights w.

From the above it can be seen that in order to utilise backpropagation, a differentiable loss function needs to be used. For continuous data, the mean-squared error (MSE) or mean-absolute error (MAE) are commonly used. For categorical data, a cross-entropy loss is commonly used.

For validation, and to help prevent overfitting, the data is often split into several splits: the training, validation and testing splits. The model is trained on the training data and its performance is evaluated on the validation split in order to check if any adjustments need to be made to the model's structure. When comparing several models, their metrics are compared on the validation set. The final model performance is evaluated on the test set.

2.3 Sequence Alignment

So far, this chapter has focused on the necessary background into the feature extraction part of the audio-to-MIDI alignment problem. In this section, the methods for alignment of the extracted features are outlined.

Most audio-to-MIDI alignment research is focused on improving feature extraction, such as Müller et al. (2006), Ewert et al. (2009), Simonetta et al. (2021), etc. By choosing a common alignment algorithm, these methods hold the alignment methodology fixed. This makes cross-comparison between methods simpler.

The *Dynamic Time Warping* (Sakoe and Chiba, 1978) is the most commonly used algorithm for audio-to-MIDI alignment. The word "dynamic" here refers to the class of algorithms known as *dynamic programming*, which means that the algorithm is computed dynamically and the output is not determined in advance or in a one-shot manner. A small example of the DTW alignment between two 1−dimensional sequences is shown in ***Figure 2.13***.

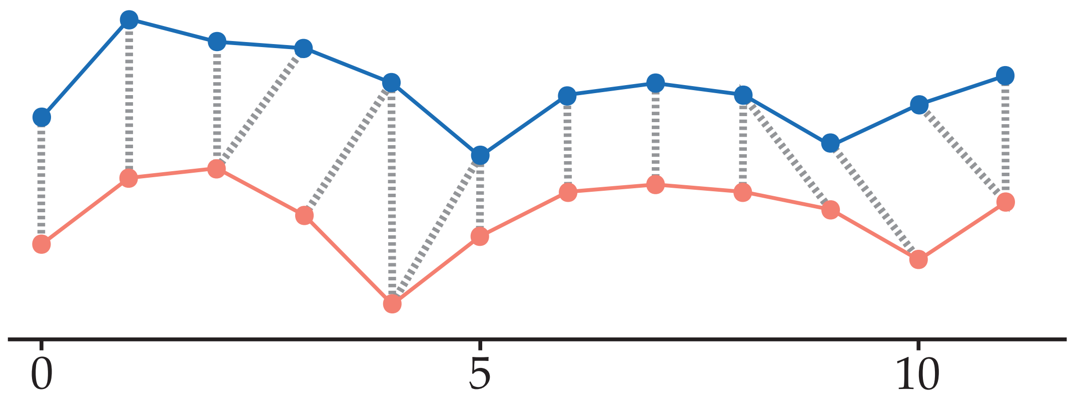

Figure 2.13: *An example of the DTW alignment between two* 1−*dimensional sequences. The alignment path is shown by the dashed lines.*

Given two matrices $X \in \mathbb{R}^{F \times N}$ and $Y \in \mathbb{R}^{F \times B}$, the dynamic time warping alignment is defined as the solution to following global minimization problem:

$$p, q = \underset{\substack{p \in \{0,\ldots,N-1\} \\ q \in \{0,\ldots,P-1\}}}{\operatorname{argmin}} \sum_{i=0}^{K} d(X[p[i]],\ Y[q[i]]) \tag{2.4}$$

where $K = \max\{N-1, P-1\}$, d is a distance function, $p,q \in \mathbb{Z}^K$ is the alignment path, and "$Z[k]$" is the notation for "kth column vector in the matrix Z".

An additional constraint that needs to be made to prevent degenerate alignments is that both p and q need to be monotonically increasing. Various ways have been proposed to achieve this, such as by Müller (2007). Sakoe and Chiba (1978) propose the following restrictions on p and q:

$$\text{Monotonicity \& Continuity} : (p[i-1], q[i-1]) = \begin{cases} (p[i], q[k]-1), & \text{or} \\ (p[i]-1, q[i]-1), & \text{or} \\ (p[i]-1, q[i]) \end{cases}$$

$$\text{Boundary} : \begin{cases} (p[0], q[0]) = (0, 0) \\ (p[K-1], q[K-1]) = (N-1, B-1) \end{cases}$$

These two conditions ensure that the alignment is "complete", in the sense that the entirety of X is aligned with the entirety of Y and that the alignment is monotonic and continuous.

The meaning of the vectors $p \in \mathbb{Z}^K$ and $q \in \mathbb{Z}^K$ is such that for any $i \in \{0, \ldots, K-1\}$, if $p_i = \tau$ and $q_i = \omega$ then the τth column vector in X is aligned with the ωth column vector in Y.

Note that (2.4) requires computing all pairwise distances between columns of X and columns of Y, which is computationally expensive for large N and P, but more efficient implementations exist which use dynamic programming – one of which is proposed by Sakoe and Chiba (1978). An even more efficient approximation is the *FastDTW* algorithm (Salvador and Chan, 2004).

An in-depth review of the dynamic time warping and its application to audio synchronization is given by Müller (2007).

3 Related Research

In this chapter, we outline the relevant works in the area of audio-to-MIDI alignment and their respective methodologies.

Audio-to-MIDI alignment methods operate a two-step procedure: feature extraction, followed by alignment of extracted sequences. In this case, the quality of the alignment will depend not only on the alignment technique, but also on the quality of the extracted features. When the performance of alignment in one medium (MIDI) depends on the quality of the feature extraction in another medium (audio), this is known as the *heterogeneity gap* (Cheng and Gu, 2021). The narrower the gap between the two modalities, the better the quality of the alignment will be.

In order to bridge the gap between the modalities, the extracted audio and MIDI features need to be in the same domain. Hence there are three possible modalities: (1) mapping the audio features into the MIDI domain, (2) mapping the MIDI features into the audio domain, and (3) mapping the audio and MIDI features into a common "middle-ground" domain.

The first, the MIDI domain, is a commonly used one, because the MIDI files provide a structured information about the musical score such as notes and their location. This is commonly done using automatic music transcription.

Many of the early methods relied on the alignment in the audio domain.

The third domain, also known as *cross-modal*, makes for a very effective audio-to-MIDI alignment, and the current state-of-the-art methods use this method.

In this chapter we will be discussing works that utilise each of these domains, as well as their advantages and drawbacks, along with the results. Throughout, the reader should pay attention to the overarching theme of "feature extraction followed by sequence alignment".

3.1 Audio Domain

In this section we will outline some audio-to-MIDI alignment works that utilise the feature extraction in the audio domain.

Arguably the first audio-to-MIDI alignment method was proposed by Turetsky and Ellis (2003), which featured alignment of audio and MIDI spectrograms. They have obtained a number of unaligned MIDI files audio pairs with the task of obtaining the alignment between the pairs. The outline of their methodology is shown in ***Figure 3.1***.

Since large collections of aligned pairs of audio and MIDI files did not exist at the time, their method of feature extraction had to be entirely computational. The features that they utilise are the spectrograms corresponding to the audio and MIDI files.

First, they converted the MIDI files into audio tracks by synthesizing them. Then they were left with the task of aligning two audio files (synthesized MIDI, and a real recording). For this purpose, they downsampled the audio tracks to 22050Hz, after which computed short-time discrete Fourier transforms of each, with frame lengths equal to 2048 samples (93ms), hop length 1024 samples (46ms), and a Hanning window function. After this, they removed all frequency bins above 2.8kHz on the basis that those frequencies are mainly overtones and don't relate greatly to the fundamental frequency.

In addition, they have added a small amount of white noise to the spectrograms in order to avoid infinite-vector similarity between zero-vectors of the spectrograms. Also, they have tried a number of processing techniques for the spectrogram, such as transforming spectrogram values (logarithmic and power transforms), appending the spectrograms with their first order differences as new features, and noise suppression.

Then, the cross-similarity matrix $D \in \mathbb{R}^{N \times P}$ was computed between the two spectrograms A, M, where $A \in \mathbb{R}^{B \times N}$ is the spectrogram of the audio and $M \in \mathbb{R}^{B \times P}$ is the spectrogram of the synthesized MIDI, B the number of frequencies and N, P the number of frames (vectors) in the matrices. The similarity matrix they used was introduced by Foote (1999), and is defined as follows:

$$D_{ij} = \frac{A_i^T \cdot M_j}{|A_i||M_j|}, \; i \in \{0, \ldots, N-1\}, j \in \{0, \ldots, P-1\}$$

where A_i is the ith vector in the audio spectrogram and M_j is the jth vector in the MIDI spectrogram.

The similarity score D_{ij} is between -1 and 1, such that a score of -1 represents very high dissimilarity, and 1 corresponds to a perfect similarity.

Next was the task of finding the optimal alignment path in the similarity matrix. This constitutes starting at the point $(0, 0)$ in the similarity matrix and computing the optimal (lowest-cost) path to the opposite corner $(N-1, P-1)$ of the matrix. For this, they chose Dynamic Time Warping (DTW), which is a Dynamic Programming (DP) algorithm. Dynamic Programming means that the path is computed dynamically, a step at a time, and not in a one-shot manner.

For the point (i, j) in the similarity matrix, the lowest-cost path is computed to all of the point's neighbours, plus the additional cost to get from the neighbour to the point. The cost is then given by:

$$C_{ij} = \max(D) - D_{ij}$$

which is inversely related to the similarity score, and makes the cost of most-similar frames to be zero, while all other frames then have a positive cost.

In a case of a perfect alignment between the audio and the MIDI, this path is diagonal. In the realistic case, this path consists of various diagonal segments of various angles, and sometimes horizontal and vertical jumps.

They have tested two definitions of the neighbouring points for the algorithm. The neighbours of the point (i, j) are defined as the set of points $\{(i+1, j+1), (i+2, j+1), (i+1, j+2)\}$, and if they also include the points $\{(i+1, j), (i, j+1)\}$ they called it an *unconstrained* algorithm, and if it excludes those points it is called a *greedy* algorithm. The morale for this is that a greedy algorithm always wants to move forward (in a diagonal), while an unconstrained one is capable of making vertical and horizontal steps, making it more cautious.

