# Automatic Chord Recognition with Deep Learning

*Pierre Lardet*

4th Year Project Report
Computer Science and Mathematics
School of Informatics
University of Edinburgh

2025

# Abstract

This skeleton demonstrates how to use the `infthesis` style for undergraduate dissertations in the School of Informatics. It also emphasises the page limit, and that you must not deviate from the required style. The file `skeleton.tex` generates this document and should be used as a starting point for your thesis. Replace this abstract text with a concise summary of your report.

# Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

# Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Pierre Lardet*)

# Acknowledgements

# Table of Contents

# Chapter 1

# Introduction

## 1.1 Motivation

TODO: Rewrite introduction to focus on chord recognition.

- Useful for musicians - Useful for musicologists - Connection to lead sheets

Chords form an integral part of music. Used in music of all forms. Used for research

To this end, we investigate the use of deep learning for automatic chord recognition. Data-drive methods have dominated the field of automatic music transcription in recent years, and have shown great promise for chord recognition. However, progress has not been made since 2015. Why? What might be done to improve?

We conduct a thorough analysis of the state-of-the-art models for automatic chord recognition, and investigate methods of improving on these models. We look at the ways others have improved these models and compare and contrast them.

We also take inspiration from other fields of music transcription and leverage modern generative models to provide new representations training data and generate synthetic data itself.

All code is available on Github.[1] Data can be available upon request.[2]

## 1.2 Aims

The aims of this project are:

- Compare state-of-the-art models for automatic chord recognition.
- Conduct a thorough analysis of the models and their performance.
- Investigate methods of improving on these models.

[1] `https://github.com/PierreRL/LeadSheetTranscription`

[2] `lardet[dot]pierre[at]gmail.com`

- To perform experiments with new sources of data, data augmentation and synthetic data generation.

## 1.3 Outline

The report is structured as follows:

- **Chapter 2** provides background information on chord transcription and related work.
- **Chapter 3** describes the datasets, evaluation metrics and training procedure used in this project.
- **Chapter 4** compares various models from the literature and investigates improvements.
- **Chapter 5** extends this work with synthetic data generation and compares results on a new dataset.
- **Chapter 6** concludes the report and provides suggestions for future work.

# Chapter 2

# Background & Related Work

In this chapter, I first provide a brief introduction to harmony and chords and their role in music. I then discuss the different ways in which music can be represented as input to a machine learning model. This is followed by an overview of the field of automatic chord recognition (ACR). This includes the datasets, models, and evaluation metrics that are commonly used in ACR and the challenges that are faced in this field. Finally, we discuss related work in the generation of synthetic data for music transcription tasks.

## 2.1 Background

### 2.1.1 Harmony and Chords

Harmony is the combination of simultaneously sounded notes. A common interpretation of such sounds is as a chord, especially in Western music. Chords can be thought of as a collection of at least two notes, built from a root note often with the third and fifth degrees of the scale. They can be extended with any notes but the most common are the seventh, ninth, eleventh and thirteenth upper extensions. A chord's *quality* is determined by the intervals between notes in the chord irrespective of the root note. The most common chord qualities are major and minor, built from the major and minor scales. Many other qualities exist such as diminished, augmented, and suspended chords. Chords can be played in *inversion*, where the root note is not the lowest note and can be played in different *voicings*, where the notes are played in different octaves. In this work, we represent chords using Harte notation [Harte et al., 2005] as described in Section 3.1.1.2.

Chords can be closely related. `C:maj7` is very close to `C:maj`, the only difference is an added major seventh. An important relation in music theory is between *relative major/minor* chords. These pairs of chords are built from the same scale so often share many notes. For example, `G:maj` and `E:min` are related in this way. If we then add extensions to these chords, they can become even more closely related. `G:maj6` and `E:min7` share the same set of notes played in different orders.

Chords are an important part of music. They provide a harmonic context for a melody, and can be used to convey emotion, tension and release [Aldwell et al., 2010]. They are also important for improvisation where musicians will often play notes that fit the chord progression such that they a create pleasing sound [Levine, 1995]. Many forms of musical musical notation rely on chords. Contemporary guitar music and accompaniments are often represented by just a chord sequence Simplicio [2003]. Chords are integral to lead sheets, a musical notation which strips down a piece of music to its melody and chord sequence. Lead sheets are often used for improvisation, especially in jazz music. A lead sheet for ‘Yesterday’ by The Beatles can be found in Figure 2.1. Chords are also important for songwriting, where a chord progression form the basis of a song. Music analysis also makes heavy use of chords. The harmonic structure of a piece can be analysed to better understand the composer’s intentions, and to understand why we enjoy certain kinds of music [Rohrmeier and Cross, 2008].

### 2.1.2 Chord Recognition

Chord recognition is the task of identifying the chords present in a piece of music. This can be useful for creating notated versions of songs for musical analysis, recommendation and generation. Those wishing to learn pieces of music may start by visiting websites such as Ultimate Guitar[1] where users submit chord annotations for songs. Musicologists may wish to analyse the harmonic structure of a piece of music or analyse the changes in common chord sequences over time and location. Music recommendation systems may recommend songs based on their harmonic content as similar music will often have similar harmonic content [Tzanetakis and Cook, 2002]. For example, modern pop music famously uses many similar chords [2] while contemporary jazz music is known for its complex and rich exploration of harmony. Music generation systems can generate audio based on a given harmonic structure [Jung et al., 2024].

All of the above motivate the need for accurate chord annotations. However, annotations from online sources can be of varying quality and may not be available for all songs [de Berardinis et al., 2023]. The task of annotating chords is time-consuming and requires a trained musician [Burgoyne et al., 2011]. Automatic chord recognition systems have the potential to alleviate these problems by providing a fast, accurate and scalable solution.

Chord recognition is a non-trivial task. Which chord is playing when is inherently ambiguous. Different chords can share the same notes and the same chord can be played in a myriad ways. The same chord can also be played in different contexts, such as a different key, time signature or on a different instrument. Precisely when a chord starts and ends can be vague and imprecise. Whether a melody note is part of a chord is ambiguous and whether a melody alone is enough to imply harmonic content is also ambiguous. In order to identify a chord, data across time must be considered as the chordal information may be spread out over time. For example, a chord may vamped or arpeggiated. Audio also contains many unhelpful elements for chord recognition such as reverb, distortion and unpitched percussion. Combined with the lack of labelled data, this makes ACR a challenging task.

[1]https://www.ultimate-guitar.com/

[2]https://www.youtube.com/watch?v=oOlDewpCfZQ


Figure 2.1: An example of a lead sheet for 'Yesterday' by the Beatles. We can see chords written above the stave and the melody written in standard musical notation. Such a chordal representation is useful for musicians who want to learn and perform songs quickly or improvise around them.

### 2.1.3 Music Features

Recorded music can be represented in a variety of ways as input to a machine learning model. The simplest is to leave the data as a raw time-series of amplitudes, referred to as the audio's waveform. Data in the raw audio domain has been successfully applied in generative models such as Jukebox [Dhariwal et al., 2020], RAVE [Caillon and Esling, 2021] and MusicGen [Copet et al., 2023]. Such models transform the raw audio into discrete tokens allowing a language model to predict future tokens which are then decoded back into audio.

**Spectrogram**: A common representation of audio data is the spectrogram. A spectrogram is a transformation of the time-series data into the time-frequency domain, calculated via a short-time Fourier transform (SFTF). Spectrograms are commonly used in many audio processing tasks such as speech recognition, music recognition [Wang, 2003] and music transcription, specifically polyphonic transcription [Toyama et al., 2023]. As of yet, linear spectrograms computed using the STFT have not been used in ACR tasks [Pauwels et al., 2019].

**Mel-spectrogram**: A common alternative to the standard linear spectrogram is the mel-spectrogram. The only difference is that the frequency scale is no longer linear. This transformation uses the mel-scale [Stevens et al., 1937]. The scale was constructed using estimates of human perception of different frequencies. It is approximately linear below 1kHz and logarithmic above. The mel-spectrogram is commonly used in speech recognition [Lee et al., 2021] and has also been used in music transcription tasks [Chris Donahue and Liang, 2022].

**CQT**: A common version of the spectrogram used in music transcription is the constant-Q transform (CQT) [Brown, 1991]. The CQT is another version of the spectrogram with frequency bins that are logarithmically spaced and bin widths that are proportional to the frequency. This is motivated by the logarithmic nature of how humans perceive pitch intervals in music: a sine wave with double the frequency is perceived as one octave higher. As such, CQTs are used in many music transcription tasks and are very popular for ACR [Humphrey and Bello, 2012a, McFee and Bello, 2017]. An example CQT from the dataset used in this work is shown in Figure 2.2. As Korzeniowski and Widmer [2016b] note, CQTs are preferred to other spectrograms for ACR due to their finer resolution at lower frequencies and for their ease with which pitches can be studied and manipulated. For example, CQTs make pitch shifting possible through a simple shifting of the CQT bins.

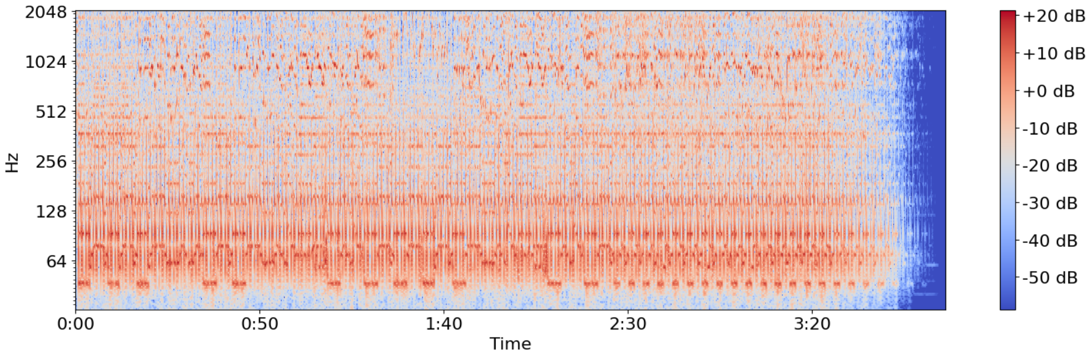


Figure 2.2: The CQT of 'Girls Just Wanna Have Fun' by Cyndi Lauper from the dataset used in this work. We can see the log-spaced frequency bins on the y-axis. There is clear structure and repetition in the song, particularly in the lower frequencies, which can be attributed to a regular drum groove and bass instruments. We can interpret movement in the upper frequencies corresponding to the melody. This is typical of songs in this dataset. Such structure and repetition gives an idea of the patterns a machine learning model may look for to identify chords.

**Chroma Vectors**: Chroma vectors are a 12-dimensional time-series representation, where each dimension corresponds to a pitch class. Each element represents the strength of each pitch class in the Western chromatic scale in a given time frame. Such features have been generated by deep learning methods [Miller et al., 2022] or by hand-crafted methods [Mauch and Dixon, 2010, McFee et al., 2015] and have seen use in recent ACR models [Chen and Su, 2019]. A representation of a song as a chroma vector over time can be thought of as a another type of spectrogram, referred to as a *chromagram*.

**Generative Features**: More recently, features extracted from generative music models have been used as input. I refer to such features as `generative features`. The proposed benefit is that the vast quantities of data used to train these models requires rich representations of the music. These features have been shown to contain useful information for music information retrieval (MIR) tasks [Castellon et al., 2021]. Chris Donahue and Liang [2022] use features from JukeBox [Dhariwal et al., 2020] to train a transformer [Vaswani et al., 2023] for both melody transcription and chord recognition. They found that these features outperformed mel-spectrograms in melody transcription tasks but did not report results for ACR.

## 2.2 Related Work

The field of ACR has seen considerable research since the seminal work of Fujishima [1999] in 1999. Below, I provide a brief overview of the field over the last 15 years including the datasets, metrics and models and representations of time that are commonly used. I conclude by discussing some of the common challenges faced and motivating the research carried out in this project.

### 2.2.1 Data

Sources of data that have seen common use in ACR relevant to this work include:

- *Mcgill Billboard*: over 1000 chord annotations of songs randomly selected from the Billboard 'Hot 100' Chart between 1958 and 1991. [Burgoyne et al., 2011]
- *Isophonics*: 300 annotations of songs from albums by The Beatles, Carole King and Zweieck. [Cannam et al., 2009]
- *RWC-Pop*: 100 pop songs with annotations available[3] for chords. [Goto et al., 2002]
- *USPop*: 195 annotations of songs chosen for popularity. [Berenzweig et al., 2004]
- *JAAH*: 113 annotations of a collection of jazz recordings. [Durán and de la Cuadra, 2020]
- *HookTheory*: 50 hours of labelled audio in the form of short musical segments, crowdsourced from an online forum called HookTheory[4]. [Chris Donahue and Liang, 2022]

Other datasets also have been used but are less relevant to this work. Many of these have been compiled together into the *Chord Corpus* by de Berardinis et al. [2023] with standardised annotation formats. However, audio is scarce, in part due to copyright issues. This has lead to discrepancies between evaluation sets used across works, making direct comparison challenging. The most common dataset is comprised of 1217 songs compiled from the first four of the above collections. This dataset is dominated by pop songs. Little work has been done on evaluation across genres.

Another problem is that the existing data is imbalanced, with a large number of common chords present like major and minor chords and fewer instances of chords with more obscure qualities like diminished and augmented chords. This can lead to models that are biased towards predicting major and minor chords. Attempts to address such a long-tailed distribution have been made by weighting the loss function [Jiang et al., 2019], adding a term in the loss function rewarding the identification of individual notes [McFee and Bello, 2017, Jiang et al., 2019], re-sampling training examples to balance chord classes [Miller et al., 2022] and curriculum learning [Rowe and Tzanetakis, 2021].

**Pitch Augmentation**: Due to the lack of labelled data, data augmentation via pitch shifting has been applied to ACR. Input audio and features are pitch shifted while

[3] `https://github.com/tmc323/Chord-Annotations`

[4] https://www.hooktheory.com/

chords are transposed. McFee and Bello [2017] note the large increase in performance using pitch shifting directly on the audio. Other works have since used pitch shifting on the audio [Park et al., 2019] or on the CQT [Jiang et al., 2019]. While augmentation directly on the audio can create artefacts that may provide variety compared to simply shifting the bins of the CQT, it is not clear whether this is beneficial. No work I found has compared the two methods.

**Synthetic Data Generation**: Data has been scaled up using augmentation and semi-supervised learning with some success [Hung et al., 2023]. Research has been done into the use of synthetic data [Kroher et al., 2023, Sato and Akama, 2024] and self-supervised learning [Li et al., 2024] for other MIR tasks but not for ACR. With the advent of new models which accept chord-conditioned input [Jung et al., 2024, Lan et al., 2024, Lin et al., 2023], the possibility of generating synthetic data for ACR is an exciting avenue of research.

### 2.2.2 Evaluation

Evaluation is typically done using weighted chord symbol recall (WCSR). This is defined as the recall of each chord class weighted by its duration. Simply put, this is the fraction of time that the prediction is correct. We define this more formally in Section 3.2. More music-aware measures of recall provide further insights such as the recall of the correct root note, third, seventh or mirex, a metric which checks whether a predicted chord has at least 3 notes in common with the true chord. These metrics are implemented by Raffel et al. [2014] in the `mir_eval` library.[5]

Other metrics targeting the imbalanced nature of the data have been proposed. These include the mean accuracy over classes and qualities. However, these metrics are defined in terms of discrete frames. I propose a definition in continuous time, similar to WCSR.

Some works also do a little qualitative evaluation. Chris Donahue and Liang [2022] provide examples of the lead sheets their model produces and categorises failure modes. However, most works focus solely on improving quantitative metrics and conducting analysis on the internal elements of the model. There is a lack of focus on the kinds of errors their models makes and on the utility of the model in real-world applications.

### 2.2.3 Models

**Model Architectures**: Since the work of Humphrey and Bello [2012b], chord recognition has been predominantly tackled by deep learning architectures. The authors used a convolutional neural network (CNN) to classify chords from a CQT spectrogram. CNNs have been combined with recurrent neural networks (RNNs) [Wu et al., 2019, Jiang et al., 2019, McFee and Bello, 2017] with a CNN performing feature extraction from a spectrogram and an RNN sharing information across frames. More recently, transformers have been applied in place of the RNN or as the sole architecture present [Chris Donahue and Liang, 2022, Chen and Su, 2019, 2021, Akram et al., 2025, Rowe and Tzanetakis, 2021, Park et al., 2019].

[5] `https://mir-evaluation.github.io/mir_eval/`

Despite increasingly complex models being proposed, performance has not improved significantly. In fact, Park et al. [2019] found that their complex transformer performed marginally worse than a simple CNN. Humphrey and Bello [2015] talk of a 'glass ceiling' with increases in performance stagnating after the advent of deep learning in ACR. This was 10 years ago and the situation has not changed significantly. Despite this, continued efforts have been made to develop complex models with the sole motivation of improving performance, with mixed success. This has lead to overly complex ACR models seeing use in this other MIR tasks such as chord-conditioned generation where Lan et al. [2024] use the model developed by Park et al. [2019] despite its lack of improvement over simpler predecessors. Furthermore, there is little comparison to simple baselines to provide context for the performance gain associated with increasing model complexity.

**Decoding**: A decoding step is often performed on the probabilities outputted by the neural network. This can smooth predictions and share information across frames. Decoding follows the Viterbi algorithm to find the most likely sequence of chords given the model's output. Miller et al. [2022] use a hidden Markov model (HMM), treating the probability distributions over chords generated by the model as emission probabilities and constructing a hand-crafted transition function. Other works have used conditional random fields (CRF) instead to model the dependencies between chords [Jiang et al., 2019]. Both methods have used learned statistics and simpler homogeneous penalties between different chords for transitions to different chords. It is unclear which method is better. In both cases, self-transition probabilities are very large and Cho et al. [2010] argue that increases in performance can be mostly attributed to the reduction in the number of transition. However, more recent analysis of such behaviour is missing from the literature.

**Model Analysis**: Korzeniowski and Widmer [2016a] visualise the outputs of layers of the CNN and find that some feature maps correspond to the presence of specific pitches and intervals. Korzeniowski and Widmer [2016b] visualise the importance of different parts of an input CQT using saliency maps, noting the clear correlation between pitch classes present in a chord and the saliency maps. Confusion matrices over chord roots and qualities are also commonly used to analyse the performance of models. For example, McFee and Bello [2017] found that similar qualities are often confused with each other and that the model favours the most common chord qualities. Park et al. [2019] attempt to interpret attention maps produced by their transformer as musically meaningful.

Regardless of such analyses, too much effort is spent on motivating complex model architectures with a focus on minor improvements in performance. In this work, I will conduct a thorough analysis of an existing model. I will take inspiration from some of the analyses above, while adding a more nuanced understanding of the model's behaviour and failure modes by way of example.

### 2.2.4 Frames and Beats

Chords exist in time. How the time dimension is processed prior to being fed into the model matters. When audio is transformed into a spectrogram, each vector of frequencies represents a fixed length of time, called a *frame*. The frame length is determined by hop length used when calculating the CQT. Constant frame lengths can be made short enough such that the constraint imposed on the model to output chord predictions on a per-frame basis is not limiting. However, different hop lengths have been used, varying from 512 Jiang et al. [2019] up to 4096 [McFee and Bello, 2017]. Which hop length works best remain unclear.

More recently, Chris Donahue and Liang [2022] used a frame length determined by beats detected from the audio. Because they focus primarily on melody transcription, they define frames to be a 1/16th note $\approx$ 125ms with 120 beats per minute (BPM). Such beat synchronicity has been proposed for chord recognition. The underlying assumption is that chords tend to change on the beat. This reduces the computational cost of running the model due to a decreased frame rate and more importantly provides a far more musically meaningful interpretation to the output. However, Cho et al. [2010] and Cho and Bello [2014] argue that because beat detection is far from perfect, restricting frames to beats can hurt performance. Beat detection models have improved since then. Proper analysis of beat-synchronous chord recognition in the modern setting is lacking from the literature. Pauwels et al. [2019] propose that we revisit this idea and I agree.

### 2.2.5 Future Directions

Pauwels et al. [2019] provide an overview of ACR up to 2019 since the seminal work of Fujishima [1999] in 1999 and provide suggestions for future avenues of research. They look at future research directions. This includes the use of different representations for both audio and chords, of addressing the mismatch between chord changes and discretised frames fed to a model, looking at the larger structures in music like verses and chords, incorporating other elements of the music such as melody and genre, methods of handling subjectivity of chords and the imbalance present in chord datasets. Since then, different works have addressed some of these problems in various ways. Among these problems, the focus has been primarily on addressing the imbalance in the chord dataset.

In this work, I will implement a simple model that remains competitive with the state-of-the-art [McFee and Bello, 2017]. I will then conduct a thorough analysis of the model and its architecture. I will look at common methods for improving ACR models with more detailed analyses than have previously been conducted. This analysis will provide insight into the strengths and weaknesses of such models. It may also provide guidance for further improvements. I will also look at novel methods of improvement made possible through generative and beat detection models. This includes the use of generative features and synthetic data as input to the model as well as the use of beat-synchronous frames. Finally, I will evaluate the improved models in terms of their performance in-distribution, across genres and as a tool for musicians and musicologists.

# Chapter 3

# Experimental Setup

In this chapter, I outline the datasets used in this work, the preprocessing applied to the audio and chord annotations, the evaluation metrics used to compare the models and details of the training process.

## 3.1 Data

Most of the initial time on this project was spent on finding a suitable dataset for training and testing. Durán and de la Cuadra [2020] use the *JAAH* dataset while Chris Donahue and Liang [2022] use the *HookTheory* dataset, defined in Section 2.2.1. Many works use combination of the *McGill Billboard*, *Isophonics*, *RWC-Pop* and *USPop* datasets. However, none have audio freely available. Furthermore, annotations come from different sources in different formats. I spent time looking at scraping audio data, looking at pre-computed features of audio which are available for some datasets and compiling annotations in different formats.

I also spent time contacting authors of previous ACR works to see if they could provide me with audio. I was able to get in contact with Andrea Poltronieri, a PhD student at the University of Bologna and one of the authors of the chord corpus or 'ChoCo' for short [de Berardinis et al., 2023]. He provided me with labelled audio for the 1217 songs that are commonly used, alongside labelled audio for the *JAAH* dataset. This was a great help despite it coming later in the project than I would have liked.

Therefore, two ACR datasets are used in this work. The first dataset is referred to as the *Pop* dataset as much of the music in the dataset comes from the Billboard Hot 100 charts or other sources of pop music from the last 70 years. This dataset the focus for much of this dissertation. The second dataset is the *JAAH* (Jazz Annotations and Analysis of Harmony) dataset mentioned and is used to assess the generalisation of the model to jazz music.

The remainder of this section discusses the processing applied to the audio and chord annotations common to both datasets, before discussing details of the *Pop* and *JAAH* datasets relevant to each.

### 3.1.1 Preprocessing

#### 3.1.1.1 Audio to CQT

The audio was first converted to a Constant-Q Transform (CQT) representation explained in Section 2.1.3. This feature common in ACR and is used as a starting point for this work. The CQT was computed using the `librosa` library [McFee et al., 2015], using the built-in `cqt` function. A sampling rate of 44100Hz was used, with a hop size of 4096, and 36 bins per octave, 6 octaves and a fundamental frequency corresponding to the note `C1`. These parameters were chosen to be consistent with previous works [McFee and Bello, 2017] and with common distribution formats. The CQT is returned as a complex-valued matrix containing phase, frequency and amplitude information. Phase information was discarded by taking the absolute value before being converted from amplitude to decibels (dB), equivalent to taking the logarithm.

This leads to a CQT matrix of size $216 \times F$ where 216 is the number of frequency bins and $F$ is the number of frames in the song. The number of frames can be calculated as $F = \lceil \frac{44100}{4096} L \rceil$ where $L$ is the length of the song in seconds, 44100 is the sampling rate in Hertz (Hz) and 4096 is the hop length in samples. A 3 minute song has just under 2000 frames. To save on computational cost, the CQT was pre-computed into a cached dataset rather than re-computing each CQT on every run.

#### 3.1.1.2 Chord Annotations

The chord annotation of a song is represented as a sorted dictionary, where each entry contains the chord label, the start time and duration. The chord label is represented as a string in Harte notation [Harte et al., 2005]. For example, C major 7 is `C:maj7` and A half diminished 7th in its second inversion is `A:hdim7/5`. The notation also includes `N` which signifies that no chord is playing and `X` symbolising an unknown chord symbol.

This annotation is too flexible to be used as directly as a target for a machine learning classifier trained on limited data. This would lead to thousands of classes, many of which would appear only once. Instead, I define a chord vocabulary. This contains 14 qualities: major, minor, diminished, augmented, minor 6, major 6, minor 7, minor-major 7, major 7, dominant 7, diminished 7, half-diminished 7, suspended 2, and suspended 4. `N` denotes no chord playing and chords outside the vocabulary are mapped to `X`, a dedicated unknown symbol. Letting $C$ denote the size of the chord vocabulary, $C = 12 \cdot 14 + 2 = 170$. This vocabulary is consistent with much of the literature [McFee and Bello, 2017, Humphrey and Bello, 2015, Jiang et al., 2019]. Jiang et al. [2019] use a more detailed vocabulary by also including inversions but I decide to remain consistent with previous works. As McFee and Bello [2017] note, $C = 170$ is sufficient for the dataset to exhibit significant imbalance in the chord distribution. Their methodology is easily extensible to larger vocabularies. If performance is not yet satisfactory on $C = 170$, it is unlikely that performance will improve with a larger vocabulary.

Both training labels and evaluation labels are converted to this vocabulary. If the evaluation labels were kept in the original Harte notation, the model would be unable to identify them.

A simpler chord vocabulary is also sometimes used. This contains only the major and minor quality for each root and the `N` and `X` symbols. For example, `C:maj7` is mapped to `C:maj` while `A:hdim7/5` is mapped to `X`. For this vocabulary, $C = 26$. I did some preliminary tests with this vocabulary but quickly found that model performance was similar over the two vocabularies. Results and analysis can be found in Appendix A.1. Additionally, the `majmin` evaluation metric compares chords over this smaller vocabulary and is mentioned in Section 3.2. The simpler vocabulary is not used in the rest of this work.

The method for converting from Harte notation to a symbol in the chord vocabulary is similar to that used by McFee and Bello [2017] and is detailed in Appendix A.2.

Frames are allocated a chord based on which chord is playing at the middle of the frame. While this may not be a perfect solution, frames are only $\approx$ 93ms long which is longer than the minimum duration of a chord in the dataset. This guarantees that the chord label for every frame is playing for at least half the frame. Furthermore, only 4.4% of frames include a chord transition.

### 3.1.2 Pop Dataset

The *Pop* dataset consists of songs from the *Mcgill Billboard*, *Isophonics*, *RWC-Pop* and *USPop* datasets mentioned in Section 2.2.1. This collection was originally proposed in work by Humphrey and Bello [2015] in order to bring together some of the known datasets for chord recognition. The dataset consists of a subset of the above source filtered for duplicates and selected for those with annotations available. In total, there are 1,217 songs. The dataset was provided with obfuscated filenames and audio as `mp3` files and annotations as `jams` files [Humphrey et al., 2014].

#### 3.1.2.1 Data Integrity

Several possible sources of error in the dataset are investigated below.

**Duplicates:** Files were renamed using provided metadata identifying them by artist and song title. This was done to identify duplicates in the dataset. There was only one: Blondie's 'One Way or Another' which had two different recordings. It was removed from the dataset. Further duplicates may exist under different names but throughout the project no other duplicates were found. Automatic duplicate detection was conducted by embedding each audio using mel-frequency cepstral coefficients (MFCC) [Davis and Mermelstein, 1980]. This function is commonly used to embed audio into low dimensions, and is designed to represent the timbre and shape of a song. It has been used as a basis for audio fingerprinting [Cano et al., 2005]. While far from perfect, this provides a fast and easy way of quantifying similarity. Audio was passed through the `mfcc` provided in `librosa` with 20 coefficients. A song's embedding was calculated as mean MFCC over all frames. Cosine similarities are then calculated for all pairs of tracks. None of the top 50 similarity scores yielded any sign of duplication. I proceed with the assumption that there are no further duplicates in the dataset.

**Chord-Audio Alignment:** It is pertinent to verify that the chord annotations align with the audio. Badly misaligned annotations could make training impossible.

10 songs were manually investigated for alignment issues. This was done by listening to the audio and comparing it to the annotations directly. It became apparent that precise timings of chord changes are ambiguous. The annotations aired on the side of being slightly early but were all well-timed with detailed chord labellings including inversions and upper extensions.