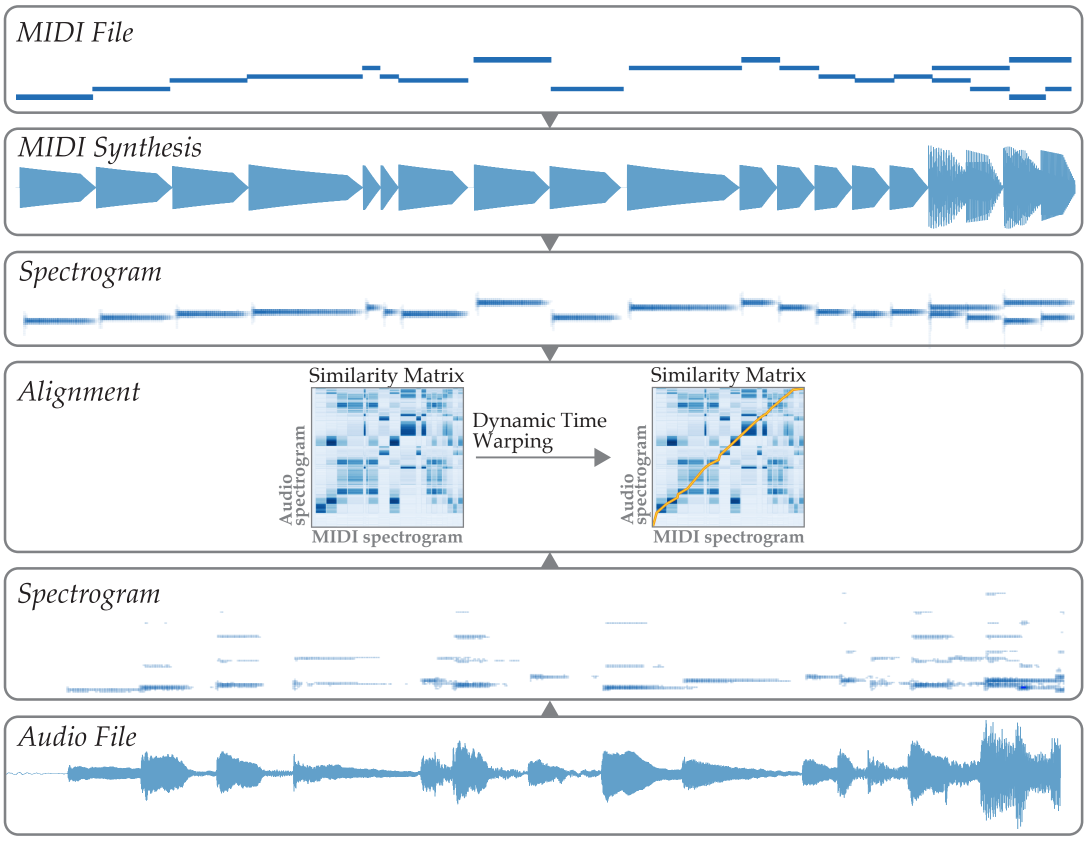

Figure 3.1: *Outline of an early audio-to-MIDI alignment method (Turetsky and Ellis, 2003), where the alignment takes place in the audio domain. A MIDI file is converted into an audio file by synthesis, after which its spectrogram is aligned with the spectrogram of the audio track using the Dynamic Time Warping algorithm. Darker regions in the similarity matrix correspond to high similarity. The optimal alignment path is annotated in yellow.*

From this, one can see that the most extreme case with which the greedy algorithm is able to deal with is when the audio and MIDI file tempos differ by a factor of 2, and if the tempo of the MIDI file is say 2.5 quicker than the audio track, it would not be able to find the correct neighbours because it would require vertical jumps. On the other hand, an unconstrained algorithm may be too cautious.

In order to get the best of both unconstrained and the greedy DTW algorithms, they employed a two-stage alignment procedure: first an unconstrained DTW is applied to find regions that align well, after which a 1.4s median filter is applied to find the segments where the steps are non-diagonal and a greedy algorithm is then used to realign those sections.

The authors applied their methodology onto 40 popular music songs and their publicly available MIDI file transcriptions. They have chosen 7 different spectrogram processing/augmentation methods, and 2 different alignments for each (first-stage only, and two-stages), giving possible 14 alignments for each song.

For the evaluation of each song, the authors have computed the 14 alignments and played back each of the aligned MIDIs and their corresponding audio tracks in stereo and evaluated the alignment on a subjective scale of 1 to 5, where 1 is "Could not align anything" and 5 is a perfect alignment. The score for the song is then chosen as the highest of all alignments for that song.

They have applied their methodology onto 40 popular music songs and obtained 27 (67.5%) perfect alignments, 4 (10%) good alignments, 3 (7.5%) average alignments, 2 (5%) poor alignments and 4 (10%) complete misalignments. As they concluded, 31 (78%) songs had alignments good enough to be used for audio transcription purposes.

At evaluation, the authors have found that many of the alignments did not work because due to the nature some MIDI files do not give a correct representation of their corresponding song. Some of the issues included missing (or skipped) large sections of the songs, incorrectly transcribed notes, or the wrong key signature (note pitches are offset by a constant).

3.2 MIDI Domain

In contrast to the previous section, where the alignment took place between the spectrograms of the audio file and synthesized MIDI file, the alignment here takes place between the MIDI features.

By 2012, large-scale datasets of aligned audio-MIDI pairs began to exist, the most prominent being the MAPS dataset (Emiya et al., 2010). It contains 270 full-length recordings made through 9 different recording condition, which makes it relatively diverse. This allows for a piano transcription model to be trained.

We will discuss the audio-to-score alignment using RNN-based automatic music transcription of Kwon et al. (2017), which is itself based on the AMT model of Böck and Schedl (2012). The outline of the methodology is shown in ***Figure 3.2***.

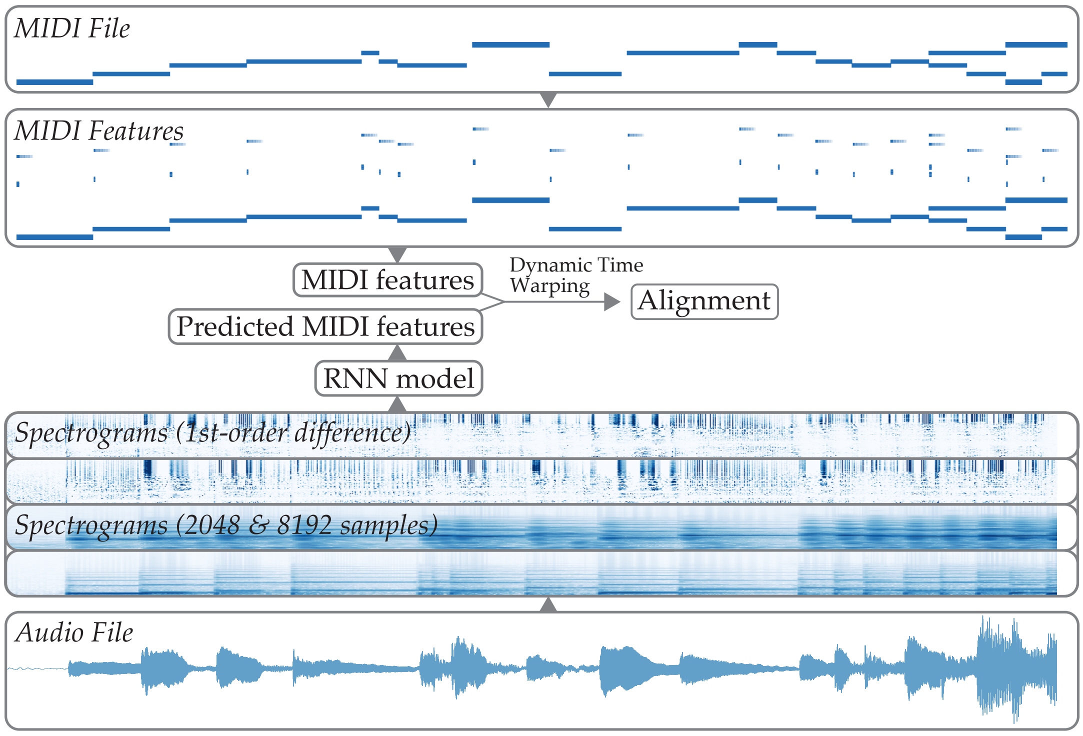

Figure 3.2: *Outline of an audio-to-MIDI alignment method from Kwon et al. (2017), where the alignment takes place in the MIDI domain. A (trained) neural network is used to predict the MIDI features given the audio spectrogram. After which, the predicted features are aligned with the MIDI features using Dynamic Time Warping. Two spectrograms have been concatenated, one using frame length* 2048 *samples, and one using* 8192 *samples. Two additional MIDI features are added: chroma onsets (MIDI features, middle, tiny) and decayed chroma onsets (MIDI features, top, blurry). For training, MIDI features are used as targets.*

Unlike in the Turetsky and Ellis (2003) methodology, where for each song we only had an audio recording and an unaligned MIDI file, in this case we have a MIDI file with its corresponding audio recording, where the audio recording perfectly matches the MIDI file. From the aligned MIDI file, one can of course also synthetically misalign the MIDI file and forget that it exists.

Broadly, the methodology is, as ever, in two stages: feature extraction and sequence alignment. The feature extraction here is happening in the audio domain, and very minimal processing is done to the MIDI file.

The authors follow the data processing steps of Böck and Schedl (2012). For the audio file, two different short-time DFT (equivalently, STFT) spectrograms are computed. One with a frame length of 2048 samples (46.4 ms at 44100Hz), the other with a frame length of 8192 samples (185.8 ms), a hop length of 441 samples (10 ms) and a Hamming window function. Phase information is omitted from the spectrograms by taking the absolute values to obtain the magnitude spectrogram.

The morale for taking two spectrograms is that the longer the frame length, the higher the frequency resolution, and the shorter the window the higher (more precise) the temporal resolution, so putting together two spectrograms combines the best of both worlds.

Then, each spectrograms receives a log-like compression as follows:

$$\widehat{S}_{ij} = \log_{10}\left[1000 \times S_{ij} + 1\right], \quad i \in \{0, \ldots, B\},\ j \in \{0, \ldots, N-1\}$$

where $S \in \mathbb{R}^{B \times N}$ is the original spectrogram and $\widehat{S}$ is the log-compressed spectrogram, B the number of features (frequencies) in the spectrogram and N is the temporal dimension (number of time steps). This maps the magnitudes into a suitable non-negative range, which Böck and Schedl (2012) found to be yielding the best preliminary results.

Each spectrogram is filtered using an overlapping triangular semitone filterbank filter. The filterbank is shown in ***Figure 3.3***. Since the DFT frequencies are linearly spaced, the purpose of this is to map them to their neighbouring piano notes. This step also reduces the number of dimensions of the spectrogram, from 1025 and 4097 (includes the 0Hz frequency) frequencies, to 81 and 102 frequency bins corresponding to the 2048 and 8192 sampled spectrograms. Hence the total number of dimensions after concatenation is 183, down from 5122.

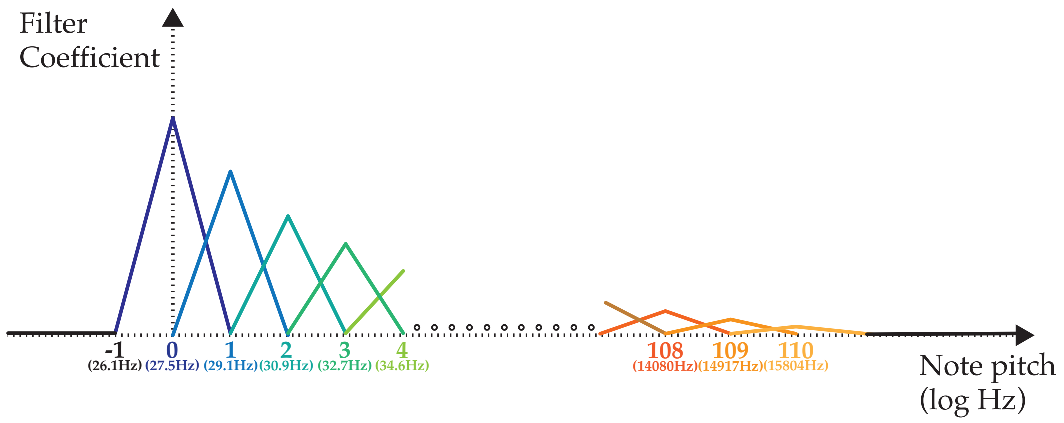

Figure 3.3: *Triangular overlapping semitone bank filter. Each frequency band in the spectrogram gets mapped to the nearest (by frequency) triangular filter and its magnitudes get multiplied by the height of the triangle (filter coefficient) above it. Even though the piano range is [0, 87], 23 additional bands up to note 110 are added to capture any higher overtones. The area below each triangle is normalized to* 1. *Note the x-axis is logarithmic in the frequency domain.*

Finally, the first-order differences of the spectrograms are calculated as follows:

$$\Delta\widehat{S}_{i,j} = \widehat{S}_{i,j} - \widehat{S}_{i,j-k}$$

where $k = 1$ for the 2048-sampled spectrogram and $k = 4$ for the 8192-sampled spectrogram. The purpose of this is to obtain information about the onsets (when the notes are pressed) in the audio. The spectrograms are then concatenated along the frequency axis to give the audio features spectrogram $A \in \mathbb{R}^{366\times N}$.

For the MIDI file processing, the piano-roll is calculated at the same resolution of 100 frames per second as the spectrograms. In addition, the 12-note chroma onsets are added.