Automatic analysis of the alignment of the audio and chord annotations was also done using cross-correlation of the derivative of the CQT features of the audio over time and the chord annotations. Correlations were calculated with a varying time lag. A maximum correlation at a lag of zero would indicate good alignment as the audio changes at the same time as the annotation. The derivative of the CQT in the time dimension was estimated using `librosa`'s `librosa.feature.delta` function. The chord annotations were converted to a binary vector, where each element corresponds to a frame in the CQT and is 1 if a chord change occurs at that frame and 0 otherwise. Both the CQT derivatives and binary vectors were normalised by subtracting the mean and dividing by the standard deviation. Finally, cross-correlation was computed using `numpy`'s `numpy.correlate` function. A typical cross-correlation for a song is shown in Figure 3.1. We can see that the cross-correlation repeats every 20 frames or so. Listening to the song, we can interpret the period of repetition as some fraction of a bar-length caused by highly correlated drum transients.

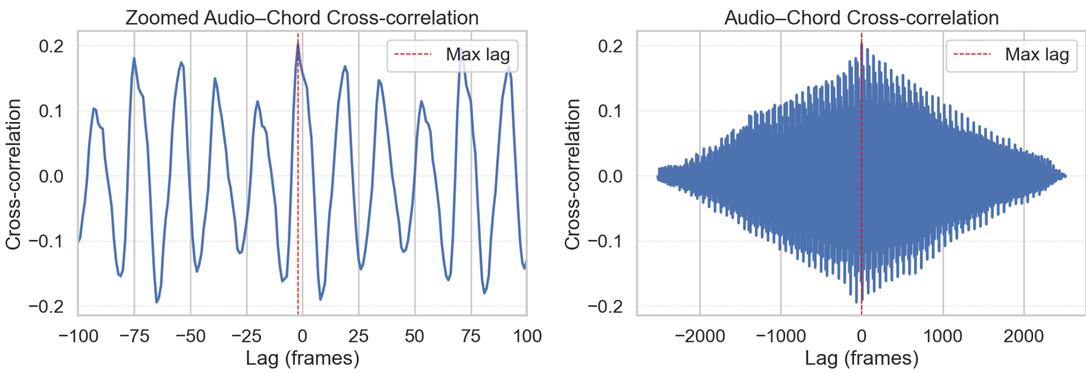


Figure 3.1: Cross-correlation of the derivative of the CQT of the audio and the chord annotations for a single song. We can see correlation peaking in regular intervals of around 20 frames. 1 frame is$\approx$ 0.093ms so 20 frames $\approx$1.86 seconds. Zooming out, we observe peaks in correlation centred around 0.

To check alignment across the dataset, we can plot a histogram over songs of the lag of the maximum cross-correlations. If we further assume that the annotations are not incorrect by more than 5 seconds, we can restrict our maximum correlation search to a window of 100 frames either side of 0. A histogram of maximum-lags per song is shown in Figure 3.2 where the maximum is within a window of 50. This reduction does not change the shape of the picture. Instead, focusing on a reduced set of lags allows more detail to be visible. The majority of songs have a maximum lag close to 0, with a few outliers. This can be attributed to noise. A final check was done by looking at the difference in length of the audio files and chord annotations. A histogram of differences in length is also shown in the figure. The majority of songs have a difference in length of 0, with a few outliers, almost all less than a second. This evidence combined with the qualitative analysis was convincing enough to leave the annotations as they are for training.

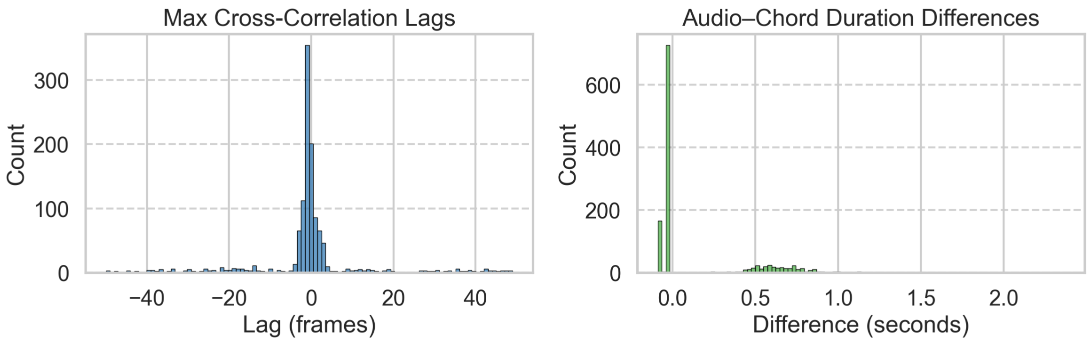


Figure 3.2: Cross-correlation of the derivative of the CQT of the audio and the chord annotations for a single song. The x-axis is the lag in frames and the y-axis is the correlation. The plot repeats every 100 frames, which corresponds to 4 bars.

**Incorrect and Subjective Annotations:** Throughout manual listening, no obviously wrong annotations were found. However, looking at songs which the preliminary models perform the worst on using the `mirex` metric, three songs stick out. 'Lovely Rita' by the Beatles, 'Let Me Get to Know You' by Paul Anka and 'Nowhere to Run' by Martha Reeves and the Vandellas all had scores below 0.05. In these songs, the model consistently guessed chords one semitone off, as if it thought the song was in a different key. Upon listening, it became clear that the tuning was not in standard A440Hz for the first two songs and the key of the annotation was wrong for the other. These songs were removed from the dataset. All reported results exclude these data points. No other songs were found to have such issues.

Chord annotations are inherently subjective to some extent. Detailed examples in *Pop* are given by Humphrey and Bello [2015]. They also note that there are several songs in the dataset of questionable relevance to ACR, as the music itself is not well-explained by chord annotations. However, these are kept in for consistency with other works as this dataset is often used in the literature. Some works decide to use the median as opposed to the mean accuracy in their evaluations in order to counteract the effect of such songs on performance [McFee and Bello, 2017]. We think that this is unnecessary as the effect of these songs is likely to be small and we do not wish to inadvertently inflate our results. Further evidence for use of the mean is given in Section 3.2.

#### 3.1.2.2 Chord Distribution

Much of the recent literature has focused on the long tail of the chord distribution, using a variety of methods to attempt to address the issue. It is first helpful to understand the distribution of chords in the datasets, shown in Figure 3.3. The distribution is broken down both by root and quality, using the chord vocabulary with $C = 170$. The plots show that the distribution over qualities is highly skewed, with major and minor chords making up the majority of the dataset and qualities like majorminor and diminished 7th chords playing for two to three orders of magnitude fewer seconds. Another display over chord qualities can be found in the work by Jiang et al. [2019]. The distribution over roots is far less skewed, although there is a preference for chords in keys with roots at C, D and E and fewer in keys with roots at C# and F#.

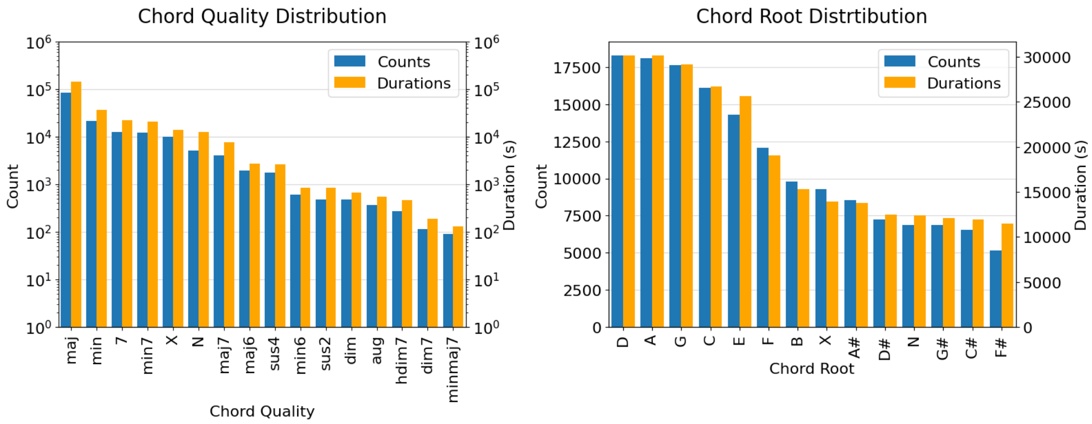


Figure 3.3: Chord distributions in the *Pop* dataset. The plots show both the raw counts in terms of frames and the duration in seconds for each chord root/quality. Note that the y-axis over qualities is in a logarithmic scale. We observe that the qualities are very imbalanced, with `maj` as the most popular. Conversely, roots are relatively balanced.

### 3.1.3 JAAH Dataset

TODO: I was warned by Andrea that the JAAH dataset has not been as commonly used as dataset the Billboard dataset. Therefore he could not guarantee that the audio was aligned for this dataset.

- As yet, JAAH is unused in this work - Data was received as `.flac` files which were first converted to `.mp3` files to be in line with the Billboard dataset - Comparison of the two datasets - Description of the JAAH dataset and its use in this work - Intended to be used as a test set to test the synthetic data generation.

## 3.2 Evaluation

As is standard for ACR, weighted chord symbol recall (WCSR) is used to evaluate classifiers. Simply put, WCSR measures the fraction of time that a classifier's prediction is correct. This is defined in Equation 3.1. Correctness can be measured in a variety of ways such as `root`, `third` and `seventh`, which compare the root, root and third, or root, third and seventh respectively. I also make use of the `mirex` score, where a prediction is correct if it shares at least three notes with the label. This allows for errors like mistaking `C:7` for `C:maj` or `G:maj7` for `E:min`. Finally, I use `acc` to denote the overall accuracy where correctness must be exact.

Other measures of correctness are sometimes used. These include `majmin`, a measure of correctness over only major and minor qualities. I utilise this measure only to substantiate the use of the larger vocabulary in Appendix A.1. Measures of correctness over triads and tetrads are also sometimes used, but these are highly correlated with `third` and `seventh` respectively. This correlation is to be expected as the third and seventh are strong indicators of the triad and tetrad of the chord. This was also verified empirically on preliminary experiments which are omitted for lack of relevance to the discussion.

These are all implemented in the `mir_eval` library [Raffel et al., 2014] which also provides utilities for converting discrete Frames-wise chord outputs to intervals from which WCSR can be calculated.

$$WCSR = 100 \cdot \frac{1}{Z} \sum_{i=1}^{N} \int_{t=0}^{T_i} M(y_{i,t}, \hat{y}_{i,t}) dt \tag{3.1}$$

$$Z = \sum_{i=1}^{N} \int_{t=0}^{T_i} \mathbb{I}_M(y_{i,t}) dt \tag{3.2}$$

where $M(y, \hat{y}) \in \{0, 1\}$ is the measure of correctness which varies across metrics. For example, $M(y, \hat{y})$ for `root` equals 1 if $y$ and $\hat{y}$ share the same root and 0 otherwise. $N$ is the number of songs, $T_i$ is the length of song $i$, $y_{i,t}$ is the true chord at time $t$ of song $i$, and $\hat{y}_{i,t}$ is the predicted chord at time $t$ of song $i$. $Z$ normalises by the length of time for which the metric $M$ is defined. This is necessary as `X` symbols are ignored and `seventh` ignores some qualities. Further details can be found in the `mir_eval` documentation. $\mathbb{I}_M(y_{i,t}) = 1$ if $M$ is defined for label $y_{i,t}$ and 0 otherwise. Finally, we multiply by 100 to convert to a percentage.

For the above metrics, the mean is computed over all songs in the evaluation set. Standard errors and 95% confidence intervals on the means are obtained via bootstrapping. Standard errors are reported for some experiments to provide a sense of the uncertainty in a model's mean performance over songs, not to claim that results are statistically significant.

Some other works report the median. Empirically, I found the median to be $\approx 2\%$ greater than the mean. This may be due to those songs identified as being unsuitable for chordal analysis by Humphrey and Bello [2015]. I report only the mean throughout this work. This was chosen for being more commonly used in recent literature and because we would to measure the model's performance across songs. If the model performs poorly over certain genres or styles, it is important for a metric to capture this.

For some experiments, we look at two more metrics. These are the mean and median class-wise accuracies, called $\texttt{acc}_{\text{class}}$ and $\texttt{median}_{\text{class}}$ respectively. $\texttt{acc}_{\text{class}}$ has previously been defined in terms of discrete frames by Jiang et al. [2019]. I redefine $\texttt{acc}_{\text{class}}$ here in terms of WCSR with a similar notation and introduce $\texttt{median}_{\text{class}}$. The definitions can be found in Equations 3.3 and 3.3.

$$\text{acc}_{\text{class}} = \frac{1}{C} \sum_{c=1}^{C} WCSR(c) \tag{3.3}$$

$$\text{median}_{\text{class}} = \text{median}_{c=1}^{C} [WCSR(c)] \tag{3.4}$$

$C$ denotes the number of chord classes and $WCSR(c)$ is the accuracy of class $c$ is defined in Equation 3.5.

$$\mathrm{WCSR}(c) = \frac{1}{Z_c} \sum_{i=1}^{N} \int_{t=0}^{T_i} M(y_{i,t}, \hat{y}_{i,t}) \cdot \mathbb{I}_c(y_{i,t}) dt \tag{3.5}$$

$$Z_c = \sum_{i=1}^{N} \int_{t=0}^{T_i} \mathbb{I}_M(y_{i,t}) \cdot \mathbb{I}_c(y_{i,t}) dt \tag{3.6}$$

where $N$, $T$, $M$, $y_{i,t}$, $\hat{y}_{i,t}$ and $\mathbb{I}_M(y_{i,t})$ are defined as before in Equation 3.1. $\mathbb{I}_c(y_{i,t})$ is 1 if the true chord at time $t$ of song $i$ is class $c$ abd 0 otherwise. $Z_c$ normalises by the length of time for which the chord $c$ is playing and for which the metric $M$ is defined, in a similar fashion to $Z$ in Equation 3.1.

These metrics are intended to measure the model's performance on the long tail of the chord distribution. It is informative to measure both the mean and median to provide a sense of the skew in performance over classes. While the metric can be defined for any measure of correctness, I report only the `acc` as I found it to be the most informative. For example, the mean class-wise `root` score is harder to meaningfully interpret.

The justification for redefining $\texttt{acc}_{\text{class}}$ is that metrics calculated over discrete frames are not comparable across different frame lengths and are dependent on the method for allocating chords to frames. Further, continuous measures more closely reflect what we truly desire from the model. To illustrate this, imagine an extremely large frame length. The model could have perfect scores on these frames be making terrible predictions for most of the song. Through preliminary experiments, it became clear that with sufficiently small hop lengths, there are negligible differences with continuous measures. Nevertheless, there is no reason the field should not adopt a continuous measure of class-wise accuracy.

I do not also compute *quality*-wise accuracies as seen introduced by Rowe and Tzanetakis [2021]. Compared to class-wise metrics, quality-wise metrics only ensure that each root is equally weighted. As roots as fairly balanced, this would not add much information so I do not include it.

For the majority of experiments, the metrics on the validation set are used to compare performance. The test set is held out for use only to compare the final accuracies of selected models in Section 5.7.

Finally, other evaluation tools were used such as confusion matrices and the average number of chord transitions per song that a model predicts. Note that confusion matrices were calculated using discrete frames for ease of computation. In an ideal setting, these would also be calculated using continuous measures. I decided it was not worth the additional engineering effort and computational cost given the small differences between the discrete and continuous for sufficiently small frame lengths.

## 3.3 Training

Three variants of the dataset are used for training, validation and testing. For training, an epoch consists of randomly sampling a patch of audio from each song in the training set. The length of this sample is kept as a hyperparameter, set to 10 seconds for the majority of experiments. For evaluation, the entire song is used because performance was found to be marginally better if the model was allowed to see the entire song at once. This is later discussed in Section 4.4.4. When validating mid-way through training, songs are split into patches of the same length as the training patches to save on computation time. Samples in a batch are padded to the maximum length of sample in the batch and padded frames are ignored for loss and metric calculation.

Experiments are run on two clusters with some further evaluation taking place locally. The first is The University of Edinburgh's ML Teaching Cluster. Here, NVIDIA GPUs are used - mostly GTX 1080's (10GB VRAM), GTX Titan X's (12GB VRAM) and RTX A6000's (48GB VRAM) depending on the size of experiment and availability on the cluster. Resources have inconsistent availability. Therefore, some experiments are run on The University of Edinburgh's research compute cluster - Eddie. Experiments on Eddie are run on CPUs due to the lack of availability of GPUs.

Training code is implemented in `PyTorch` [Paszke et al., 2019]. Unless stated otherwise, models are trained with the Adam optimiser [Kingma and Ba, 2015] with a learning rate of 0.001 and pytorch's `CosineAnnealingLR` scheduler, set to reduce the learning rate to 1/10th of its initial value over the run. Models are trained to minimise the cross entropy loss between the predicted chord and the true chord distribution. I use a batch size of 64 for a maximum of 150 epochs unless stated otherwise. This batch size was found to complete an epoch faster than other batch size tested. Validation part-way through training is conducted every 5 epochs in order to save on computation time. Optionally, training is stopped early if the validation loss does not improve for 25 epochs. The model was is whenever the validation loss improves. Each training run takes approximately 30 minutes of GPU time or 1 hour 30 minutes of CPU time. This can vary up to 10 hours of CPU time for experiments with more expensive computations and larger input. The `pytorch` seed was set to 0 for all experiments.

For the majority of experiments, a random 60/20/20% training/validation/test split is used. This split is kept constant across experiments. This contrasts much of the literature which uses a 5-fold cross validation introduced by Humphrey and Bello [2015]. I did not maintain this status quo in order to obtain clean estimators of the generalisation error using the held-out test set and to save on computation time. This makes results hard to compare directly to those reported by the literature. For final testing, models are re-trained on the combined training and validation sets and tested on the test set. To test on the *JAAH* dataset, some models are trained on the entire *Pop* dataset.

# Chapter 4

# A Convolution Recurrent Neural Network

In this chapter, I implement a convolutional recurrent neural network (CRNN) from the literature, train it on the *Pop* dataset and compare it to two baselines. I then conduct a thorough analysis of the behaviour and failure modes of the CRNN and provide motivation for improvements.

## 4.1 The CRNN Model

I implement a convolutional recurrent neural network (CRNN) as described in McFee and Bello [2017], referred to as *CRNN*. It remains competitive with state-of-the-art, is often used as a comparative baseline and is relatively fast and easy to train.

The model receives input of size $B \times F$ where $B = 216$ is the number of bins in the CQT and $F$ is the number of frames in the song. The input is passed through a layer of batch normalisation [Ioffe and Szegedy, 2015] before being fed through two convolutional layers with a rectified linear unit (ReLU) after each one. The first convolutional layer has a $5 \times 5$ kernel and outputs only one channel of the same size as the input. It is intended to smooth out noise and spread information about sustained notes across adjacent frames. The second layer has a kernel of size $1 \times I$ and outputs 36 channels, intended to collapse the information over all frequencies into a single 36-dimensional vector. This acts as a linear layer across frames with shared parameters for each frame. The output is passed through a bi-directional GRU [Cho et al., 2014], with hidden size initially set to 256 and a final dense layer with softmax activation. This produces a vector of length $C$ for each frame. The model is trained using the cross-entropy loss with the true chord distribution.

The authors of the model also propose using a second GRU as a decoder before the final dense layer, called 'CR2'. However, a similar effect could be achieved with more layers in the initial GRU. Furthermore, both in the paper and in brief empirical tests, the results with 'CR2' were indistinguishable from the model without it. I therefore do not include this addition in model. Results left to Appendix A.3 as they are neither relevant nor interesting.

### 4.1.1 Hyperparameter Tuning

To ensure that the training hyperparameters are set to reasonable values, I conduct a grid search over learning rates and learning rate schedulers. This is followed by a random search over model hyperparameters.

#### 4.1.1.1 Learning rates

I perform a grid search over learning rates and learning rate schedulers in the set `[0.1, 0.01, 0.001, 0.0001]` and `[cosine, plateau, none]` respectively. `cosine` is as described in Section 3.2 and `plateau` reduces the learning rate to half its current value when validation loss has not improved for 10 epochs and stops training if it has not improved for 25 epochs.

I report a subset of metrics in Table 4.1. The best performing model by validation metrics was found to be with `lr=0.001`, and very similar results over the different schedulers. I proceed with a learning rate of 0.001 and `cosine` scheduling for the rest of the experiments.

Early stopping is disabled in order to check for convergence and overfitting without the possibility of a pre-emptive stop. Judging by training graphs seen in 4.1, the best learning rate is 0.001. Any lower and we do not converge fast enough. Any higher and large gradient updates cause the validation accuracy to be noisy. These figures also show that the validation loss does not increase after convergence. I conclude that the model is not quick to overfit, perhaps due to the random sampling of audio patches during training. Combined with the fact that training is relatively quick and that the model is only saved on improved validation loss, I decided to remove early stopping.

Future experiments are all conducted without early stopping, for a total of 150 epochs and with a learning rate of 0.001 and `cosine` scheduling.

In order to check that Adam is the best optimiser to use, a training run with stochastic gradient descent (SGD) was also carried out. Results from Reddi et al. [2018] and Wilson et al. [2017] suggest that SGD can find better minima with a stable learning rate over many epochs. To test this, I trained a CRNN over 2000 epochs with a learning rate of 0.001, the `cosine` scheduler and momentum set to 0.9. While the model did converge, it did not perform any better than the models trained with Adam. Results are left to Appendix A.4 for lack of interest.

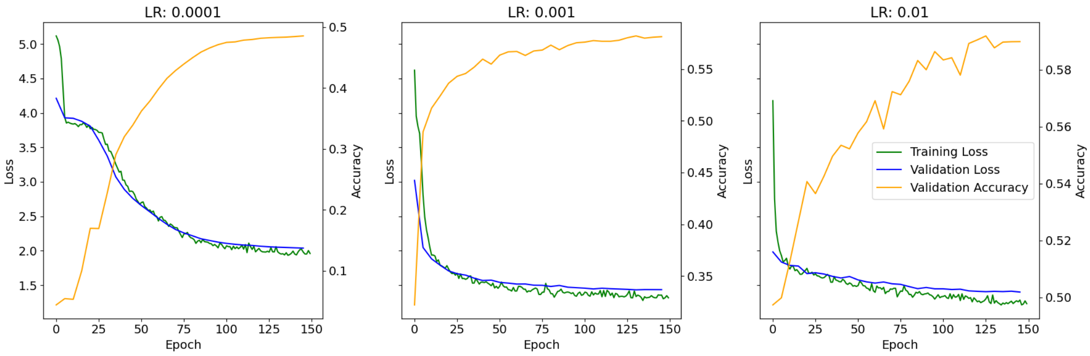


Figure 4.1: Training graphs for *CRNN* with different learning rates and the `cosine` scheduler. The learning rate of $0.001$ seems to be the best, as it converges in a reasonable time and the validation accuracy increases in a stable fashion. While it may seem that running for more epochs may increase performance, this was not found to be the case empirically. The best model was often achieved around epoch 100.

| lr | scheduler | acc | root | third | seventh | mirex |
|---|---|---|---|---|---|---|
| 0.01 | Cosine | 53.6 | 69.5 | 66.9 | 55.7 | 78.6 |
| 0.001 | Cosine | *59.7* | *78.3* | *75.0* | *62.0* | ***79.8*** |
| 0.0001 | Cosine | 53.2 | 72.1 | 66.9 | 55.2 | 72.0 |
| 0.001 | Plateau | **59.9** | 78.4 | 75.2 | **62.2** | 79.7 |
| 0.001 | None | 59.8 | **78.7** | **75.5** | 62.0 | 78.8 |

Table 4.1: *CRNN* model results with different learning rates and schedulers. Best results over learning rates are *italicised* and best results over schedulers are in **boldface**. A learning rate of $0.001$ performs the best on all metrics. The differences between learning rate schedulers are so small that the choice between them is arbitrary.

#### 4.1.1.2 Model Hyperparameters

With this learning rate and learning rate scheduler fixed, I perform a random search over the number of layers in the GRU, the hidden size of the layers in the and the training patch segment length. I also experiment increasing the number of convolutional layers prior to the GRU, the kernel size of these layers and the number of channels outputted by each of these layers. The search is performed by independently and uniformly randomly sampling 50 points over discrete sets of possible hyperparameter values. These sets can be found in Appendix A.5.

A sample of the results are shown in Table 4.2. The models were ranked according to each metric and their ranks for each metric were added up. The models were ordered by this total rank. The best model was found to have a hidden size $h = 231$, a single layer GRU and a segment length of $L = 23$ and the same single convolutional layer of $5 \times 5$ outputting a single channel. In general, increased complexity hurts model performance but the differences between models are relatively small. Such small differences are indicative that the model is learning something simple and that increased model complexity would not help. It also seems that the choice of hyperparameters is somewhat arbitrary. The model does not utilise information from a larger context. Deeper layers with more parameters do not help performance. In fact, the parameters suggested by McFee and Bello [2017] perform just as well as the best performing hyperparameter selection found in this random search. I therefore proceed with the same hyperparameters suggested by McFee and Bello [2017] as default for the remainder of experiments.

| $L$ | $h$ | $k$ | $c$ | $g$ | $ch$ | acc | root | third | seventh | mirex |
|---|---|---|---|---|---|---|---|---|---|---|
| 23 | 231 | 5 | 1 | 1 | 1 | **59.8** | 78.2 | 74.7 | **62.0** | **79.9** |
| 11 | 150 | 7 | 2 | 1 | 3 | 59.6 | 78.5 | 75.0 | 61.9 | 79.2 |
| 43 | 222 | 11 | 2 | 2 | 2 | 59.5 | **78.7** | **75.2** | 61.8 | 78.9 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 34 | 159 | 14 | 4 | 2 | 1 | 56.9 | 75.5 | 72.2 | 59.1 | 77.7 |

Table 4.2: A subset of *CRNN* model results on the large vocabulary with different hyperparameters. Best results for each metric are in **boldface**. $L$ is the length of training patches of audio in seconds, $h$ and $g$ are the hidden size and number of layers in the GRU respectively and $k$, $c$ and $ch$ are the kernel sizes, number of layers and number of channels in the CNN respectively. Models are ordered by their 'Rank', calculated by adding the model's rank order over each metric, and ordering by this total. Results across most hyperparameters are very similar. Comparing with the best results from the learning rate search in Table 4.1, it seems that the parameters suggested by McFee and Bello [2017] are good choices. In general, models with more parameters and longer input tend to perform worse, perhaps due to overfitting. This suggests that the model is learning something simple.

## 4.2 Baseline Models

I consider two models as baselines. First, I train a single layer neural network with softmax activation which treats each frame of each song independently. The layer receives an input of size $B = 216$ outputs a $C = 170$-dimensional vector for each frame. Finally, the cross-entropy loss with the true chord distribution is calculated. This model is called *logistic* as it can be viewed as a logistic regression model trained using SGD. I could have used a logistic regression model implemented in `sklearn` but the implementation as a neural network was fast and easy to implement and unlikely to yield significantly different results.

Secondly, I train a convolutional neural network (CNN). The number of convolutional layers, kernel size and number of channels are left as hyperparameters. The convolutional layers operate on the CQT similarly to how a convolution operates on an image. A ReLU acts upon the activations between each layer. These convolutional layers are followed by ths same 36-channeled $I \times 36$ and final dense layer as in *CRNN*.

I tested models of increasing depths, kernel sizes and channels. In general, the deeper models performed better. Two of these models serve as baselines in reported results.

The first model has a single layer and channel and a kernel size of 5. It serves as an ablation on the GRU part of *CRNN*. This configuration is referred to as *CNN1*. A second model with 5 layers of kernel size 9, each with 10 channels, is referred to as *CNN5*.

I performed a grid search over learning rates and schedulers for these baselines to ensure that convergence was reached. Convergence results were not meaningfully different than those obtained with the CRNN and are hence omitted. I use the best performing results in each case. This was with a learning rate of 0.001 for both models and with schedulers of `plateau` and `cosine` for *logistic* and *CNN1*/*CNN5* respectively.

## 4.3 Results

Table 4.3 shows the results of *CRNN* compared with the baseline models.

*CRNN* performs the best out of these models. The GRU layer increases `root` accuracy by 5.2%. However, similar performance increases can be achieved by adding convolutional layers as in *CNN5* rather than an RNN. Combined with the lack of performance improvement from increasing the audio patch length observed in Section 4.1.1.2, there is strong evidence that the model does not share information across time very far.

We also observe diminishing performance increases with increased model complexity. Performance begins to level out with accuracies of only around 60%. Indeed, the best models trained by Park et al. [2019] and by Akram et al. [2025] never achieve an accuracy of more than 66%. Humphrey and Bello [2015] refer to this as the 'glass ceiling' through which the field of ACR is still struggling to break through. The problem is very far from solved.