The onsets of a piano roll matrix is itself a sparse binary piano roll matrix R^{onset} taking values 1 if a new note is played, and 0 otherwise:

$$R^{\text{onset}}_{i,j} = \begin{cases} 1, & \text{if } R_{i,j} > R_{i,j-1} \\ 0, & \text{otherwise} \end{cases}$$

The 12-note chroma labels are the musical equivalent of a modulo arithmetic, and it compresses the entire note spectrum onto just 12 distinct values. The 12-tone chroma is defined as a direct transformation of the note axis indices (i) of a piano roll:

$$i^{\text{chroma}} = (i - 3) \bmod 12$$

Following the methodology of Ewert et al. (2009), the authors also add decayed chroma onsets. This is done by copying every onset 10 times, multiplying by decaying weights: $(1, \sqrt{0.9}, \sqrt{0.8}, \ldots, \sqrt{0.1})$, thus transforming every onset into a decaying row vector of length 10.

Similar to the spectrograms, the 3 MIDI features piano roll, 12-note chroma onsets, and 12-note decaying chroma onsets are concatenated along the feature axis. The overall dimension of the MIDI features is 88 + 12 + 12 = 112. The MIDI features matrix is then padded with zeros such that it matches with the audio features spectrogram, and is denoted by $M \in \mathbb{R}^{112\times N}$.

The processed audio spectrograms and the MIDI features were then used to train two different neural networks: the first would predict the notes in the piano roll and in the 12-note chroma roll, while the second predicts just the 12-note chroma onsets. The authors use a $173 - 43 - 54$ split for the training, validation and the test sets respectively.

The model used by the authors was the Böck and Schedl (2012) model, which is a recurrent neural network (RNN) with bidirectional Long Short-Term Memory (LSTM) units.

The RNNs were first introduced by Hopfield (1982), with the idea that the neural networks should be able to have some form of "memory" of the data that it observes and to be able to put that data into some context. The memory units are made of recurrent connections in the hidden layers of the model. The regular RNNs were found to suffer from vanishing and exploding weights. The LSTM architecture (Hochreiter and Schmidhuber, 1997) solves this.

For evaluation, the authors synthetically misaligned the MIDI files by splitting the MIDI file into 20 chunks and altering the tempo of each chunk by ±30%. This is a standard way of testing the alignment quality and has also been previously used by Müller et al. (2006), Ewert et al. (2009) and Joder et al. (2011).

After this, the misaligned MIDI files are processed in the same way as the training features, and the audio features are used to compute the predicted MIDI features. Finally, the predicted MIDI features and the misaligned MIDI features are aligned using the Dynamic Time Warping algorithm.

For efficiency, the FastDTW (Salvador and Chan, 2004) variant of the DTW is used. The computational complexity of the FastDTW is at most $O(rN_{\max})$, where $N_{\max} = \max(N, P)$, and r is the radius parameter; while the complexity of the regular DTW is $O(NP)$. When $r = N$, the FastDTW is equivalent to the regular DTW. Kwon et al. (2017) use $r = 10$, which is a commonly used value for the audio-to-MIDI alignment purposes.

The authors use the *onset errors* as an accuracy metric, which is the absolute difference between the aligned onset and the true onset. They obtain an average mean error of 8.62 (median 5.57, s.d. 31.14), with 91.60% and 99.61% of the onsets within 10ms and 100ms thresholds respectively.

Unfortunately, the authors do not provide the precise details of the neural network models, which makes the work difficult to reproduce.

3.3 Cross-Modal Domain

Cross modal deep neural networks have proven to be very effective in the image feature extraction tasks such as facial identification (Sarfraz and Stiefelhagen, 2016), person identification (Zhu et al., 2019) and audio-visual speech classification (Cangea et al., 2017). Cross-modal neural networks operate on the principle of learning a *joint embedding space* between two disjoint modalities. In our case, the two modalities are MIDI and audio.

One particularly popular way of creating a joint embedding space are the *Siamese Neural Networks*, which were first proposed by Bromley et al. (1993). The name "Siamese" refers to a model consisting of a number of independent neural networks working in parallel – each having its own input – that are then joined at some point to produce outputs in a shared space. In order to compare the outputs, these networks typically use a *contrastive* loss function. The contrastive loss, first introduced by Hadsell et al. (2006), works on the principle of minimizing the embedding distance between similar inputs, and vice versa.

The work discussed in this section is an audio-to-MIDI alignment method using triplet pair by Wang et al. (2019), which features an example of a Siamese network. It is the state-of-the-art model for audio-to-MIDI alignment at the time of the writing. Their approach is itself based on an earlier work by Raffel and Ellis (2015). As before, we present the outline of their method in Figure 3.4.

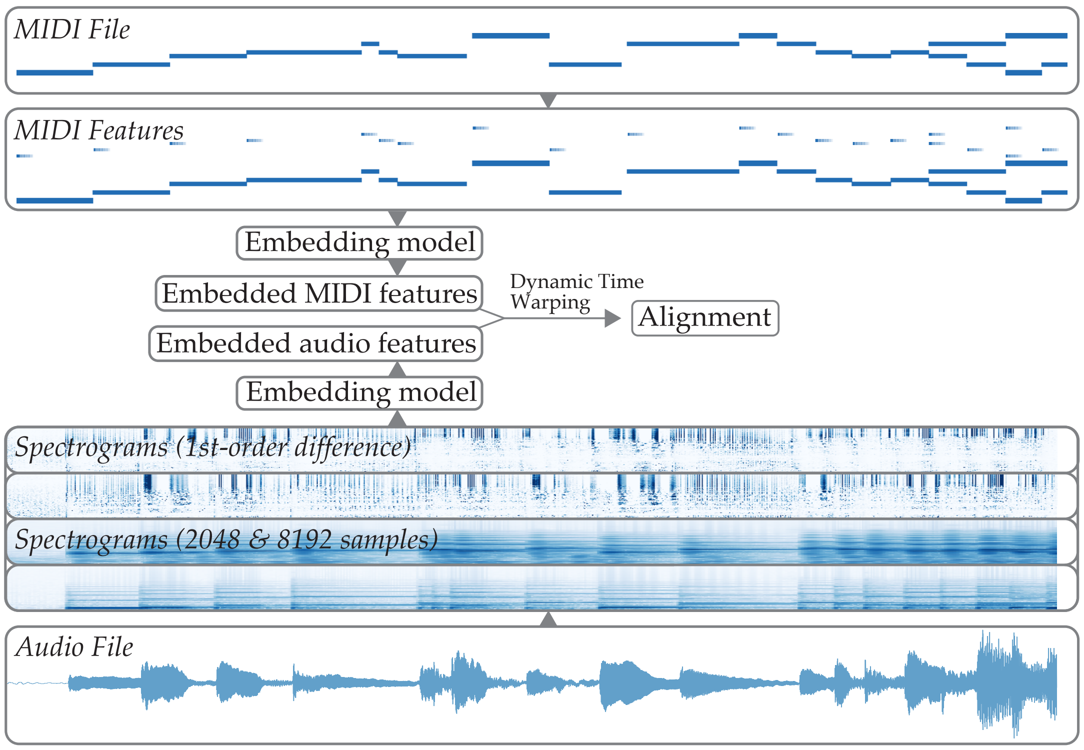

Figure 3.4: *Outline of a cross-modal audio-to-MIDI alignment method from Wang et al. (2019). The audio and MIDI processing steps are the same as in Kwon et al. (2017) shown in Figure 3.2. The difference is that here a single model learns both the audio and the MIDI embeddings. The embedded features are then aligned using Dynamic Time Warping, as before.*

The triplet loss is an idea that is useful for learning the joint embedding space. For each audio feature sequence (*Anchor*), 3 MIDI feature sequences are presented – a correct one (*Positive*), an incorrect one (*Negative*), and a partially correct one (*Overlapping*) – with the goal of finding the model that minimises the distance between the anchor, positive and overlapping embeddings; and simultaneously maximises the distance between the anchor and the negative embeddings. The overlapping sequence is a novelty of this framework, and is intended to capture the onset information.

Conveniently, the authors follow the same steps as Kwon et al. (2017) and Böck and Schedl (2012) for both audio and MIDI pre-processing, which we discussed in detail in section 3.2, with the only difference being that they do not include the 12-note chroma onsets.

Unlike Kwon et al. (2017), who used the properties of RNNs for sequential data to model the context of a spectrogram vector, the authors instead use context windows to model the context of a frame. Instead of an input consisting of a single vector, the input consists of a frame of length $L \in \mathbb{N}$ (odd integer): for time index $t \in \{0, \ldots, N-1\}$, the context frame is $t_L = \left\{t - \frac{L-1}{2}, \ldots, t + \frac{L-1}{2}\right\}$.

Let $M \in \mathbb{R}^{100\times N}$ and $A \in \mathbb{R}^{366\times N}$ be the processed MIDI and audio features matrices respectively as defined in section 3.2. For index t, the anchor, positive, negative and overlapping frames are defined as:

$$\begin{aligned}
\text{Anchor} : x_t^{\text{A}} &= A_{i,t_L}, \\
\text{Positive} : x_t^{\text{P}} &= M_{i,t_L}, \\
\text{Negative} : x_k^{\text{N}} &= M_{i,k_L}, \qquad |k-t| > \frac{L-1}{2} \\
\text{Overlapping} : x_l^{\text{O}} &= M_{i,l_L}, \quad 0 < |l-t| < \frac{L-1}{2}
\end{aligned}$$

where A_{i,t_L} denotes the matrix $A_{i,t}$ $\forall t \in t_L$, etc. In addition, the MIDI frames (positive, negative, overlapping) have to be distinct from each other, so the $|k-t|$ and $|l-t|$ conditions may be broken, for example when the notes are held for a long time. In any case, the overlapping frame should contain overlapping information with the positive frame. An example of a triplet selection is shown in ***Figure* 3.5**. The final inputs are $x^{\text{A}} \in \mathbb{R}^{L\times 366\times N}$, and $x^{\text{P}}, x^{\text{N}}, x^{\text{O}} \in \mathbb{R}^{L\times 100\times N}$ each.

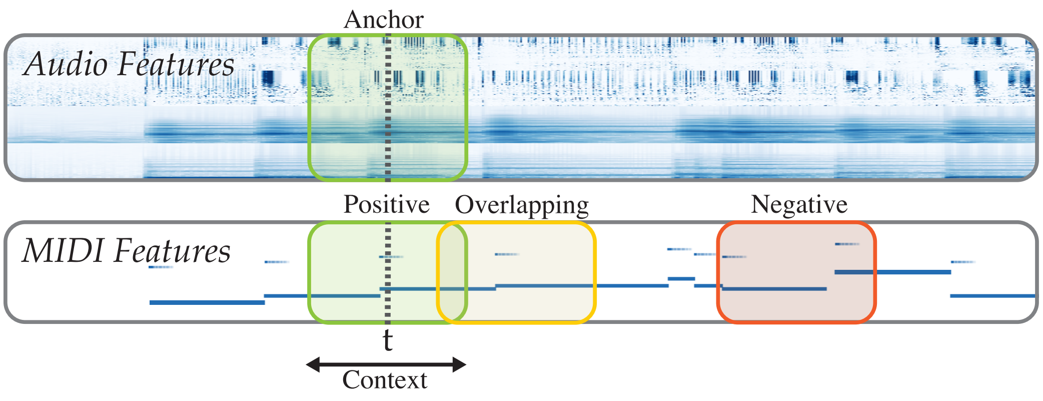

Figure 3.5: *Example of a selection of a single triplet for training for the Wang et al. (2019) triplet loss. Anchor and Positive represent the current frame in the audio and MIDI respectively, Overlapping overlaps with the Positive, while Negative does not. For optimal performance, the frames need to be chosen smartly, such that it is not too easy for the machine to learn the Positives and Negatives.*

For the model, the authors opted for two parallel convolutional neural networks: one takes the sequence of anchors as the input, the second takes in the sequences of positives, negatives and overlapping. The authors chose $L = 3$ for the context window length.