Korzeniowski and Widmer [2016a] train a deep CNN which remains competitive with state-of-the-art to this day. It contains 8 layers. Park et al. [2019] find that the performance of this deep CNN is very similar to that of *CRNN*, both reaching `root` recalls between 81 and 82. Training much deeper convolutional networks was found to be far more computationally expensive than training *CRNN*, with little performance gain to be had. Therefore, I proceed with *CRNN* for further experiments.

| model | acc | root | third | seventh | mirex | $\text{acc}_{\text{class}}$ | $\text{median}_{\text{class}}$ |
|---|---|---|---|---|---|---|---|
| *logistic* | 43.0 | 64.5 | 56.9 | 44.7 | 60.9 | 12.0 | 1.7 |
| *CNN1* | 54.5 | 74.4 | 69.0 | 56.6 | 73.5 | 16.0 | 2.3 |
| *CNN5* | 57.8 | 78.1 | 74.0 | 60.0 | 77.8 | 19.2 | **3.2** |
| *CRNN* | **59.7** | **78.3** | **75.0** | **62.0** | **79.8** | **19.6** | 2.3 |

Table 4.3: Results for *logistic*, *CNN1*, *CNN5* and *CRNN*. We see that *CRNN* performs the best on nearly all metrics. *CNN5* performs almost as well. This suggests that shallower CNNs can reach similar performance as the deep CNN trained by Korzeniowski and Widmer [2016a].

## 4.4 Model Analysis

While quantitative metrics summarise how well a model performs over songs, they do not tell us much about the predictions the model makes and where it goes wrong. In this section, I answer several questions about the behaviour of the model.

### 4.4.1 Qualities and Roots

**How does the model deal with the long tail of the chord distribution?** The class-wise metrics in Table 4.3 give strong indication that the performance is poor. I use a confusion matrix over qualities of chords to provide more granular detail.

The confusion matrix is illustrated in Figure 4.2. The model struggles with rarer chords. On the rarest quality of majorminor7, the model has a recall of 0. Recall is 0.86 on the major chord but consistently predicts major for similar chord qualities like major7, major6,sus4 and sus2. A similar effect is observed on minor and similar chord qualities. The model also confuses diminished 7 chords for diminished chords. This explains the median class-wise accuracies of nearly 0 for all models.

There are many methods of addressing an imbalanced distribution in machine learning. The simplest is to add a weighting to the loss function which I explore in Section 5.3.1. I also look at a 'structured' loss function which exploits similarity between chords in Section 5.3.2. Performance can also be improved by improving the data. I explore the use of data augmentation in Section 5.5 and synthetic data generation in Section 5.6.

I also produced a confusion matrix over roots. This is left to Appendix A.6 as it is less insightful. The model performs similarly over all roots with a recall of between 0.73 and 0.81. This is not surprising as the distribution over roots is relatively uniform, previously seen in Figure 3.3. Recall on the no chord symbol `N` is 0.73. Many of the `N` chords are at the beginning and end of the piece. The model may struggle with understanding when the music begins and ends. An example of the model erroneously predicting that chords are playing part-way through a song is discussed in Section 4.4.5.

Performance is much worse on the unknown chord symbol with a recall of 0.18. The low performance on `X` is to be expected. It is a highly ambiguous class with many very different sounds that are mapped to it. All of the chords mapped to `X` will share many notes with at least one class in the known chord portion of the vocabulary. Therefore, it is unreasonable to expect the model to be able to predict this class well. This supports the case for ignoring this class during evaluation as is standard in the literature.

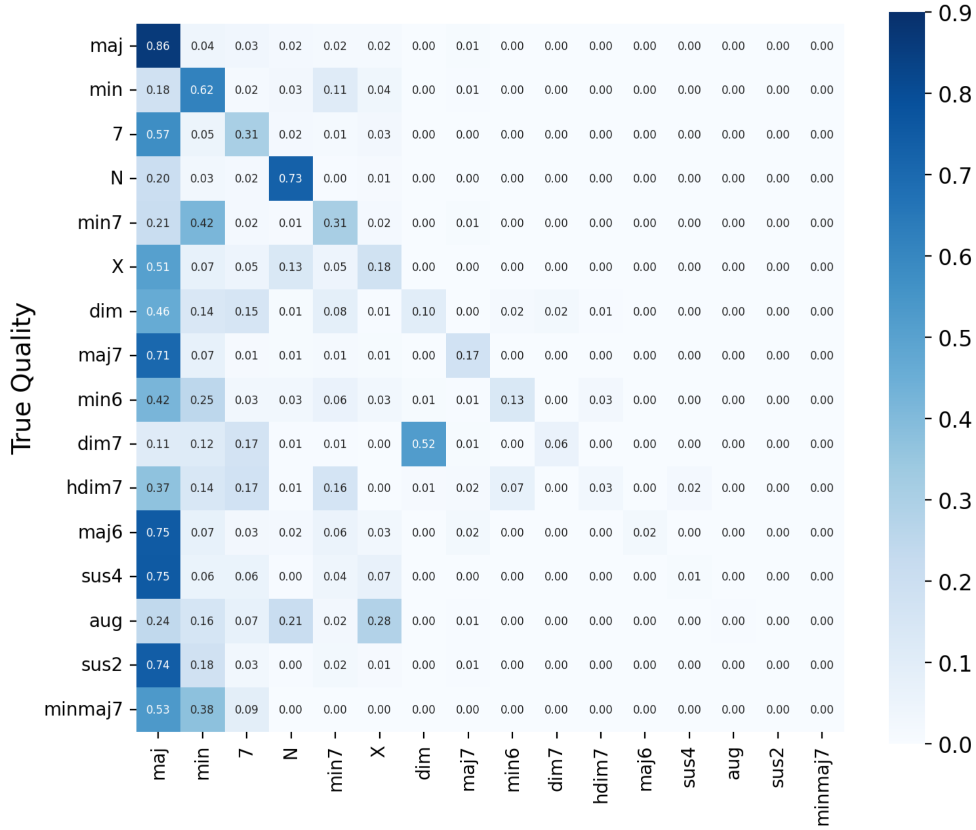


Figure 4.2: Row-normalised confusion matrices over qualities of the *CRNN* model without (above) and with (below) a weighted loss, with $\alpha = 0.3$ as described in Section5.3.1. Rows are ordered by frequency of chord quality. We can see that both models struggle with the imbalanced distribution. However, weighting the loss does improve the model, notably doubling on the `maj7` quality and predicts `maj` less often. Recall on the `maj` worsens by $0.07$ and recall on `X` decreases from $0.18$ to $0.09$. The weighted model predicts `X` $2.2$ times less often. This may be how the weighted model improves class-wise metrics without sacrificing too much overall accuracy since `X` frames are ignored for evaluation. We can also see that both models frequently confuse `dim7` and `dim` qualities, consistently predict `maj` for `sus2, sus4, maj6, maj7` and struggles with the two rare qualities of `minmaj7` and `aug`.

### 4.4.2 Transition Frames

**Are predictions worse on frames where the chord changes?** Such *transition frames* are present because frames are calculated based on hop length irrespective of the tempo and time signature of the song. Thus, some frames will contain a chord transition.

To test this, I compute accuracies for transition and non-transition frames separately. The model achieves only 37% on the transition frames. compared with 61% on non-transition frames. Therefore, the model is certainly worse at predicting chords on transition frames. Nonetheless, *CRNN* achieves an overall accuracy of 60%. This is because only 4.4% of frames are transition frames with a hop length of 4096. Improving performance on these frames to the level of non-transition frames would increase the overall frame-wise accuracy by much less than 1%.

Through qualitative evaluation discussed in Section 4.4.5, the model was found to struggle with identifying the boundary of a chord change on some songs. This would not be captured by the above metrics if the boundary is ambiguous enough to span multiple frames. Thus, there may be a larger impact in accuracy than a single frame. Furthermore, the ambiguity of chord transition timing will vary over songs. For some songs, this may be the main limiting factor in performance.

### 4.4.3 Smoothness

**Are the models outputs smooth?** There are over 10 frames per second. If many of the model's errors are due to rapid fluctuations in chord probability, the model will over-predict chord transitions. I use two crude measures of smoothness to answer this question.

Firstly, I look at the number and length of *incorrect regions*. Such a region is defined as a sequence of incorrectly predicted frames with the same prediction. 26.7% of all incorrect regions are one frame wide and 3.7% of incorrect frames have different predictions on either side. This can suggests that at least 3.7% of errors are caused rapidly changing chord predictions. A histogram over incorrect region lengths can be found in Appendix A.7. This plot shows that distribution of lengths of incorrect regions is long-tailed, with the vast majority very short.

Secondly, I compare the mean number of chord transitions per song predicted by the model with the true number of transitions per song in the validation set. The model predicts 168 transitions per song while the true number is 104. This is convincing evidence that smoothing the outputs of the model could help.

With these two observations combined, I conclude that further work on the model to improve the smoothness would might performance. Although we might hope to improve on at least 3.8% of errors, this would not improve overall accuracy very much. While rapid changes may be smoothed out, there is no guarantee that smoothing will result in correct predictions. Indeed, it may even render some previously correct predictions erroneous. Nonetheless, the model is clearly over-predicting transitions in general and when being used by a musician or researcher, smoothed predictions are valuable to make the chords more interpretable. This motivates the exploration of a 'decoding' step in Section 5.2.

### 4.4.4 Performance Across the Context

**How much does the model rely on context?** I hypothesise that the model is worse at predicting chords at the beginning and end of a patch of audio as it has less contextual information close to these frames.

To test this, I evaluate the model using the same fixed-length validation conducted during training as described in Section 3.3. Average frame-wise accuracies over the context are then calculated. A plot can be found in Appendix A.8. I use a segment length of 10 seconds corresponding to $L = 107$ frames. We observe that performance is worst at the beginning and end of the patch but not by much. Performance only dips by 0.05 at either extreme, perhaps because the model still does have significant context on one side. We can also see that performance starts decreasing 5 or 6 frames from either end, suggesting this is extent to which bidirectional context is useful.

I conducted a further experiment, measuring overall accuracy with increasing segment lengths used during evaluation. Results can found in Appendix A.9. The plots show that accuracy increases by 0.5 after increasing the segment length from 5 seconds to 60 seconds. Although this is not much of an increase, it confirms that it is better to evaluated over the entire song at once.

The predictions of the *logistic* baseline use no context at all and yet achieve an accuracy of 43%. *CNN1* only shares context a maximum of 5 frames either side as this is its kernel size. It achieves an accuracy of 54.5. This suggests that the majority of the performance gain associated with including contextual information is not complex nor far-reaching. Together with these experiments, I conclude that while context improves performance, it does not use context in a complex manner.

#### 4.4.4.1 Generalising Across Songs

**Does the model have consistent performance over different songs?** The set of accuracies over songs of *CRNN* has a standard deviation of 13.5. This suggests that performance is not stable over songs. To provide further insight, I plot a histogram of accuracies and `mirex` scores over the validation set in Figure 4.3. We observe that the model has mixed performance with accuracy, with 15% of songs scoring below 40%.

When we use the more generous `mirex` metric, there are very few songs below 40% and only 7% are below 0.6. This large discrepancy between accuracy and `mirex` suggests that many of the mistakes that the model makes are small. This mistakes are a 'good guess' in the sense that the prediction may have omitted a seventh or mistaken a major 7 for its relative minor. Examples of such mistakes are discussed in Section 4.4.5.

I conclude that many of the model's predictions are reasonable but often lack the detail contained in good annotations like correct upper extensions. Whether these reasonable guesses are correct can vary widely over songs.

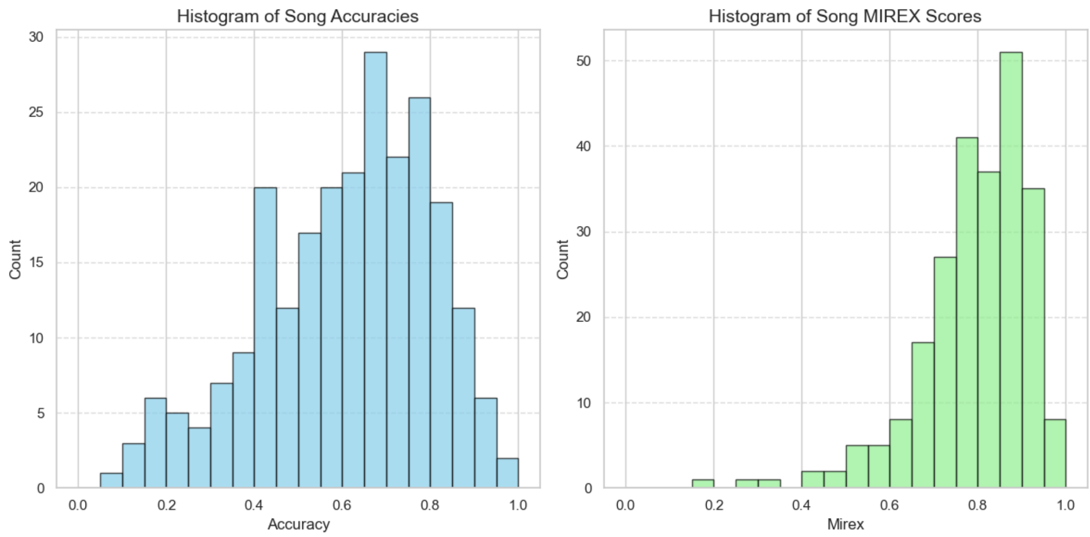


Figure 4.3: Histogram of accuracies and mirex scores over songs in the validation set. Accuracies are mixed, with $15\%$ of songs below $40\%$, and $69\%$ between $0.4$ and $0.8$. However, with the more generous `mirex` metric, we find that there are almost no songs below a score of $40\%$ and only $7\%$ below $0.6$. This suggests that many of the mistakes the model makes are small, like predicting `C:maj` instead of `C:maj7`. The very low outliers in the `mirex` score were found to be songs with incorrect annotations found in Section 3.1.2.1.

### 4.4.5 Four Illustrative Examples

Let us now inspect a few songs to see how the model performs. We choose four examples showing different behaviours and failure modes of the model. We show illustrations of frame-by-frame correctness as measured by both accuracy and `mirex` in Figure 4.4.

In 'Mr. Moonlight', there are few differences between the accuracy and mirex. There are regular repeated errors, many of which are mistaking `F:sus2` for `F:maj`. This is an understandable mistake to make, especially after hearing the song and looking at the annotation where the main guitar riff alternates between `F:maj` and `F:sus2`. As evidenced by the confusion matrices in Figure 4.2, this mistake is very fairly common on qualities like `sus2` which are similar to `maj`.

In 'Ain't not Sunshine', the mirex is significantly higher than the accuracy. This is because the majority of the mistakes the model makes are missing out a seventh. For example, the model predicts `A:min7` for the true label of `A:min7` or `G:maj` for `G:7`. Other mistakes that mirex allows for include confusing the relative minor or major, such as `E:min7` for its relative major `G:maj`. All of these mistakes occur frequently in this song. The mean difference between the accuracy and mirex is 0.2, with one song reaching a difference of 0.9. Hence, we can attribute many of the model's mistakes to such behaviour. 'Ain't no Sunshine' also contains a long incorrect section in the middle. This is a section with only voice and drums which the annotation interprets as `N` symbols but the model continues to predict harmonic content. The model guesses

`A:min`, which is a sensible label as when this melody is sung in other parts of the song, `A:min7` is playing. Examples like this combined with incorrect predictions of when a song starts and ends explain why the mode's recall on the `N` class is only 0.63.