The models consist of 3 convolutional blocks, followed by 2 fully-connected dense layers.

Each convolution block consists of two convolutional layers and a max-pooling layer. All convolutional layers have a 3×3 kernel, 1×1 stride, and a ReLU activation, while max-pooling layers have 1×2 kernel and 1×2 stride, hence reducing the dimensionality of the frequency features but not the temporal dimension. The fully-connected blocks consist of two dense layers with 2048 units each and a ReLU activation. Finally, both models are connected to a single dense layer with 100 units and a hyperbolic tangent activation function.

For training, the following loss function was used:

$$\begin{aligned} Loss &= L^P + \alpha L^O \text{ where} \\ L^P &= \frac{1}{|B|} \sum \max \left(0,\ m + \left\| \widehat{y}^A - \widehat{y}^P \right\|_2 - \left\| \widehat{y}^A - \widehat{y}^N \right\|_2 \right) \\ L^O &= \frac{1}{|B|} \sum \max \left(0,\ m + \left\| \widehat{y}^A - \widehat{y}^O \right\|_2 - \left\| \widehat{y}^A - \widehat{y}^N \right\|_2 \right) \end{aligned}$$

where $|B|$ is the batch size, $\widehat{y}$ are the learned embeddings of x, α, m are hyperparameters chosen to be $\alpha = 0.8$, $m = 1$, and max() is an element-wise maximum function.

For evaluation, the authors used the same method as Kwon et al. (2017) by splitting each MIDI tracks into 20 sections and randomly multiplying the tempos by a random factor in the range $[0.7, 1.3]$.

The authors used the same evaluation metrics as Kwon et al. (2017) and report a mean onset error of 6.46ms (median 4.65ms, S.D. 11.11), with 87.26% and 99.92% of the onsets correctly aligned within 10ms and 100ms thresholds, which is an improvement on the results of Kwon et al. (2017).

While attempting to replicate their results, we have found our implementation of their models to be rapidly overfitting onto the training data. This could be due to the choice of the triplets. The authors do not provide enough information about their particular triplet selection method, and while we followed their methodology we were not successful in reproducing their results.

4 Methodology

In this chapter the proposed methodology, including the dataset, all pre-processing, training and evaluation steps is discussed.

Broadly speaking, the method consists of a special type of MIDI processing followed by a deep convolutional regression model, and an alignment in the MIDI domain. The intuition behind the proposed method is to make the alignment robust to errors in the performance.

First, two motivating examples. Consider the case of a standard AMT model that predicts which of the 88 notes are played at any time. Let A^{Score}, A^{Played}, $A^{\text{Predicted1}}$, $A^{\text{Predicted2}}$ be the vectors of the notes intended, notes played, and two competing predictions respectively. The following case provides an illustration into the drawbacks of the typical classification AMT model:

$$A^{\text{Score}} = \begin{bmatrix}1\\0\\0\\\vdots\\0\\0\end{bmatrix},\quad A^{\text{Played}} = \begin{bmatrix}0\\1\\0\\\vdots\\0\\0\end{bmatrix},\quad A^{\text{Predicted1}} = \begin{bmatrix}0\\0\\1\\\vdots\\0\\0\end{bmatrix},\quad A^{\text{Predicted2}} = \begin{bmatrix}0\\0\\0\\\vdots\\0\\1\end{bmatrix}$$

In this case, the performer has made a mistake by playing a wrong note, but at the model has also made a wrong prediction. By all standard metrics, the loss between the two competing predictions is the same, for example:

$$\sum_{i=0}^{87}\left|A_i^{\text{Played}} - A_i^{\text{Predicted1}}\right| = 2 = \sum_{i=0}^{87}\left|A_i^{\text{Played}} - A_i^{\text{Predicted2}}\right|$$

However, note that the note written in the score represented by A^{Score} is the 0th note, the note played is the 1st note, and the two predictions are note 2 and note 87. It should be easy to see that the first prediction is a lot closer to the note that was played than the second.

In the context of melodies, consider the following example, using a condensed piano roll notation:

$$A^{\text{Score}} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} A^{\text{Played}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} A^{\text{Predicted}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Here the model has made a correct prediction. The problem is that the recording is off by one note, which can happen if the piano is tuned in a different key, or the pianist is reading the notes wrong. It is also unaligned with the recording. When it comes to the alignment of A^{Score} and $A^{\text{Predicted}}$, from (2.4) it can be deduced that the DTW will always output a degenerate (identity) alignment because the distances between the pairs of columns of $A^{\text{Predicted}}$ and A^{Score} are all the same. If the indices are instead used, the above can be written as $A^{\text{Score}} = (0, 0, 1, 0, 1, 1)$ and $A^{\text{Predicted}} = (1, 1, 2, 1, 1, 2)$, then the alignment (in the context of a larger piece) has a higher likelihood of being found.

The proposed idea is to switch from the classification problem of the AMTs to a regression problem by extracting the information from the MIDI file in a specified way such that it forms a number of $1-$dimensional sequences where the values are continuous in the interval $[0, 87]$, which correspond to the pitches of the notes being played. This type of transformation induces a similarity measure onto the features whereby two notes close together on the piano keyboard remain close together in the feature space. Next, train a deep convolutional neural network using audio features as inputs and proposed MIDI features as targets.

At inference time, process the unaligned MIDI file in the same way as during training, and compute the alignment between the unaligned features and the predicted features.

An outline of our proposed method is shown in ***Figure 4.1***, and the outline for MIDI processing ***Figure 4.2***.

4.1 MIDI Processing

Let $R \in \mathbb{R}^{88 \times P}$ be the MIDI piano roll as before. For time $t \in \{0, \dots, P-1\}$, we define the *aggregate feature* vector as:

$$F_t = \begin{bmatrix} g_1(I_t) \\ \vdots \\ g_n(I_t) \end{bmatrix}, \quad I_t = \{i \mid R_{i,t} > 0\}$$

where n is the desired MIDI feature size and $g_1, \dots, g_n$ are some aggregating functions. I_t denotes the indices (equivalently, note pitches) of the notes that are being played at time t.

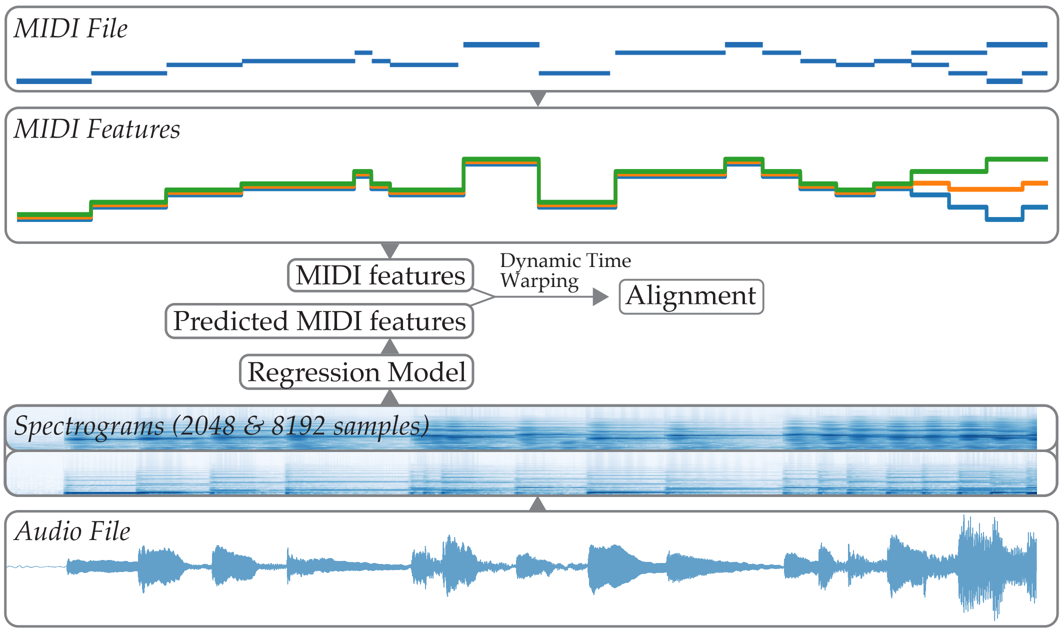

Figure 4.1: *An outline of the proposed methodology. MIDI features are aggregated to form several* 1−*dimensional sequences, in this example three sequences.*

For this project, we choose $n = 3$ and the following aggregating functions:

$$g_1(x) = \max(x)$$

$$g_2(x) = \text{mean}(x) = \frac{1}{|x|}\sum_{k=1}^{|x|} x_k$$

$$g_3(x) = \min(x)$$

where $|x|$ denotes the cardinality of a set x. Additionally, we see that the values of the functions are not defined when $x = \varnothing$, i.e. when no notes are pressed. In that case, we extend the preceding notes to cover the empty space. This seems to be the most reasonable way of dealing with empty notes – since excluding them from the data would lead to a biased model that would be unable to deal with silent regions. A drawback of this is that the model might mistakenly learn pick up silent sections as music and predict wildly incorrect notes in them, which could inflate the loss during training and hinder convergence. But at the time of the writing, no other method for dealing with this has been proposed.

An example of the proposed MIDI features is shown in ***Figure 4.2***.

Working with MIDI features in this form might have several potential benefits. The classical feature extraction models, such as the ones we discussed in the previous chapter

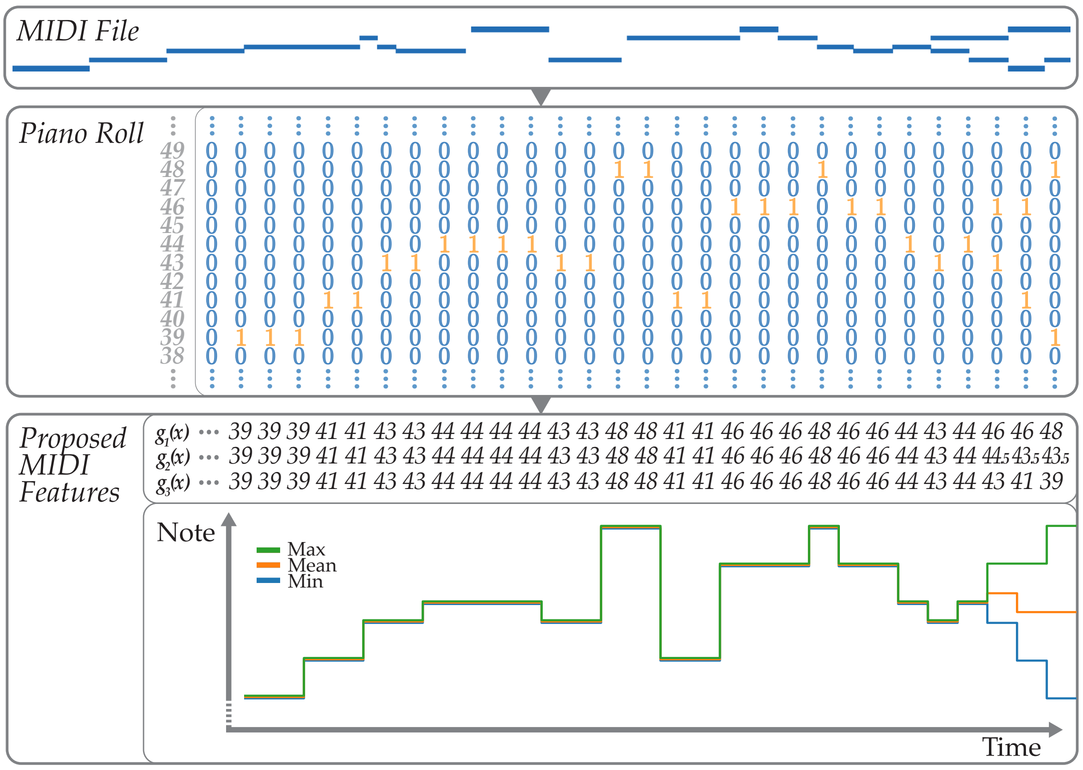
MIDI File
Piano Roll
Proposed MIDI Features
g1(x) ··· 39 39 39 41 41 43 43 44 44 44 44 43 43 48 48 41 41 46 46 46 48 46 46 44 43 44 46 46 48
g2(x) ··· 39 39 39 41 41 43 43 44 44 44 44 43 43 48 48 41 41 46 46 46 48 46 46 44 43 44 44.5 43.5 43.5
g3(x) ··· 39 39 39 41 41 43 43 44 44 44 44 43 43 48 48 41 41 46 46 46 48 46 46 44 43 44 43 41 39
Note
Max
Mean
Min
Time

Figure 4.2: *Example of the proposed MIDI feature aggregation. At each time point, the features are selected using aggregating functions g: in our case these are g_1 = max (highest note index), g_2 = mean (mean note index) and g_3 = min (lowest note index).*

– Kwon et al. (2017) and Wang et al. (2019) – as well as all existing AMT models, utilise discrete feature bins. While their models may implicitly learn that some notes are located close together in the note space (e.g. piano keyboard), the space is not explicitly defined with regards to their feature vectors. In other words, the classical models treat elements of the feature vectors as independent elements. A problem might then arise if the audio recording contains a mistake, such as a wrong note, the model would not be able to quantify the degree of "wrongness" of the note. By framing the classification problem as a regression problem, we should be able to quantify such performance error using the standard methods such as MSE and MAE.

On the other hand, it can be seen that this method of processing the MIDI file removes all information about the onsets and it would not be able to deal with musical sections where the same notes are being repeated. Our MIDI features for the repeated notes are equal to those of the held notes. Hence we theorize that our method is best suited to musical pieces where the top and bottom notes do not feature many consecutive repeated notes.

4.2 Audio Processing

With regards to audio pre-processing, we follow the methodology of Böck and Schedl (2012), which was also used in the aforementioned works of Kwon et al. (2017) and Wang et al. (2019) for their audio pre-processing. More precisely, we compute 2 short-time DFT spectrograms – one with frame length of 2048 samples, another with frame length 8192 samples – with a Hamming window and a 441 (10ms) hop length. After this we apply log-like compression to each value S in the spectrograms:

$$\text{LogCompression}(S) = \log_{10}\left[S \times 1000 + 1\right],$$

and then filter the frequencies using a triangular semitone filterbank, as shown in ***Figure 3.3***.

Since we do not propose modelling the onset features, we do not compute the first-order differences of the 2048 and 8192 sample spectrograms, yielding the overall audio feature dimension of 183.

In addition, we experiment by using a short-time CQT spectrograms for our audio features. We use 36 frequency bins per octave, which is equivalent to 3 bins for each note, up to the note 112. This yields 336 frequency bins in total.

Finally, we also investigate the hybrid approach using both the CQT and DFT features. As we discussed in section 2.1.5, the DFT has linearly spaced frequency bins, therefore its coverage differs between the low and the high frequencies – the coverage is sparser in the low frequencies and denser in the highs. Therefore we can expect the CQT to outperform DFT on the sections of music where the low-frequency notes are playing. For this purpose, we compute the CQT of the lowest 39 notes at a resolution of 3 bins per note, and augment it with the DFT audio features. This yields $183 + 39 \times 3 = 300$ features in total. We chose 39 as a cut-off because that note is the middle-C and is located exactly half-way between the bass clef and the treble clef in the piano notation. We could have included the entire CQT alongside the DFT, but that would lead to 519 features, which is more computationally expensive to process and would defeat the purpose of a hybrid approach.

Since the DFT is log-compressed, and the CQT is in the decibel units, it is a good idea to put the two on the same scale. The decibels are a logarithmic scale, so we should scale the CQT by a constant factor. We use the following scaling:

$$\text{CQT}_{\text{scaled}} = \frac{(\text{CQT} - \min(\text{CQT}))}{\max(\text{CQT}) - \min(\text{CQT})} \times \max(\text{DFT})$$

which puts the magnitudes of the CQT onto the range $[0, \max(\text{DFT})]$, which is equal to the range of the DFT. Alternatively we could have mapped both the DFT and the CQT to $[0, 1]$, but mapping the DFT spectrogram onto a fixed range is not commonly done.

4.3 Proposed Models

In this section, the audio spectrogram A is transposed such that $A \in \mathbb{R}^{N \times F}$. This is because the sample size is used as the first dimension during training. For all other purposes, $A \in \mathbb{R}^{F \times N}$.

The proposed model needs to be able to take the audio features as an input and give the predicted MIDI sequences and an output.

We propose a deep convolutional neural network that has similar architecture to the model of Wang et al. (2019). Similar to them, we use context frames of time length 3, such that each audio frame has dimension $3 \times F$, where F are the number of features. Starting with their model, we have increased the depth of the convolutional layers as well as the fully-connected layers for as long as the validation metrics were improving.

The outline of the proposed models is shown in ***Figure 4.3***. Our proposed model consists of 6 convolutional blocks of increasing feature maps, followed by a fully-connected dense network.

Each convolutional block consists of two convolutional layers, followed by a max-pooling layers. The convolutional layers have kernel $3 \times k$, 1×1 stride, ReLU activation and padding such that the input and output shapes are preserved. For model 1, k is fixed at $k = 3$, which is the same kernel as used by Wang et al. (2019). For the other two models we experiment by using kernels with a larger feature dimension. This way we should be able to capture more complex features such as chords and overtones.

The fully-connected dense network consists of two dense layers with 1024 and 512 neurons respectively, followed by a fully-connected block. The block consists of a dense layer of 128 neurons followed by 5 layers of 88 neurons each.

To improve the stability of the model as well as to make the convergence faster during training, we add batch normalization layer between every layer, with the exception that the max-pooling layer precedes batch normalization. Whereas Wang et al. (2019) place batch normalization before max-pooling layers, Ioffe and Szegedy (2015) recommend performing max-pooling prior to batch normalization.

We have also experimented with using drop-out layers instead of batch normalization in the fully-connected network, but preliminary results showed that batch normalization performs slightly better for prediction.

We choose to model each of the 3 proposed features $g_1(x), g_2(x), g_3(x)$ independently. The model weights are computed for each of these features. So for each audio frame, the output of each model is a single number $\hat{y} \in \mathbb{R}$. Alternatively, we could have allowed each model to have 3 outputs, which would have reduced the training time by a factor of 3. However since the number of outputs is very small, the computational cost allows us to compute the models separately. It also allows for simpler diagnostics of the models' performance on each feature.

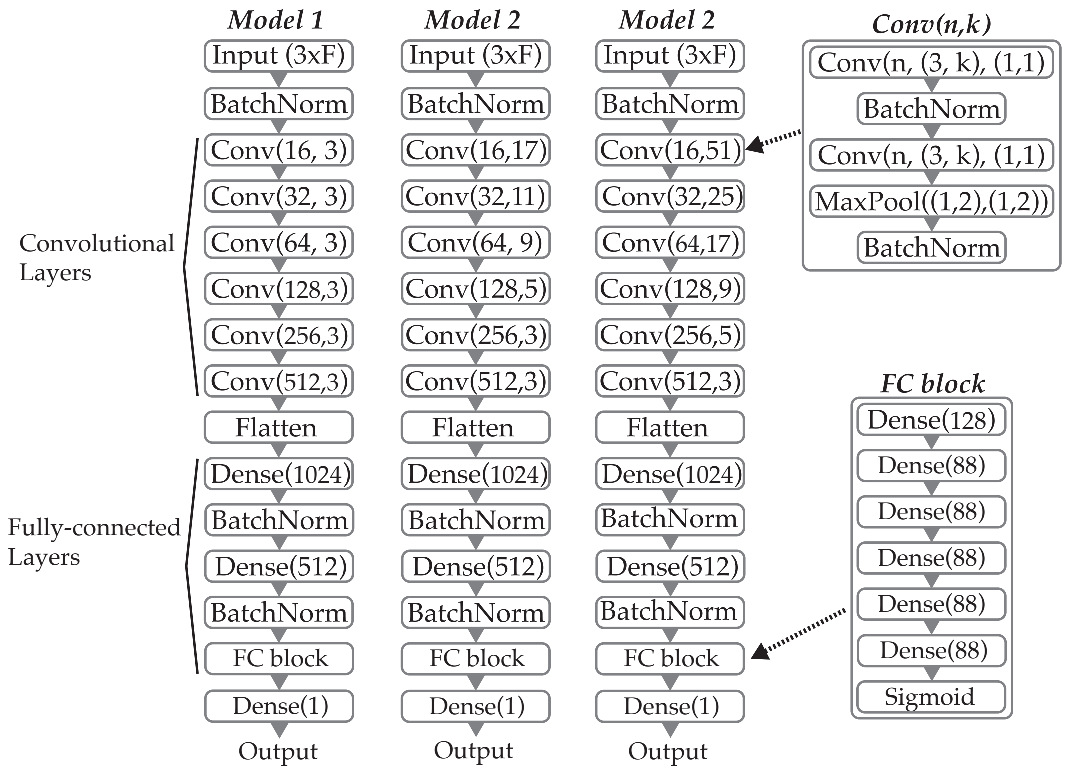

Figure 4.3: *An outline of the* 3 *proposed deep convolutional models. The inputs have shape* $3 \times F$*, where F is the number of features. Model* 1 *is based on the CNN of Wang et al. (2019), but with an increased depth of the network. Models* 2 *and* 3 *are based on Model* 1*, but with much taller kernels. Convolutional layers have n feature maps,* $3 \times k$ *kernels and* 1×1 *strides. Max pooling layers have* 1×2 *kernels and strides. Every layer has a ReLU activation function apart from the final dense layers of the models and the FC blocks. Every layer is followed by a Batch Normalization layer.*

4.4 Dataset

In order to train our model for the MIDI feature prediction task, a relevant dataset has to contain a large number of audio files with their corresponding perfectly matching MIDI files. We have several choices here.

First, there is the MAPS dataset (Emiya et al., 2010), which was used by both Kwon et al. (2017) and Wang et al. (2019). It contains 270 classical piano recordings across 9 recording conditions, 7 of which are the various synthetic pianos and 2 recording conditions feature a real Disklavier piano. As an aside, the Yamaha Disklavier is a type of reproducing piano, similar to the old reproducing pianos mentioned in section 2.0.2.

One of the newer datasets is the MAESTRO (Hawthorne et al., 2019), which features

1276 full-length aligned pairs of audio and MIDI files. This is a much larger dataset than MAPS, and it has been used by Simonetta et al. (2021).

In order to make the comparison between the proposed method and the aforementioned methods simpler, we choose to work with the MAPS dataset.

In order to speed up the training, we pre-process all pairs of audio and MIDI files into two large *NumPy* (Harris et al., 2020) arrays one contains all audio features, the other all MIDI features. When processing each pair of audio and MIDI files, the processed features are ensured to start and end at matching times, and he array are then cut to have matching length.

To process the MIDI files, we use the *pretty_midi* Python library, created by Raffel and Ellis (2014). In particular, we make use of its methods for MIDI to piano roll conversion and tempo distortion.

For computing the audio spectrograms using CQTs and the DFTs, we use the *librosa* Python library (McFee et al., 2022). It allows for an efficient spectrogram computation given the chosen parameters: sample rate, frame length, hop length, window type, etc.

4.5 Training

The total sample size of the training set comes out to 4,084,684 vectors.

The network is trained on the Imperial College's nvidia4 GPU cluster, using an Nvidia RTX 2080Ti graphics card. In order to obtain the utilise the most of the GPU's 11GB of memory, a batch size of 2048 was used, which is the largest batch size the GPU is capable of processing under our models.

This batch size yields 1994 training steps per epoch. Following Wang et al. (2019), we also use 50,000 training steps, which is approximately 6 epochs.