In the next two songs, 'Brandy' and 'Earth, Wind and Fire', the model's mistakes are less interpretable. While performance is okay on 'Brandy' with a `mirex` of 0.74, the model struggles with the boundaries of chord changes resulting in sporadic short incorrect regions in the figure. In 'Earth, Wind and Fire', the model struggles with the boundaries of chord changes and also sometimes predicts completely wrong chords which are harder to explain. Listening to the song and inspecting the annotation makes it apparent that this is a difficult song for even a human to annotate well and similarly the model does not fare well.

Despite these mistakes, the average mirex over the validation set is 0.79 while the accuracy is 0.58. The examples above highlight the models' errors but the model fares well with many songs. We conclude that the majority of the model's outputs are reasonable predictions but that many lack the detail contained in good annotations like correct upper extensions of the chords. The model consistently confuses qualities that can be easily mistaken for major or minor chords. Sometimes the model makes mistakes on the boundaries of chord changes, and sometimes it predicts completely wrong chords, although these are on songs which are more difficult to annotate for a human too.

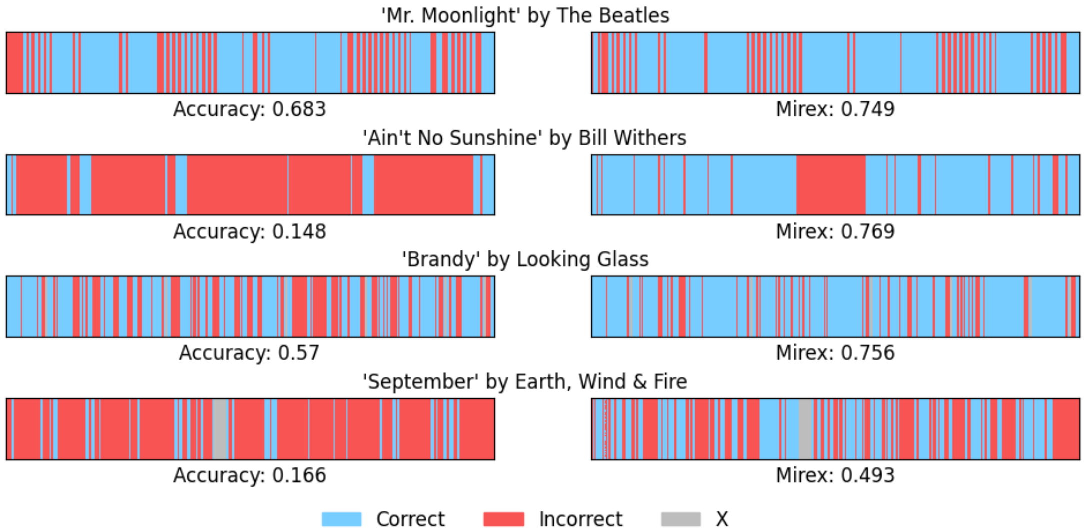


Figure 4.4: Chord predictions of the *CRNN* model on four songs from the validation set (blue: correct, red: incorrect, gray: `X`). This allows us to understand some of the behaviour of the model. We can see regular repeated errors in 'Mr. Moonlight', which are mostly mistaking two similar qualities. The discrepancy between accuracy and `mirex` on 'Ain't No Sunshine' can be explained by missing sevenths in many predictions. The large incorrect region is a voice and drum only section where the model continues to predict chords due to implied harmony by the melody. Predictions in 'Brandy' are quite good in general, though many errors arise from predicting the boundaries of chord changes incorrectly. The model struggles with 'Earth, Wind and Fire', missing chord boundaries, and sometimes predicting completely wrong chords. There are clearly songs where the model's outputs are less sensible. However, in general most of the model's mistakes can be explained and are reasonable.

# Chapter 5

# Improving the Model

## 5.1 Revisiting the Spectrogram

### 5.1.1 Hop Lengths

### 5.1.2 Spectrogram Variants

## 5.2 Decoding

As observed in 4.4.3, taking the maximum probability over each frame results in 170 transitions per song as opposed to the 104 seen in the ground truth data. We implemented a decoding step over the frame-wise probability vectors to smooth predicted labels. Common choices for decoding models include a conditional random field (CRF) [Jiang et al., 2019, Park et al., 2019] and a hidden Markov model (HMM) [Miller et al., 2022].

For the sake of simplicity, we first implemented an HMM and found it to be able to smooth chords predictions well. The HMM treats the frame-wise probabilities as emission probabilities and the chord labels as hidden states. O'Hanlon and Sandler [2019] note that using a transition matrix with all non-recurrent transitions equally likely performs similarly to using a learned transition matrix. We adopt such a transition matrix for our HMM, with a parameter $\beta$ denoting the self-transition probabilities, and all other transition probabilities equal to $\frac{1-\beta}{C-1}$. We then compute a forward and backward pass of the Viterbi algorithm to output the most likely sequence of chords.

A plot of the effect of $\beta$ on the model's performance and the number of transitions per song is shown in Figure 5.1. From this plot we conclude that smoothing has little affect on `root` while successfully reducing the number of transitions per song to that of the true labels. We choose $\beta = 0.2$ as it results in 102 transitions per song while maintaining high performance.

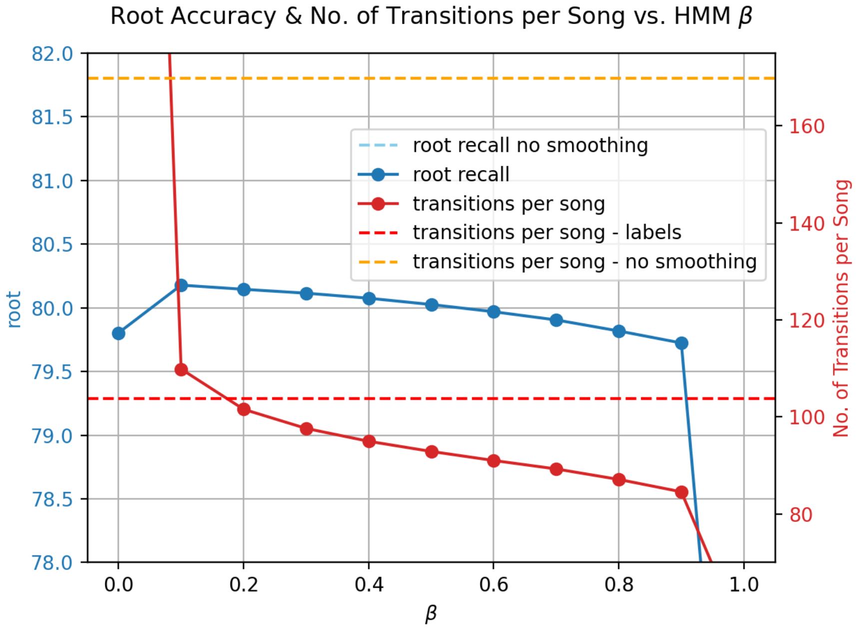


Figure 5.1: Effect of the HMM smoothing parameter $\beta$ on the *CRNN* model. As we increase $\beta$, the number of transitions per song decreases. We choose $\beta = 0.2$ as it results $102$ transitions per song, very close to the $104$ of the ground truth. Performance is stable across $\beta$ with a slight degradation for $\beta > 0.3$. Other performance metrics showed similarly stable results.

The effect of the HMM on the incorrect regions previously discussed in Section 4.4.3 can be found in Appendix A.7. The HMM reduced the percentage of incorrect regions which are a single frame long from 26.7% to 16.7%. A more intuitive way to see the effect of the HMM is to look at a section of a song which was the model previously predicted many chord transitions for. We show this in Figure 5.2.

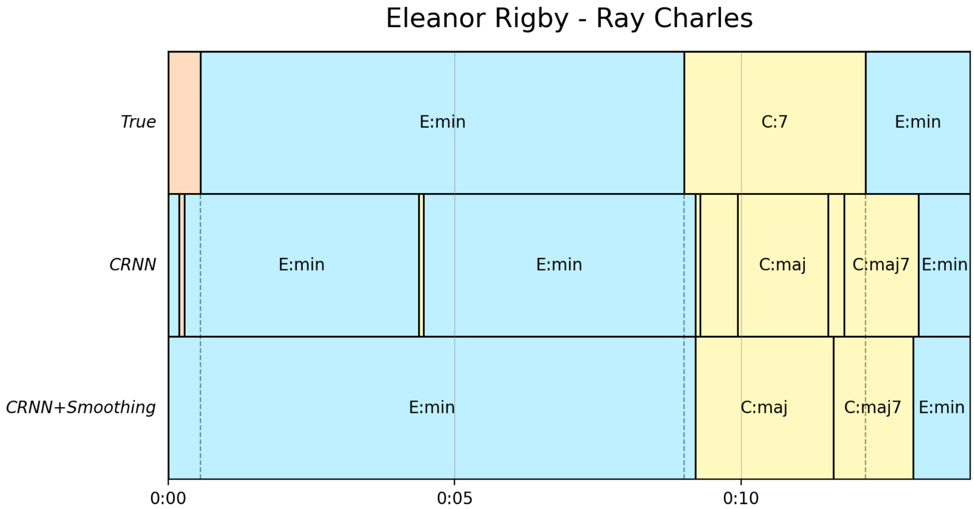


Figure 5.2: An example of the effect of the HMM on the *CRNN* model. The top plot shows the ground truth. The middle plot shows frame-wise predictions of the *CRNN* without smoothing. The bottom plot shows the predictions after smoothing. Chords are coloured by their equivalent chord in the small vocabulary as it makes the plot easier to interpret. The original predictions contain many unnecessary and nonsensical chord transitions. These have been smoothed out by the HMM. The resulting chords appear more similar to the ground truth even if frame-wise accuracy has not changed much.

We did not implement a CRF. All related works to use a CRF use a linear chain CRF with either hand-crafted transition penalties or learned transition penalties. We believe that such simple CRF's are unlikely to outperform the HMM given the effective smoothing of the HMM and do not explore further. We also do not wish to bias the model towards the transitions contained in the dataset. Given the long-tailed distribution, a learned transition matrix will further encourage the model to predict more common transitions. Thus, we believe that the HMM with a simple transition matrix which effectively smooths the predictions is a satisfactory solution.

## 5.3 The Loss Function

### 5.3.1 Weighted Loss

One of the biggest problems highlighted above is low recall on less common qualities. Two common methods for dealing with long-tailed distributions are re-sampling and weighting the loss function. Rowe and Tzanetakis [2021] also explore the use of curriculum learning as form of re-sampling which we do not explore here. Sampling is explored by Miller et al. [2022] but they use a different model based on pre-computing chroma vectors and re-sampling these chroma vectors for use in training a random forest for frame-wise decoding. In our setting, re-sampling training patches of audio may be interesting but is left as future work as it would require significant effort to manage sampling many chords at once. Weighting has been explored by Jiang et al. [2019] however their weighting is over chord classes and chord 'components' which they define in their work. We employ a similar but simpler implementation here.

TODO: Cite Reweighting vs Resampling and claim it won't make a big difference.

A standard method of weighting is to multiply the loss function by the inverse of the given class' frequency, with a parameter controlling the strength of the weighting. This is defined as below.

$$w_c = \frac{1}{(\text{count}(c)+1)^\alpha} \tag{5.1}$$

Where $w_c$ is the weight for chord $c$, $\text{count}(i)$ is the number of frames with chord $c$ in the dataset and $\alpha$ is a hyperparameter controlling the strength of weighting. $\alpha = 0$ results in no weighting and increasing $alpha$ increases the severity of weighting. We add 1 in the denominator to avoid dividing by 0 and to diminish the effect of chords with very few occurrences. We then define normalised weights $w_c^*$ below.

$$w_c^* = \frac{w_c}{s} \text{ where } s = \frac{\sum_{c \in \mathcal{C}} \text{count}(c) \cdot w_c}{\sum_{c \in \mathcal{C}} \text{count}(c)} \tag{5.2}$$

Where $\mathcal{C}$ is the set of all chords in the vocabulary. This keeps the expected weight at 1 such that the effective learning rate remains the same. We calculate these values over the training set. We test values of $\alpha$ in the set $\{0, 0.05, 0.1, \ldots, 0.95, 1\}$. The plot in Figure 5.3 shows the effect of the weighting on the model's performance.

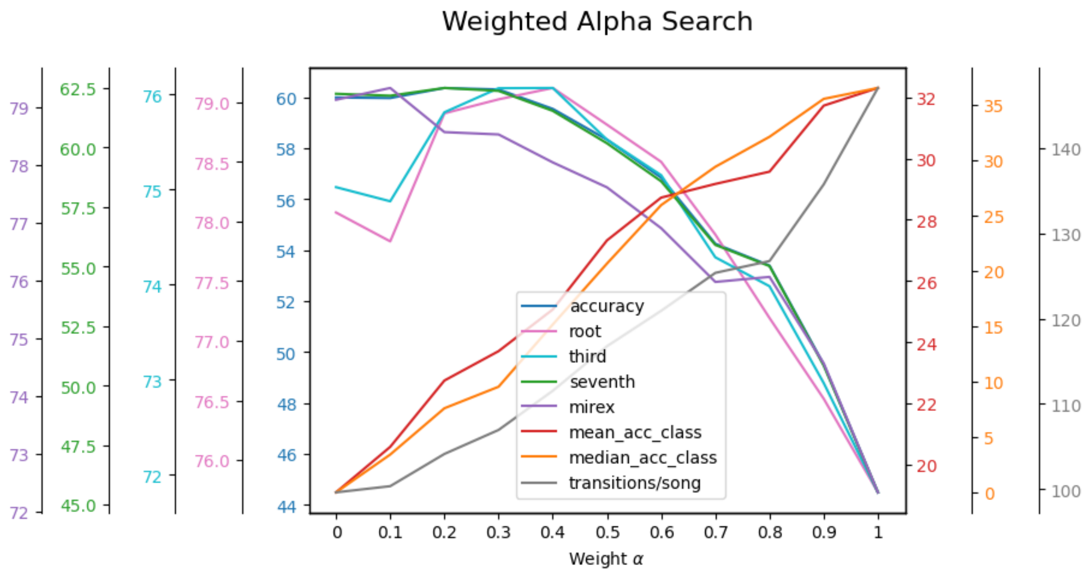


Figure 5.3: Effect of weighted loss on the *CRNN* model with varying $\alpha$. As we increase $\alpha$, class-wise metrics improve but accuracy-based metrics worsen. We claim a sweet-spot in the middle where we trade only a little overall performance for better class-wise recall. We choose this to be $\alpha = 0.55$. The `root` and `third` metrics improve and less than $3\%$ is lost on other metrics while mean class-wise accuracy improves by $6\%$ and the median improved by $0.2$. This plot also reveals strong correlation between metrics.