Similar to Wang et al. (2019), the Adam optimizer (Kingma and Ba, 2014) with a decaying learning rate was used. Since the learning rate is a hyper-parameter that depends on the model and the data, after performing preliminary training we chose the learning rate for each epoch to be $\{5\times10^{-3},\ 1\times10^{-3},\ 1\times10^{-4},\ 1\times10^{-5},\ 5\times10^{-6},\ 1\times10^{-6}\}$.

The training was done using the Python implementation of TensorFlow (Abadi et al., 2015), version 2.7.0.

4.6 Alignment

For the alignment task, the FastDTW implementation of the Dynamic Time Warping (DTW) algorithm is used. This way, the alignment algorithm is exactly the same as the methods of (Kwon et al., 2017) and Wang et al. (2019), and therefore the feature extraction models can be compared effectively.

Given a processed audio spectrogram $A \in \mathbb{R}^{F\times N}$, the predicted MIDI features $\widehat{M} \in \mathbb{R}^{3\times N}$

are computed by performing a forward pass through the model with the processed audio spectrogram serving as the input.

Next, given an unrelated MIDI file of the same piece of music, the aggregated MIDI features $\widetilde{M} \in \mathbb{R}^{3\times B}$ are computed using the same aggregating functions g_1, g_2, g_3 as during training.

Finally, the alignment is computed between $\widehat{M}$ and $\widetilde{M}$ using FastDTW. Following the common methodology, the radius parameter r is set to 10.

4.7 Evaluation

The evaluation follows the methodology of Kwon et al. (2017) and Wang et al. (2019). It is outlined in detail in this section.

In order to evaluate the alignment accuracy, an additional MIDI file is required that is not aligned with the audio. This is done synthetically by taking an aligned MIDI file, splitting it into 20 segments of equal time, and multiplying the tempo of each segment by some random number $u_i \in \mathcal{U}[0.7, 1.3]\ \forall i \in \{0, \ldots, 19\}$. This way the MIDI file is misaligned from the audio, but is still in a reasonable distortion range that it is recognizable. The alignment between the distorted MIDI and the predicted MIDI features is then calculated using section 4.6.

To refresh the memory of section 2.3, the alignment between two matrices $\widehat{M}$ and $\widetilde{M}$ is given by p,q such that $\widehat{M}[p[i]]$ is aligned with $\widetilde{M}[q[i]]$. Since the columns of both matrices have resolution of 10 ms, both p and q have resolution of 10 ms, and p contains the times corresponding to the audio features while q contains the times of the MIDI features. Therefore, the MIDI file can be re-aligned to the audio by mapping its times q to new times p, and linearly interpolating all the points in between. The *pretty_midi* library provides the functionality for this.

Next, there are two distinct evaluations that are performed: the feature extraction evaluation and the alignment evaluation. The first assesses the prediction performance of the models, while the second evaluates how far the aligned notes are from the correct ones.

For feature extraction, the metrics used are the mean-squared error (MSE), the mean absolute error (MAE) and the accuracy. They are defined as:

$$\begin{aligned}
\text{MSE}(\hat{y}, y) &= \frac{1}{N}\sum_{i=1}^{N}\left(\hat{y}_i - y_i\right)^2 \\
\text{MAE}(\hat{y}, y) &= \frac{1}{N}\sum_{i=1}^{N}\left|\hat{y}_i - y_i\right| \\
\text{Accuracy}(\hat{y}, y) &= \frac{1}{N}\sum_{i=1}^{N}\mathbb{I}\left[\ \left|\hat{y}_i - y_i\right| < 1\ \right]
\end{aligned}$$

where $\hat{y}$ are the predicted MIDI features, y are the true aggregated MIDI features and $\mathbb{I}$ is an indicator function. The first two are metrics are classical, while the accuracy is defined such that it measures the proportion of the predictions that are within 1 note pitch of the true values.

For alignment, the metrics used are the mean, median and standard deviations of the absolute onset errors. The absolute onset error is defined as:

$$AOE(\hat{p}_i, p_i) = \left|\hat{p}_i - p_i\right|$$

where p_i is the true onset time and $\hat{p}_i$ is the onset time of the same note in the aligned MIDI file. The evaluation is repeated on all pairs of audio and MIDI files, and the statistics are computed on the entire collection of onset errors.

Additionally, the morale of the proposed method is to make the alignment robust to errors in the performance. This is evaluated using simulated errors. Since the MIDI files are easier to adjust than audio files, the errors were simulated by adding Gaussian noise to the misaligned MIDI targets $\widetilde{M} \in \mathbb{R}^{3\times B}$ at each time t in the following way:

$$\widetilde{M}_{it}^{\text{Noise}} = \widetilde{M}_{it} + Z_{it} \quad \forall i \in \{0,1,2\},\ t \in \{0,\ldots,B-1\} \tag{4.1}$$

$$\text{where} \quad Z_{it} \sim N\left(0, \sigma^2\right). \tag{4.2}$$

The alignment evaluation is repeated for increasing values of σ until the alignment is deteriorated significantly.

This type of error simulation is simple but may not be realistic, since it implies that the errors happen at very high frequency, are uncorrelated, and have mean 0. Other simulation methods are possible, such as simulating correlated noise (Delisle et al., 2020).

5 Results

In this chapter, the proposed models are compared according to their feature extraction performance, after which the best model is selected and its alignment performance is evaluated. The results are split by aggregated MIDI feature and by audio spectrogram type. The feature extraction performance is measured in the deviation of the predicted note from the correct note.

For the best model, the alignment performance is evaluated.

Finally, in order to gain more insight into when the proposed method performs well and poorly, the alignment is closely examined across different sections of music: the music that is very well aligned, the music that is poorly aligned, and the music that is completely misaligned.

5.1 Feature Extraction Performance

Three different models, defined in section 4.3, were compared for their quality of predictions of the MIDI features. In addition, three different audio processing methods were compared: the DFT, CQT and a hybrid between the DFT and CQT.

The complete validation results for feature extraction are shown in ***Table 5.1***.

From the metrics of the combined features, it can be seen that the CQT pre-processing gives the best results by all three metrics compared to DFT and hybrid. Additionally, Model 1 yields the best performance for the combined features by all three metrics.

When looking at the features in isolation, the best metric for the highest note feature (max), the hybrid audio features yield the best performance under Model 1 (MSE: Hybrid 57.9, CQT 62.9, DFT 60.1). In fact the CQT gives the worst prediction results for the highest notes of the three processing approaches. This might be because some synthesized audio recordings in the dataset have very prominent overtones in the high frequencies.

For the lowest and mean notes, the CQT outperforms DFT and the hybrid approach.

Overall, the best obtained MAE for the combined features (under Model 1 and CQT spectrograms) is 3.39, which suggests that on average the predictions are within 3.39 notes away from the true notes. For the individual features, the MAE are 2.33, 3.39, 4.45 – corresponding to the lowest, mean and highest notes respectively.

	Feature g	Metric	Model 1	Model 2	Model 3
DFT	Max	MSE	60.1	62.0	63.3
		MAE	4.26	4.51	4.4
		Accuracy	34.5%	28.4%	28.4%
	Mean	MSE	28.0	28.0	29.2
		MAE	3.46	3.55	3.63
		Accuracy	25.4%	23.7%	23.3%
	Min	MSE	40.1	41.4	43.1
		MAE	3.24	3.53	3.55
		Accuracy	45.1%	37.7%	41.1%
	Combined	MSE	42.7	43.8	45.2
		MAE	3.66	3.86	3.93
		Accuracy	35.0%	29.9%	30.9%
CQT	Max	MSE	62.9	61.2	64.3
		MAE	4.45	4.66	4.74
		Accuracy	38.0%	28.7%	30.3%
	Mean	MSE	26.8	27.8	29.8
		MAE	3.39	3.48	3.74
		Accuracy	26.6%	25.0%	23.9%
	Min	MSE	29.8	36.0	36.4
		MAE	2.33	2.79	2.88
		Accuracy	64.6%	54.7%	53.5%
	Combined	MSE	**39.8**	41.7	42.5
		MAE	**3.39**	3.64	3.75
		Accuracy	**43.1%**	36.1%	35.9%
Hybrid	Max	MSE	57.9	60.9	66.9
		MAE	4.22	4.68	4.96
		Accuracy	36.0%	25.1%	23.4%
	Mean	MSE	28.4	28.4	29.9
		MAE	3.49	3.58	3.71
		Accuracy	25.5%	23.2%	23.1%
	Min	MSE	34.3	38.7	41.0
		MAE	2.72	3.20	3.23
		Accuracy	58.7%	46.7%	44.8%
	Combined	MSE	40.2	42.67	45.0
		MAE	3.48	3.82	4.14
		Accuracy	40.0%	31.7%	30.1%

Table 5.1: *Feature extraction performance of the three models broken down by audio pre-processing method (DFT, CQT, hybrid), feature (max, mean, min, combined) and validation metric (MSE, MAE, accuracy). Model 1 with CQT pre-processing is the best model by the combined metrics. The MSE was used as the loss for training the models. Best combined metrics are underlined.*

5.2 Alignment Performance

The test set alignment results are shown in ***Table 5.2***. The last column shows that the total proportion of notes aligned such that the onsets are within 1 second of the correct onset is 94.81%, indicating that overall the methodology is capable of producing alignments. The median absolute onset error is 51.05, while the mean 263.4. The standard deviation is quite large, at 830.2. The empirical distribution of the absolute onset errors is shown on ***Figure 5.1*** using the empirical probability mass functions.

Model	Mean	Median	S.D.	≤ 10 ms	≤ 30 ms	≤ 50 ms	≤ 100 ms	≤ 1 s
Model 1 w/ CQT	263.4	51.05	830.2	19.13	39.31	49.59	63.35	94.81
Kwon et al. (2017)	8.62	5.57	31.14	91.60	98.00	98.97	99.61	
Wang et al. (2019)	6.46	4.65	11.11	86.41	98.71	99.52	99.92	

Table 5.2: *Alignment performance of the proposed model (Model 1) with the CQT audio features, compared to two state-of-the-art models. The results are the note onsets errors, in milliseconds. Right five columns show the percentage of absolute errors within the corresponding thresholds.*

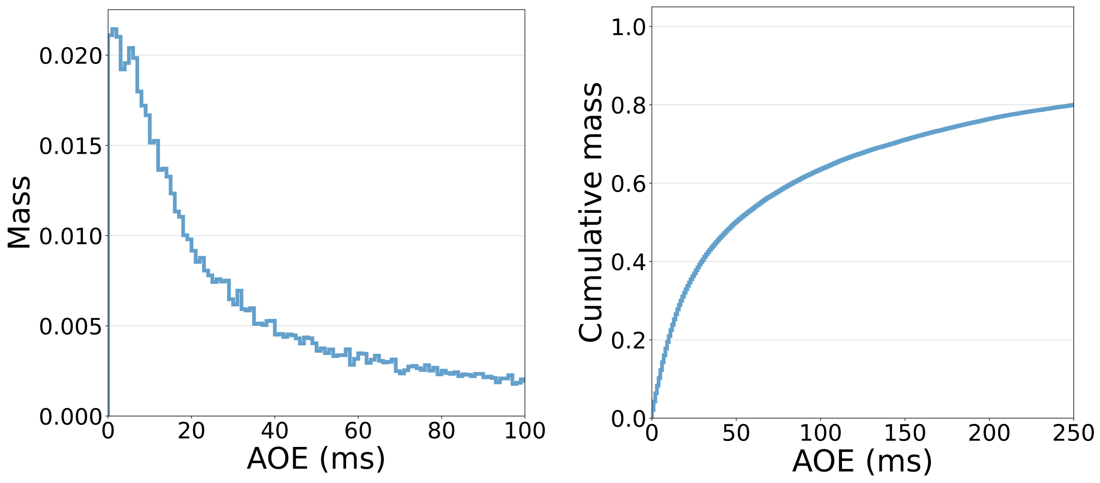

Figure 5.1: *Empirical probability mass (left) and cumulative mass (right) functions of the absolute onset errors (AOE) of the alignment under the proposed model (Model 1).*

The fact that the standard deviation is very large along with the fact that the median is a lot smaller than the mean suggests that there are heavy outliers among the aligned notes. This is not surprising, since the proposed method has drawbacks that can potentially hinder its performance in some circumstances.

Furthermore, the alignment performance differs greatly by instrument. The results split by instrument are shown in ***Table 5.3***. For the Disklavier recordings, the mean AOE are 362.2 and 423.0 with the microphone up close and far away respectively. The Steinway D software recordings under ambient conditions (`SptkBGAm`) also show signs of poor alignment, with mean AOE 392.2, along with the Bechstein D280 in the "concert hall position" (`AkPnBht`, mean AOE 362.2).

`SptkBGCl`	`AkPnBsdf`	`ENSTDkCl`	`AkPnCGdG`	`StbgTGd2`
135.7	148.2	362.2	147.1	158.4
`SptkBGAm`	`AkPnStgb`	`ENSTDkAm`	`AkPnBcht`	
392.2	152.3	423.0	305.3	

Table 5.3: Mean AOE (in ms) by instrument name. Prefix "ENSTDk" refers to the real piano Disklavier recordings. Suffixes "Cl" and "Am" refer to microphone up close and in an ambient position. All other instruments are software pianos.

With respect to the robustness to errors under increasing variance of the noise from (4.2), the results are shown in ***Table 5.4***. The mean AOE (absolute onset error) remains relatively constant all the way up to $\sigma = 30$, although deterioration in alignment is visible around $\sigma = 20$ among the proportion of onsets within 100 ms which falls below 50% at $\sigma = 30$. The median AOE rises steadily with increased σ, which indicates a decrease in fine alignment precision, and is not surprising. The proportion of the AOE within the 1 second threshold remains relatively constant at 94% until around $\sigma = 30$, which suggests that the general alignment is still achieved even at very high noise in the targets.

σ	Mean	Median	S.D.	≤10 ms	≤30 ms	≤50 ms	≤100 ms	≤1 s
None	263.4	51.05	830.2	19.13	39.31	49.59	63.35	94.81
0.1	267.1	51.75	835.9	19.68	39.63	49.35	62.54	94.61
0.5	291.6	60.11	886.2	18.61	37.43	46.75	60.02	94.34
1	278.3	63.86	787.4	18.06	36.42	45.59	58.91	94.44
2	310.6	65.28	995.5	16.88	35.00	44.69	58.93	93.97
5	283.1	68.13	818.2	15.11	32.95	43.31	58.86	94.12
10	270.5	74.10	743.8	13.04	30.30	41.00	57.35	94.34
20	281.8	94.48	717.7	9.70	24.37	34.80	51.39	94.55
30	287.1	118.2	629.5	7.56	20.29	29.83	45.69	94.70
50	411.8	186.3	688.8	5.16	14.09	21.22	34.40	89.77

Table 5.4: *Performance of the proposed model under simulated mistakes, or deviations from the score. For simulation, MIDI targets were perturbed by white noise:* $\tilde{g}(x) = g(x) + e$ *where* g *is an aggregating function and* $e \sim N(0, \sigma^2)$*. After that, alignment performance is evaluated as before. The general alignment is preserved all the way up to* $\sigma = 30$*, but finer detail is lost at each* σ *increase.*

5.3 The Good

Some songs obtain a precise alignment under the proposed methodology. An example of one is shown on ***Figure 5.2***.

For that song, the predicted features follow the true MIDI features quite closely. It is also notable that the CQT spectrogram of the audio features looks relatively clean, and the notes can be spotted by the eye. This likely makes the job easier for the model. Even though some predicted features stick out a bit, the general extraction performance is good enough to achieve a very good alignment.

The mean and the median AOE for the alignment of this song is 34 ms and 15 ms respectively, which is a lot lower than the average AOE in the test set.

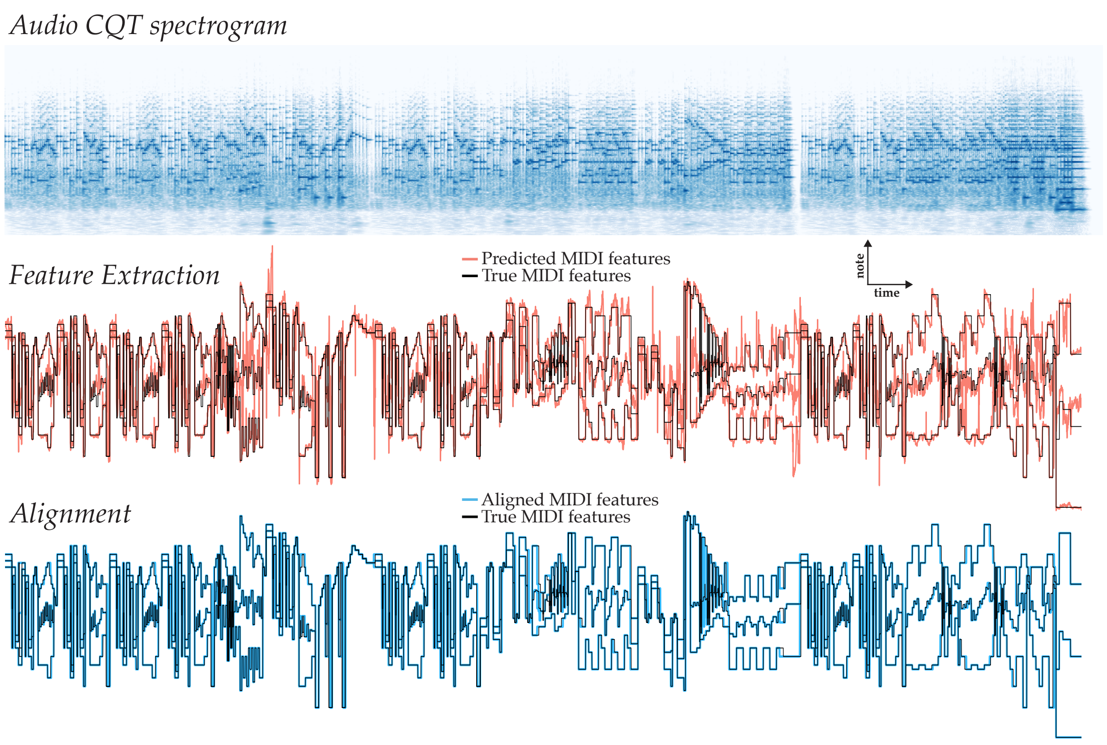

Figure 5.2: *The song with the best resulting alignment, i.e. lowest mean absolute onset error. The mean alignment error here is 34 ms. Even though there is visible noise in the predicted features, the alignment is good.*

5.4 The Bad

An issue with the dataset, which became apparent only during the evaluation stage, is that some tracks contain certain notes that are sustained in the audio, but not sustained in the MIDI. ***Figure* 5.3** shows an example of this. The problematic regions are highlighted in dashed red circles. This is either caused by the overlapping signals for the sustain pedal, or the lowest note is held past the pedal press. We were not able to reproduce this feature of the recording of this MIDI file using software instruments to playback the MIDI file.

This suggests a limitation of the proposed low-dimensional approach. Since there are only three predicted features, having just one of the predicted features being so far away from the true feature can increase the loss by a wide margin, as shown here. Under the classification task of a standard AMT model, the loss would not increase greatly if the model gets a single note wrong.

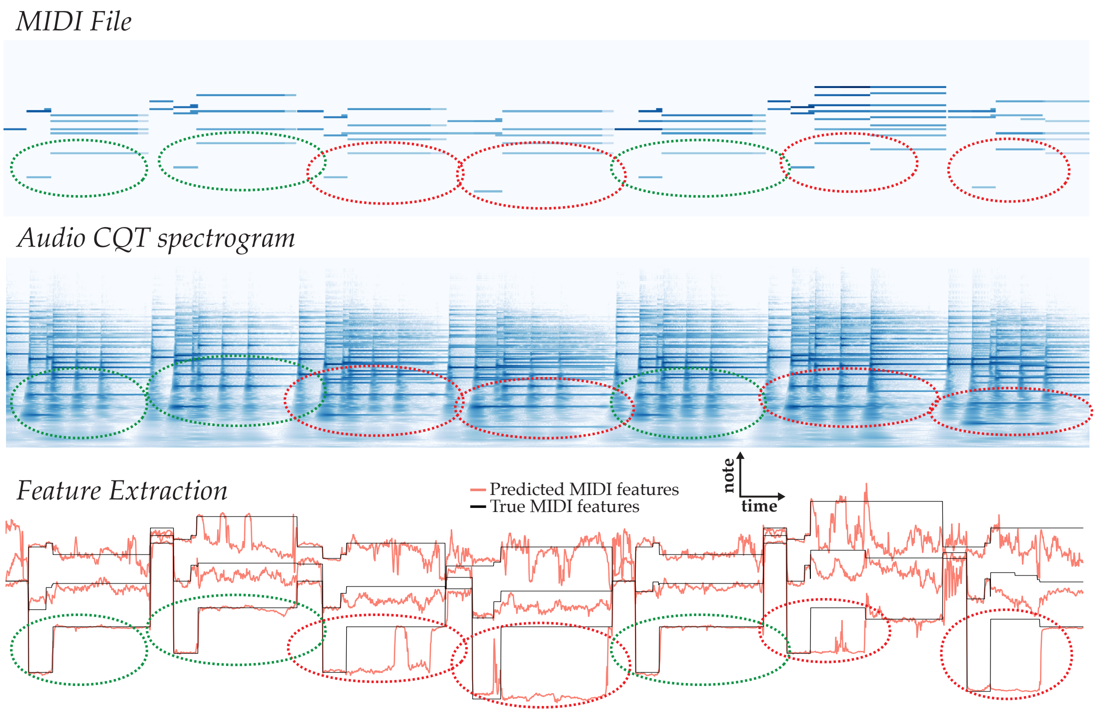

Figure 5.3: *Section with particularly high error in the lowest notes. The notes encircled in red were sustained by the pedal and do not match with the corresponding notes in the MIDI file. The model is predicting the features correctly.*

5.5 The Ugly

To gain insight into the worst scenario for the proposed method, the song with the worst alignment is shown in ***Figure 5.4***. The MIDI file shows The aggregated MIDI features for this piece look quite unstable, especially in the middle section. This is caused by short low notes. The note is struck and released quickly (known as "staccato"), causing the lowest note feature to oscillate up and down.

This suggests an unfavourable scenario under which the proposed methodology is not effective – when the sustained notes are mixed with the staccato notes, causing instability in the aggregated features. A way to overcome this might be by limiting large leaps in short amounts of time in the aggregated MIDI features, or extending the short notes to cover the gaps between them. The issue with extending the notes is that sometimes the notes need to be short and extending them might cause problems with alignment.