### 5.3.2 Structured Loss

## 5.4 Generative Features

- As in [MelodyTranscriptionViaGenerativePreTraining], use Jukebox (?) to generate features at frames (? is it possible to do it on the same frames?). Train on these features and evaluate.

## 5.5 Pitch Augmentation

- Two methods: - On CQT Jiang et al. [2019], not good. - Using `pyrubberband`[1] on the audio [everyone else], works?

Pitch augmentation has been done in other works on chord recognition. This has been done on the CQT [Jiang et al., 2019] by shifting the CQT bins and directly on the audio [Park et al., 2019, McFee and Bello, 2017]. These are not the same process. Shifting the CQT is a simple matrix operation, whereas pitch shifting introduces other artefacts due to

## 5.6 Synthetic Data Generation

Motivation

#### 5.6.0.1 Generation method

- Generation method.

#### 5.6.0.2 Experiments

- Brief description of the experiments and metrics I'm looking at

#### 5.6.0.3 Results

- Results of the experiments on the validation set

## 5.7 Results on the Test Set

- Directly compare CRNN, weighted loss, pitch augmentation, structured, transformer, generative features, generated data and any meaningful combination on the test set. - Also compare to BTC as a transformer model.

### 5.7.1 Performance on JAAH

- The performance of existing models on JAAH

[1] `https://github.com/bmcfee/pyrubberband`

## 5.8 Qualitative Analysis

- Qualitative analysis of the results

# Chapter 6

# Conclusions, Limitations and Further Work

## 6.1 Conclusions

- What do the results say? What did we find?

## 6.2 Limitations

Limitations: - Genre - Standard tuning, Western - No lyrics - Size of vocabulary: inversion etc - Some labels don't have a clear meaning

## 6.3 Further Work

Further work: - More detailed expts: - more models for gen features, more varied, with vocals etc - better future chord conditioned models for synthetic data - More data e.g. HookTheory - Better beat tracking? - Jointly predicting chord segmentation, cite 20 years and cite Choco people for inverse problem.

- Incorporate functional harmony or chord vectors as targets - Better understanding of the glass ceiling, human inter-annotator scores.

Finally, chord annotations are inherently subjective. Inter-annotator agreement of the root of a chord is estimated at lying between 76% [Ni et al., 2019] and 94% [De Clercq and Temperley, 2011] but these metrics are calculated using only four and two annotators respectively. These agreement estimates use the same metrics defined in Section 3.2. Furthermore, Humphrey and Bello [2015] and Harte and Sandler [2010] posit that agreement between annotations can be far lower for some songs. Little has been done to address

# Bibliography

Muhammad Waseem Akram, Stefano Dettori, Valentina Colla, and Giorgio Carlo Buttazzo. Chordformer: A conformer-based architecture for large-vocabulary audio chord recognition, 2025. URL `https://arxiv.org/abs/2502.11840`.

Edward Aldwell, Carl Schachter, and Allen Cadwallader. *Harmony and Voice Leading*. Cengage Learning, 2010. ISBN: 9780495189756.

Adam Berenzweig, Beth Logan, Daniel P. Ellis, and Brian Whitman. A large-scale evaluation of acoustic and subjective music-similarity measures. *Computer Music Journal*, 28(2):63–76, 2004.

Judith Brown. Calculation of a constant q spectral transform. *Journal of the Acoustical Society of America*, 89:425–, 01 1991. doi: 10.1121/1.400476.

John Burgoyne, Jonathan Wild, and Ichiro Fujinaga. An expert ground truth set for audio chord recognition and music analysis. pages 633–638, 01 2011.

Antoine Caillon and Philippe Esling. Rave: A variational autoencoder for fast and high-quality neural audio synthesis, 2021. URL `https://arxiv.org/abs/2111.05011`.

Chris Cannam, Craig Landone, and Mark Sandler. Omras2 metadata project. In *Proceedings of the 10th International Society for Music Information Retrieval Conference (ISMIR)*, pages 309–310, 2009.

Pablo Cano, Eloi Batle, Ton Kalker, and Jaap Haitsma. A review of audio fingerprinting. *Journal of VLSI signal processing systems for signal, image and video technology*, 41 (3):271–284, 2005. doi: 10.1007/s11265-005-4151-3.

Ryan Castellon, Chris Donahue, and Percy Liang. Codified audio language modeling learns useful representations for music information retrieval. In *Proceedings of the 22nd International Society for Music Information Retrieval Conference (ISMIR)*, pages 885–892, 2021.

Tsung-Ping Chen and Li Su. Harmony transformer: Incorporating chord segmentation into harmony recognition. In *International Society for Music Information Retrieval Conference*, 2019. URL `https://api.semanticscholar.org/CorpusID:208334896`.

Tsung-Ping Chen and Li Su. Attend to chords: Improving harmonic analysis of symbolic music using transformer-based models. *Trans. Int. Soc. Music. Inf. Retr.*, 4:1–13, 2021. URL `https://api.semanticscholar.org/CorpusID:232051159`.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1179. URL `https://aclanthology.org/D14-1179/`.

Taemin Cho and Juan Pablo Bello. On the relative importance of individual components of chord recognition systems. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(2):477–492, 2014.

Taemin Cho, Ron J. Weiss, and Juan P. Bello. Exploring common variations in state of the art chord recognition systems. In *Proceedings of the 7th Sound and Music Computing Conference (SMC)*, page 31, Barcelona, Spain, 2010. Sound and Music Computing Network.

John Thickstun Chris Donahue and Percy Liang. Melody transcription via generative pre-training, 2022. URL `https://arxiv.org/abs/2212.01884`.

Jade Copet, Felix Kreuk, Itai Gat, Tal Remez, David Kant, Gabriel Synnaeve, Yossi Adi, and Alexandre Defossez. Simple and controllable music generation. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 47704–47720. Curran Associates, Inc., 2023. URL `https://proceedings.neurips.cc/paper_files/paper/2023/file/94b472a1842cd7c56dcb125fb2765fbd-Paper-Conference.pdf`.

Michael Scott Cuthbert and Christopher Ariza. Music21: A toolkit for computer-aided musicology and symbolic music data. In J. Stephen Downie and Remco C. Veltkamp, editors, *ISMIR*, pages 637–642. International Society for Music Information Retrieval, 2010. ISBN 978-90-393-53813. URL `http://dblp.uni-trier.de/db/conf/ismir/ismir2010.html#CuthbertA10`.

Steven Davis and Paul Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(4):357–366, 1980. doi: 10.1109/TASSP.1980.1163420.

Jacopo de Berardinis, Albert Meroño-Peñuela, Andrea Poltronieri, and Valentina Presutti. Choco: a chord corpus and a data transformation workflow for musical harmony knowledge graphs. *Scientific Data*, 10, 09 2023. doi: 10.1038/s41597-023-02410-w.

T. De Clercq and D. Temperley. A corpus analysis of rock harmony. In *Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR)*, pages 23–28, 2011. URL `https://ismir2011.ismir.net/papers/OS6-1.pdf`.

Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. Jukebox: A generative model for music, 2020. URL `https://arxiv.org/abs/2005.00341`.

Gabriel Durán and Patricio de la Cuadra. Transcribing Lead Sheet-Like Chord Progressions of Jazz Recordings. *Computer Music Journal*, 44(4):26–42, 12 2020. ISSN 0148-9267. doi: 10.1162/comj_a_00579. URL `https://doi.org/10.1162/comj_a_00579`.

Takuya Fujishima. Realtime chord recognition of musical sound: a system using common lisp music. In *International Conference on Mathematics and Computing*, 1999. URL `https://api.semanticscholar.org/CorpusID:38716842`.

Masataka Goto, Hiroki Hashiguchi, Takuichi Nishimura, and Ryuichi Oka. Rwc music database: Popular, classical, and jazz music databases. 01 2002.

C. Harte and M. Sandler. Understanding effects of subjectivity in measuring chord estimation accuracy. In *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR)*, pages 345–350, 2010. URL `https://www.researchgate.net/publication/260711996_Understanding_Effects_of_Subjectivity_in_Measuring_Chord_Estimation_Accuracy`.

Christopher Harte, Mark Sandler, Samer Abdallah, and Emilia Gómez. Symbolic representation of musical chords: A proposed syntax for text annotations. pages 66–71, 01 2005.

Eric J. Humphrey and Juan P. Bello. Rethinking automatic chord recognition with convolutional neural networks. In *2012 11th International Conference on Machine Learning and Applications*, volume 2, pages 357–362, 2012a. doi: 10.1109/ICMLA.2012.220.

Eric J. Humphrey and Juan P. Bello. Rethinking automatic chord recognition with convolutional neural networks. In *2012 11th International Conference on Machine Learning and Applications*, volume 2, pages 357–362, 2012b. doi: 10.1109/ICMLA.2012.220.

Eric J. Humphrey and Juan Pablo Bello. Four timely insights on automatic chord estimation. In *International Society for Music Information Retrieval Conference*, 2015. URL `https://api.semanticscholar.org/CorpusID:18774190`.

Eric J. Humphrey, Justin Salamon, Oriol Nieto, Jon Forsyth, Rachel M. Bittner, and Juan P. Bello. Jams: A json annotated music specification for reproducible mir research. pages 591–596, 2014. 15th International Society for Music Information Retrieval Conference, ISMIR 2014 ; Conference date: 27-10-2014 Through 31-10-2014.

Yun-Ning Hung, Ju-Chiang Wang, Minz Won, and Duc Le. Scaling up music information retrieval training with semi-supervised learning, 2023. URL `https://arxiv.org/abs/2310.01353`.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, volume 37, pages 448–456. PMLR, 2015.

Junyan Jiang, K. Chen, Wei Li, and Gus G. Xia. Large-vocabulary chord transcription via chord structure decomposition. In *International Society for Music Information Retrieval Conference*, 2019. URL `https://api.semanticscholar.org/CorpusID:208334209`.

Jongmin Jung, Andreas Jansson, and Dasaem Jeong. Musicgen-chord: Advancing music generation through chord progressions and interactive web-ui, 2024. URL `https://arxiv.org/abs/2412.00325`.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1412.6980`.

Filip Korzeniowski and Gerhard Widmer. A fully convolutional deep auditory model for musical chord recognition. In *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2016a. doi: 10.1109/MLSP.2016.7738895.

Filip Korzeniowski and Gerhard Widmer. Feature learning for chord recognition: The deep chroma extractor. In *Proceedings of the 17th International Society for Music Information Retrieval Conference (ISMIR)*, pages 37–43, 2016b.

Nadine Kroher, Helena Cuesta, and Aggelos Pikrakis. Can musicgen create training data for mir tasks?, 2023. URL `https://arxiv.org/abs/2311.09094`.

Yun-Han Lan, Wen-Yi Hsiao, Hao-Chung Cheng, and Yi-Hsuan Yang. Musicongen: Rhythm and chord control for transformer-based text-to-music generation. In *Proceedings of the 25th International Society for Music Information Retrieval Conference (ISMIR)*, San Francisco, United States, 2024. URL `https://arxiv.org/abs/2407.15060`.

Sang-Hoon Lee, Hyun-Wook Yoon, Hyeong-Rae Noh, Ji-Hoon Kim, and Seong-Whan Lee. Multi-spectrogan: High-diversity and high-fidelity spectrogram generation with adversarial style combination for speech synthesis. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35:13198–13206, 05 2021. doi: 10.1609/aaai.v35i14.17559.

Mark Levine. *The Jazz Theory Book*. Sher Music Co., 1995. ISBN: 9781883217044.

Yizhi Li, Ruibin Yuan, Ge Zhang, Yinghao Ma, Xingran Chen, Hanzhi Yin, Chenghao Xiao, Chenghua Lin, Anton Ragni, Emmanouil Benetos, Norbert Gyenge, Roger Dannenberg, Ruibo Liu, Wenhu Chen, Gus Xia, Yemin Shi, Wenhao Huang, Zili Wang, Yike Guo, and Jie Fu. Mert: Acoustic music understanding model with large-scale self-supervised training, 2024. URL `https://arxiv.org/abs/2306.00107`.

Liwei Lin, Gus Xia, Junyan Jiang, and Yixiao Zhang. Content-based controls for music large language modeling, 2023.

Matthias Mauch and Simon Dixon. Approximate note transcription for the improved identification of difficult chords. pages 135–140, 01 2010.

Brian McFee and Juan Pablo Bello. Structured training for large-vocabulary chord recognition. In *International Society for Music Information Retrieval Conference*, 2017. URL `https://api.semanticscholar.org/CorpusID:3072806`.

Brian McFee, Colin Raffel, Dawen Liang, Daniel Ellis, Matt Mcvicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and music signal analysis in python. pages 18–24, 01 2015. doi: 10.25080/Majora-7b98e3ed-003.

Jeff Miller, Ken O'Hanlon, and Mark B. Sandler. Improving balance in automatic chord recognition with random forests. In *2022 30th European Signal Processing Conference (EUSIPCO)*, pages 244–248, 2022. doi: 10.23919/EUSIPCO55093.2022.9909558.

Y. Ni, M. McVicar, J. Devaney, I. Fujinaga, and M. Sandler. Annotator subjectivity in harmony annotations of popular music. *Journal of New Music Research*, 48 (2):118–135, 2019. doi: 10.1080/09298215.2019.1613436. URL https://www.tandfonline.com/doi/full/10.1080/09298215.2019.1613436.

Ken O'Hanlon and Mark B. Sandler. Comparing cqt and reassignment based chroma features for template-based automatic chord recognition. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 860–864, 2019. doi: 10.1109/ICASSP.2019.8682774.

Jonggwon Park, Kyoyun Choi, Sungwook Jeon, Dokyun Kim, and Jonghun Park. A bi-directional transformer for musical chord recognition. In *Proceedings of the 20th International Society for Music Information Retrieval Conference (ISMIR)*, pages 620–627, 2019. URL https://archives.ismir.net/ismir2019/paper/000075.pdf.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL https://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Johan Pauwels, Ken O'Hanlon, Emilia Gómez, and Mark B. Sandler. 20 years of automatic chord recognition from audio. In *International Society for Music Information Retrieval Conference*, 2019. URL https://api.semanticscholar.org/CorpusID:208334309.