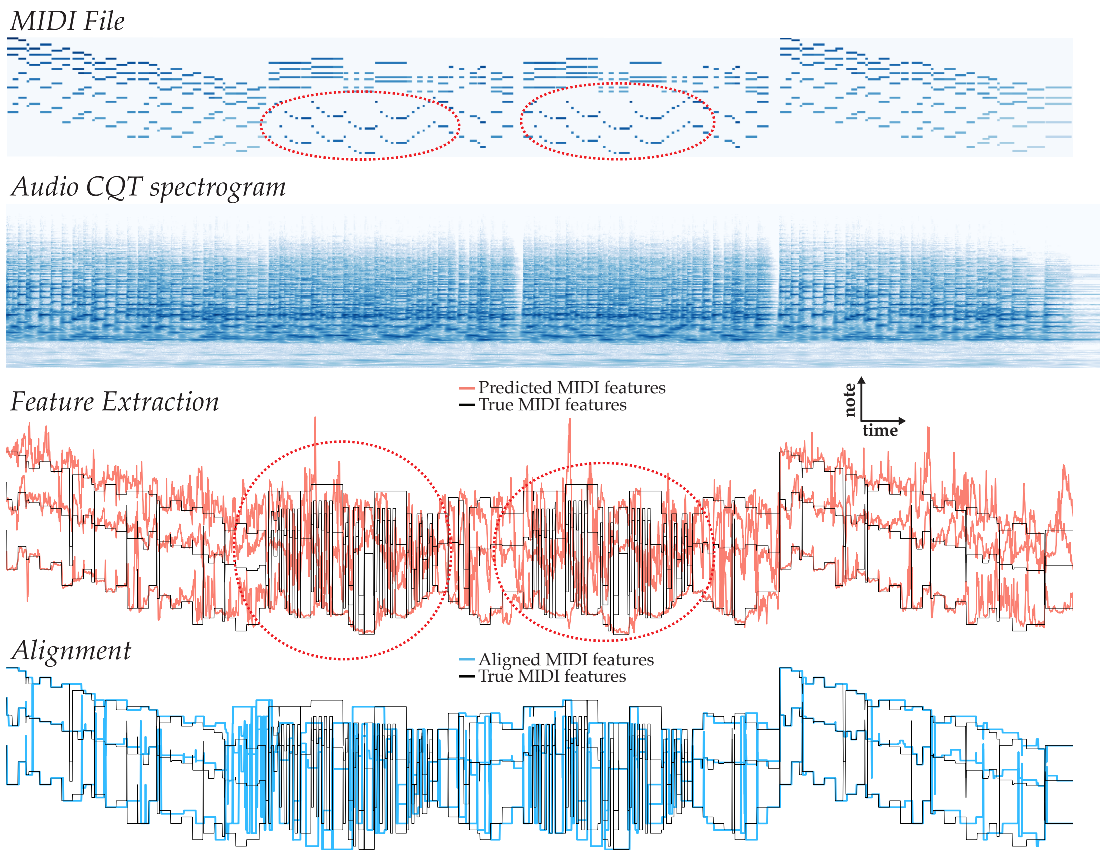

Figure 5.4: *The song with the worst overall alignment – Schumann's Op. 15, No. 6. The alignment is visibly quite poor. The mean absolute onset error here is* 4.492 *seconds, which represents complete misalignment. The middle sections have some messy features that are off by a wide margin. Problematic regions are encircled.*

6 Conclusion

In this chapter the conclusion of the research conducted is concisely recapitulated, together with the challenges that were faced during the research.

The proposed audio-to-MIDI alignment framework features a regression-type supervised feature extraction model using aggregating functions to process the MIDI file from a sparse matrix form into 1−dimensional sequences. This type of feature aggregation is different from the existing methodology whereby it induces a similarity quality onto the target feature space whereby two notes close together on the keyboard remain close together in the feature space. This makes the alignment robust to errors in the performance.

The difficulties faced during the project were numerous, and were mostly related to the implementation and data processing. The state-of-the-art systems do not open-source their implementations, which makes reproducing their work difficult, and meant that everything had to be prepared from scratch: from obtaining MIDI file names from the audio file names, to creating and training the neural networks, to evaluating the alignment performance. The scale of the MAPS dataset, along with high dimensionality of the audio features, made data wrangling a challenge.

6.1 Future Work

There are a lot of things that could have been attempted, had the time allowed.

Only the convolutional networks architecture has been attempted for the regression models. Kwon et al. (2017) had success with the recurrent BLSTM networks, which models the context differently to convolutional networks, and is something that can be explored.

Kwon et al. (2017) also found that adding onset features greatly reduces error (mean AOE: from 25.31 to 8.62). Since there are only 3 features in the proposed framework, the onset features could be represented by a single binary variable, in order to not drastically increase the number of features.

We found that the MAPS dataset contains some inaccuracies between the MIDI and the audio files. A different dataset might offer a higher quality model training. The Hawthorne et al. (2019) dataset is larger and contains high quality recordings from international piano competitions. Although the recordings are less varied, since they all come from the same source. A varied publicly-available dataset with matching MIDI and audio files coming from different sources does not exist at the time of the writing.

References

Abadi, M., Agarwal, A., Barham, P., and et. al. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Agrawal, R., Wolff, D., and Dixon, S. (2022). A convolutional-attentional neural framework for structure-aware performance-score synchronization.

Arias-Vergara, T., Klumpp, P., Vasquez, J., Noeth, E., Orozco, J. R., and Schuster, M. (2021). Multi-channel spectrograms for speech processing applications using deep learning methods. *Pattern Analysis and Applications*, 24:1–9.

Bracewell, R. N. and Bracewell, R. N. (1986). *The Fourier transform and its applications*, volume 31999. McGraw-Hill New York.

Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., and Shah, R. (1993). Signature verification using a "siamese" time delay neural network. In Cowan, J., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems*, volume 6. Morgan-Kaufmann.

Brown, J. C. (1991). Calculation of a constant q spectral transform. *The Journal of the Acoustical Society of America*, 89(1):425–434.

Brusa, E., Delprete, C., and Di Maggio, L. (2021). Randomized eigen-spectrograms extraction for an effective fault diagnosis of bearings.

Böck, S. and Schedl, M. (2012). Polyphonic piano note transcription with recurrent neural networks. pages 121–124.

Cangea, C., Velikovi, P., and Liò, P. (2017). Xflow: Cross-modal deep neural networks for audiovisual classification.

Cheng, Q. and Gu, X. (2021). Bridging multimedia heterogeneity gap via graph representation learning for cross-modal retrieval. *Neural Networks*, 134:143–162.

Cooley, J. W. and Tukey, J. W. (1965). An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301.

Delisle, J.-B., Hara, N., and Sé gransan, D. (2020). Efficient modeling of correlated noise. *Astronomy &amp Astrophysics*, 638:A95.

Dennis, J., Dat, T., and Chng, E. (2014). Analysis of spectrogram image methods for sound event classification. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, pages 2533–2537.

Dörfler, M., Bammer, R., and Grill, T. (2017). Inside the spectrogram: Convolutional neural networks in audio processing. In *2017 International Conference on Sampling Theory and Applications (SampTA)*, pages 152–155.

Emiya, V., Bertin, N., David, B., and Badeau, R. (2010). Maps - a piano database for multipitch estimation and automatic transcription of music.

Ewert, S., Müller, M., and Grosche, P. (2009). High resolution audio synchronization using chroma onset features. pages 1869 – 1872.

Finson, J. W. (2002). Nineteenth-century music : the western classical tradition. prentice hall.

Foote, J. (1999). Methods for the automatic analysis of music and audio. Technical report, In Multimedia Systems.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics volume 36*, pages 193–202.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In Gordon, G., Dunson, D., and Dudík, M., editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA. PMLR.

Hadsell, R., Chopra, S., and LeCun, Y. (2006). Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

Hawthorne, C., Stasyuk, A., Roberts, A., Simon, I., Huang, C.-Z. A., Dieleman, S., Elsen, E., Engel, J., and Eck, D. (2019). Enabling factorized piano music modeling and generation with the MAESTRO dataset. In *International Conference on Learning Representations*.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.

Higuchi, H., Asahi, K., Sagawa, Y., and Sugie, N. (2004). Sound source separation with two spectrograms by image processing. *Ieej Transactions on Electronics, Information and Systems*, 124:2439–2445.

Hinton, G. E., O. S. . T. Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7).

Hirschman, I. I. and Widder, D. V. (1955). Convolution transform. volume Princeton University Press.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):17351780.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558.

Hullah, J. (1876). Rudiments of musical grammar. new ed. london: Longmans, green, reader, and dyer.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.

ISMIR (2022). International society for music information retrieval (ismir). [Online; accessed 1 September 2022].

Joder, C., Essid, S., and Richard, G. (2011). A conditional random field framework for robust and scalable audio-to-score matching. *Audio, Speech, and Language Processing, IEEE Transactions on*, 19:2385 – 2397.

Khalighifar, A., Jiménez-García, D., Campbell, L., Ahadji-Dabla, K., Aboagye-Antwi, F., Ibarra, L., and Peterson, A. (2021). Application of deep learning to community-science-based mosquito monitoring and detection of novel species. *Journal of Medical Entomology*, 59.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C., Bottou, L., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

Kwon, T., Jeong, D., and Nam, J. (2017). Audio-to-score alignment of piano music using rnn-based automatic music transcription. Available online: [Accessed 1 September 2022].

Liu, J., Kong, X., Xia, F., Bai, X., Wang, L., Qing, Q., and Lee, I. (2018). Artificial intelligence in the 21st century. *IEEE Access*, PP:1–1.

McFee, B., Metsai, A., McVicar, M., Balke, S., Thomé, C., Raffel, C., Z. F., and et. al. (2022). librosa: 0.9.2 (0.9.2). Zenodo. https://doi.org/10.5281/zenodo.6759664.

Mead, C. A. (1964). Possible connection between gravitation and fundamental length. *Phys. Rev.*, 135:B849–B862.

Müller, M. (2007). Dynamic time warping. *Information Retrieval for Music and Motion*, 2:69–84.

Müller, M. (2021). Music representations. [Accessed 1 September 2022].

Müller, M., Mattes, H., and Kurth, F. (2006). An efficient multiscale approach to audio synchronization. In *ISMIR*, pages 192–197.

Raffel, C. and Ellis, D. P. W. (2014). Intuitive analysis, creation and manipulation of midi data with pretty_midi. in 15th international conference on music information retrieval late breaking and demo papers.

Raffel, C. and Ellis, D. P. W. (2015). Large-scale content-based matching of midi and audio files. In *ISMIR*.

Rao, K. R., Kim, D. N., and Hwang, J. J. (2010). *Discrete Fourier Transform*, pages 5–40. Springer Netherlands, Dordrecht.

Rodriguez, C., Angeles, D., Chafloque, R., Kaseng, F., and Pandey, B. (2020). Deep learning audio spectrograms processing to the early covid-19 detection. pages 429–434.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. psychological review, 65(6), 386408.

Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning representations by back-propagating errors. nature 323, 533536.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2014). Imagenet large scale visual recognition challenge.

Sakoe, H. and Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49.

Salvador, S. and Chan, P. K. (2004). Fastdtw: Toward accurate dynamic time warping in linear time and space.

Sangeetha, J., Hariprasad, R., and Subhiksha, S. (2021). Chapter 8 - analysis of machine learning algorithms for audio event classification using mel-frequency cepstral coefficients. In Dey, N., editor, *Applied Speech Processing*, Primers in Biomedical Imaging Devices and Systems, pages 175–189. Academic Press.

Sarfraz, M. S. and Stiefelhagen, R. (2016). Deep perceptual mapping for cross-modal face recognition. *International Journal of Computer Vision*, 122(3):426–438.

Schörkhuber, C. and Klapuri, A. (2010). Constant-q transform toolbox for music processing. In *Proceedings of the 7th Sound and Music Computing Conference, Barcelona, Spain*.

Simonetta, F., Ntalampiras, S., and Avanzini, F. (2021). Audio-to-score alignment using deep automatic music transcription. Available online: `https://arxiv.org/abs/2107.12854` [Accessed 1 September 2022].

Turetsky, R. and Ellis, D. (2003). Ground-truth transcriptions of real music from force-aligned midi syntheses. *4th International Symposium on Music Information Retrieval ISMIR-03*, pp. 135-141.

Wang, Y., Liu, S., and Guo, L. (2019). An improvement on audio-to-midi alignment using triplet pair. In *1st International Workshop on Multimodal Understanding and Learning for Embodied Applications*, MULEA '19, page 4348, New York, NY, USA. Association for Computing Machinery.

Yamashita, R., N. M. D. R. e. a. (2018). Convolutional neural networks: an overview and application in radiology. *Bulletin of mathematical biology*, Insights Imaging 9, 611629.

Zhu, Y., Yang, Z., Wang, L., Zhao, S., Hu, X., and Tao, D. (2019). Hetero-center loss for cross-modality person re-identification.