Colin Raffel, Brian McFee, Eric J. Humphrey, Justin Salamon, Oriol Nieto, Dawen Liang, and Daniel P. W. Ellis. Mir_eval: A transparent implementation of common mir metrics. In *International Society for Music Information Retrieval Conference*, 2014. URL https://api.semanticscholar.org/CorpusID:17163281.

Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of Adam and beyond. In *International Conference on Learning Representations (ICLR)*, 2018. URL https://openreview.net/forum?id=ryQu7f-RZ.

Martin Rohrmeier and Ian Cross. Statistical properties of tonal harmony in bach's chorales. *Proceedings of the 10th International Conference on Music Perception and Cognition*, 01 2008.

Luke O. Rowe and George Tzanetakis. Curriculum learning for imbalanced classification in large vocabulary automatic chord recognition. In *Proceedings of the 22nd International Society for Music Information Retrieval Conference (ISMIR)*, pages 586–593, 2021. URL `https://archives.ismir.net/ismir2021/paper/000073.pdf`.

Gakusei Sato and Taketo Akama. Annotation-free automatic music transcription with scalable synthetic data and adversarial domain confusion, 2024. URL `https://arxiv.org/abs/2312.10402`.

Joseph Simplicio. *Guitar Chord Bible: 500 More Chords*. Hal Leonard Corporation, 2003. ISBN: 9780634057314.

S. S. Stevens, J. Volkmann, and E. B. Newman. A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America*, 8 (3):185–190, 1937. doi: 10.1121/1.1915893.

Keisuke Toyama, Taketo Akama, Yukara Ikemiya, Yuhta Takida, Wei-Hsiang Liao, and Yuki Mitsufuji. Automatic piano transcription with hierarchical frequency-time transformer, 2023. URL `https://arxiv.org/abs/2307.04305`.

G. Tzanetakis and P. Cook. Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, 10(5):293–302, 2002. doi: 10.1109/TSA.2002.800560.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL `https://arxiv.org/abs/1706.03762`.

Avery Wang. An industrial strength audio search algorithm. 01 2003.

Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70 of *Proceedings of Machine Learning Research*, pages 414–423. PMLR, 2017. URL `https://proceedings.mlr.press/v70/wilson17a.html`.

Yiming Wu, Tristan Carsault, and Kazuyoshi Yoshii. Automatic chord estimation based on a frame-wise convolutional recurrent neural network with non-aligned annotations. In *2019 27th European Signal Processing Conference (EUSIPCO)*, pages 1–5, 2019. doi: 10.23919/EUSIPCO.2019.8902741.

# Appendix A

# Appendix

## A.1 Small vs Large Vocabulary

## A.2 Chord Mapping

Chords in Harte notation were mapped to the vocabulary with $C = 170$ by first converting them to a tuple of integers using the Harte library. These integers represent pitch classes and are in the range 0 to 11 inclusive. They are transposed such that 0 is the root pitch. These pitch classes were then matched to the pitch classes of a quality in the vocabulary, similar to the work by McFee and Bello [2017]. However, for some chords, this was not sufficient. For example, a `C:maj6(9)` chord would not fit perfectly with any of these templates due to the added 9th. Therefore, the chord was also passed through Music21's [Cuthbert and Ariza, 2010] chord quality function which matches chords such as the one above to major. This function would not work alone as its list of qualities is not as rich as the one defined above. If the chord was still not matched, it was mapped to `X`. This additional step is not done by McFee and Bello [2017] but gives more meaningful labels to roughly one third of the chords previously mapped to `X`.

## A.3 CRNN with CR2

| cr2 | acc | root | third | seventh | mirex | $\text{acc}_{\text{class}}$ | $\text{median}_{\text{class}}$ |
|---|---|---|---|---|---|---|---|
| on | 59.7 | **78.9** | **75.6** | 61.9 | **80.5** | 18.4 | 0.4 |
| off | **60.2** | 78.4 | 75.3 | **62.5** | 79.5 | **19.4** | **1.1** |

Table A.1: *CRNN* with and without the added 'CR2' decoder. Performance is very similar between the two. It could be argued that te model with CR2 on is better, but for simplicity, I proceed with the model without CR2. One could also argue that the effect of CR2 is similar to simply adding more layers to the GRU already present in *CRNN*.

## A.4 A Long Run with SGD

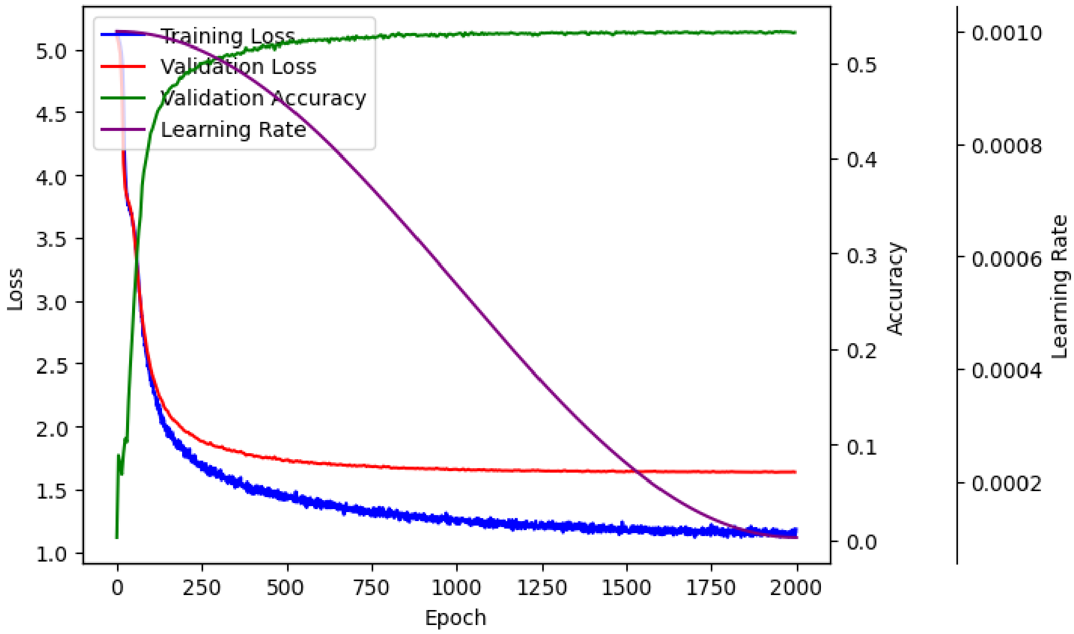


Figure A.1: Training graphs for *CRNN* trained with SGD, momentum 0.9, a learning rate of $0.001$ and the `cosine` scheduling for 2000 epochs. Convergence is reached but performance does not exceed that which is achieved by Adam over 150 epochs. Furthermore, there is signifcant computational cost associated with running for 2000 epochs. I proceed with Adam for the remainder of experiments.

## A.5 Random Hyperparameter Search Sets

`hidden_size` $\in \{32, 33, \ldots, 512\}$, `num_layers` $\in \{1, 2, 3\}$, `segment_length` $\in \{5, 6, \ldots, 45\}$, `kernel_size` $\in \{5, 6, \ldots, 15\}$, `cnn_layers` $\in \{1, 2, \ldots, 5\}$ and `cnn_channels` $\in \{1, 2, \ldots, 5\}$

## A.6 Confusion Matrix of CRNN over Roots

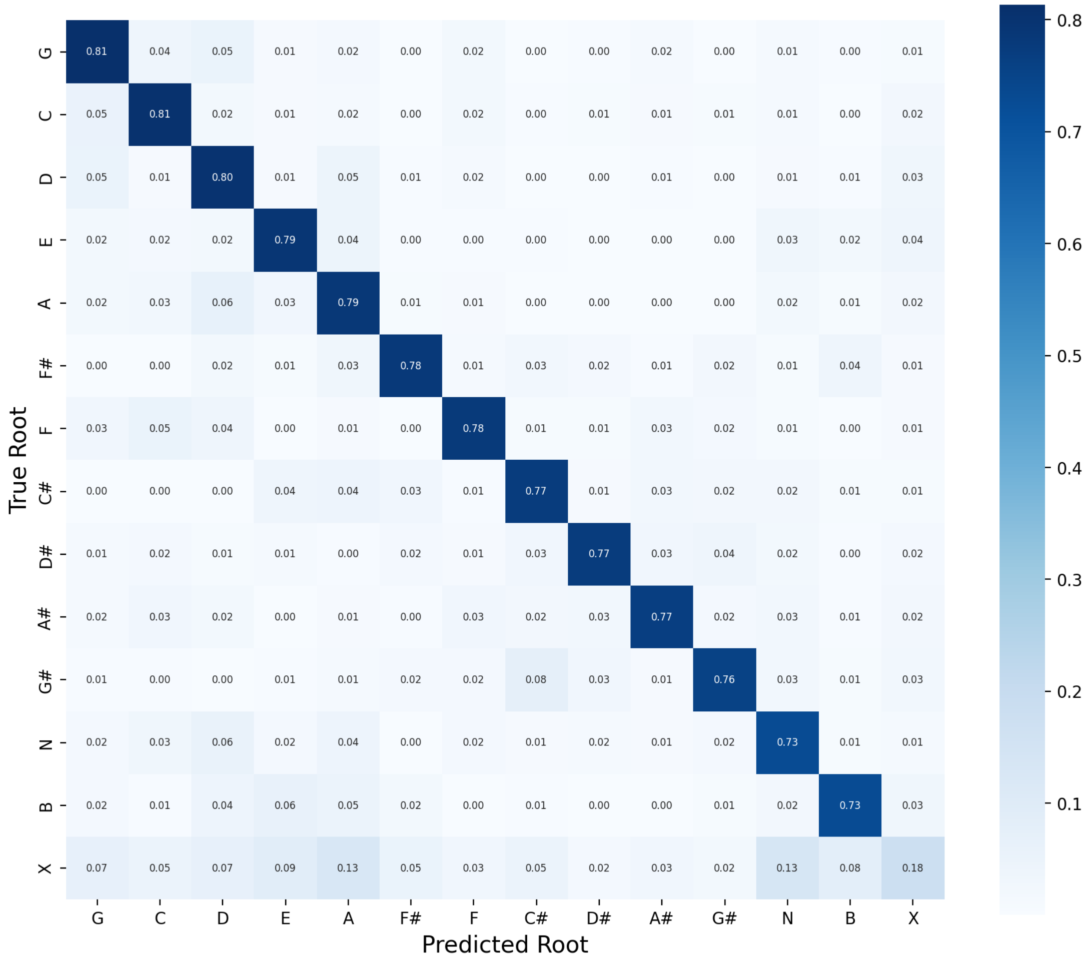


Figure A.2: Performance is relatively stable across roots. The only outlier is the unknown chord symbol X. This is to bexpected given the ambiguous nature of the chord.

## A.7 Incorrect Region Lengths With/Without Smoothing

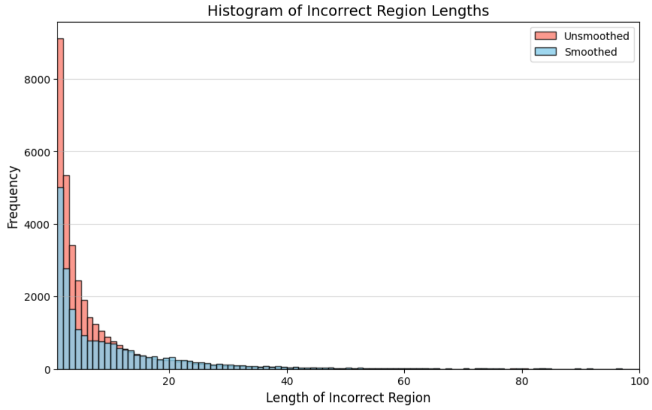


Figure A.3: Histogram over incorrect region lengths for a *CRNN* with and without smoothing. An incorrect region is defined as a sequence if incorrect frames with correct adjacent of either end. Both distributions have a long-tail, with $26.7\%$ regions being of length 1 without smoothing. This raises concerns over the smoothness of outputs and requires some form of post-processing explored in Section 5.2. The distribution is more uniform with smoothing, with approximately half the very short incorrect regions.

## A.8 Accuracy over the Context

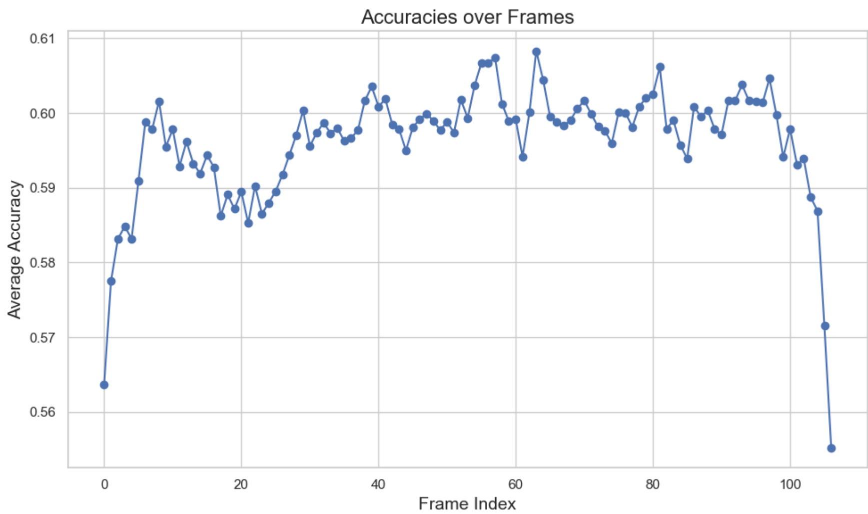


Figure A.4: Average frame-wise accuracy of the *CRNN* model over the patch of audio. The model performs worse at the beginning and end of the patch of audio, as expected. However, the differences are only $0.05$. We propose that the context on one side is enough for the model to attain the vast majority of the performance attained with bi-directional context. This plot supports our procedure of evaluating over the entire song at once.

## A.9 Accuracy vs Context Length of Evaluation

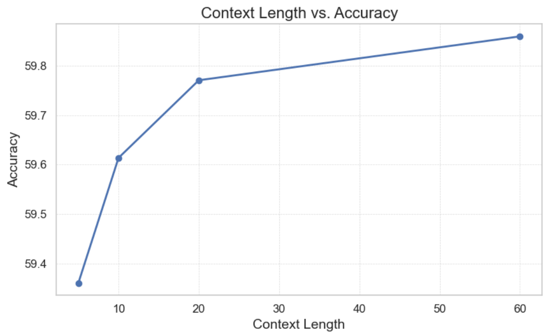


Figure A.5: Accuracy with increasing segment length of validation set. The accuracy increases very slightly. I choose to continue evaluating over the entire song at once.