Lead Sheet Transcription

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

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Aims . . . 1
- 1.3 Outline . . . 1

2 Background & Related Work **2**
- 2.1 Background . . . 2
  - 2.1.1 Harmony, Chords and Chord Recognition . . . 2
  - 2.1.2 Music Features . . . 4
- 2.2 Related Work . . . 5
  - 2.2.1 Automatic Chord Recognition . . . 5
  - 2.2.2 Synthetic Data Generation . . . 6

3 Experimental Setup **7**
- 3.1 Datasets . . . 7
  - 3.1.1 Preprocessing . . . 7
  - 3.1.2 Pop . . . 9
  - 3.1.3 JAAH Dataset . . . 12
- 3.2 Evaluation . . . 12
- 3.3 Training . . . 14

4 Model Comparison **15**
- 4.1 Logistic Baseline . . . 15
- 4.2 CRNN . . . 16
  - 4.2.1 Model Description . . . 16
  - 4.2.2 Small to Large Vocabulary . . . 16
  - 4.2.3 Hyperparameter Tuning . . . 18
  - 4.2.4 Model Analysis . . . 21
- 4.3 Weighted Loss . . . 27
- 4.4 Pitch Augmentation . . . 28
- 4.5 Structured Training . . . 29
- 4.6 Transformer . . . 29
- 4.7 Using Generative Features . . . 29
- 4.8 Decoding . . . 29
- 4.9 Results on Test Set . . . 29

5 Synthetic Data Generation **30**
5.1 Performance on JAAH 30
5.2 Generation method 30
5.3 Experiments 30
5.4 Results 30
5.5 Qualitative Analysis 30

6 Conclusions, Limitations and Further Work **31**
6.1 Conclusions 31
6.2 Limitations 31
6.3 Further Work 31

Bibliography **32**

A Appendix **36**
A.1 Histogram over Incorrect Region Lengths 36
A.2 Accuracy vs Context Length of Evaluation 37
A.3 Accuracy vs Hop Length 37

Chapter 1

Introduction

1.1 Motivation

TODO: Rewrite introduction to focus on chord recognition.

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

1.2 Aims

The aims of this project are:

- Compare state-of-the-art models for automatic chord recognition.
- Investigate methods of improving on this baseline model for music
- To investigate the use of synthetic data generation for improving the performance of the model.

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, we first provide a brief introduction to harmony and chords in music and the importance of recognising them. We then discuss the different ways in which music can be represented as input to a machine learning model. We then provide an overview of the field of automatic chord recognition (ACR). We discuss 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, Chords and Chord Recognition

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. Chords are often built from a root note and the third and fifth degrees of the scale. Chords can be further 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, based off of the major and minor scales but many others 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 many 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 are important for playing music. They provide a harmonic context for a melody, and can be used to convey emotion, tension and release. They are also important for improvisation where musicians will often play notes that fit the chord progression such that they create pleasing sound. Lead sheets are a form of musical notation which strip down a piece of music to a single melody line, a chord sequence and lyrics if present. 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 can be the basis of a song. They also play an important role in music analysis, where 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.

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 music analysis, for music recommendation and for music 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. Music researchers may wish to analyse the harmonic structure of a piece of music, or analyse the changes in common chord sequences over time. Music recommendation systems may wish to recommend songs based on their harmonic content as similar music will often have similar harmonic content. For example, pop music famously uses many similar chords [2] while jazz music is known for its rich, complex and nuanced harmonic exploration. Music generation systems can generate based on the harmonic structure of existing songs.

All of the above motivate the need for accurate chord recognition systems. Annotations from online sources can be of varying quality and may not be available for all songs. Furthermore, the task of annotating chords is time-consuming and requires a trained musician. Automatic chord recognition systems can help to alleviate these problems by providing a fast, accurate and scalable solution.

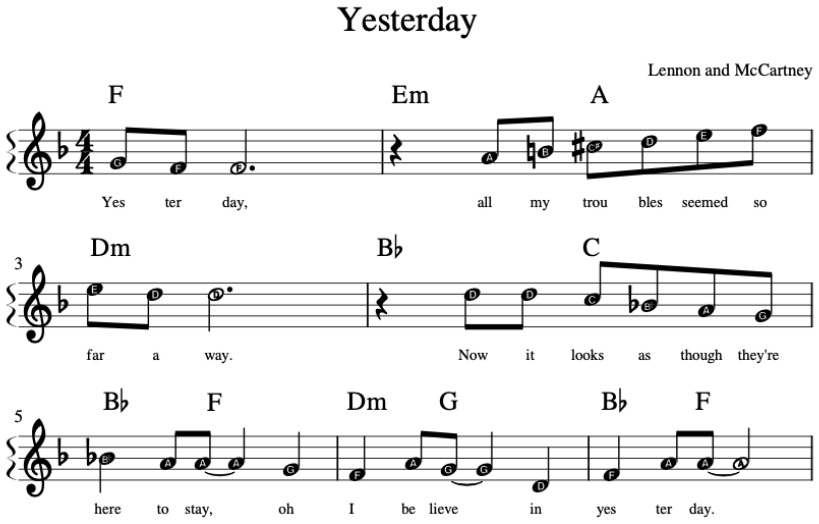

Figure 2.1: An example of a lead sheet for 'Yesterday' by the Beatles. We can see chords written above the stave, and the melody written in standard musical notation. Such a chordal representation is useful for musicians who want to learn and perform songs quickly and easily.

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

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

2.1.2 Music Features

Recorded music can be represented in a variety of ways as input to a machine learning model. The simplest way is to leave the data as a waveform: a time-series vector of amplitudes. Data in the raw audio domain has seen successful use in generative models such as Jukebox (Dhariwal et al., 2020) and RAVE (Caillon and Esling, 2021).

Spectrogram: A very commonly used representation of general audio data, and specifically musical data, is the spectrogram. A spectrogram is a conversion of the time-series data into the time-frequency domain, calculated by 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). However, as noted by Pauwels et al. (2019), only logarithmic spectrograms have been used in ACR tasks while linear spectrograms have been used in melody transcription tasks.

CQT: A common version of the spectrogram used in music transcription is the Constant-Q Transform (CQT), originally proposed by Brown (1991). The CQT is a version of a 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: a sine wave that is perceived as one octave higher than another has double the frequency. The CQT is used in many music transcription tasks, such as automatic chord recognition (Humphrey and Bello, 2012), and a popular alternative, Melspectrograms, have found use in melody transcription (Toyama et al., 2023). An example CQT from the dataset used in this work is shown in Figure 2.2.

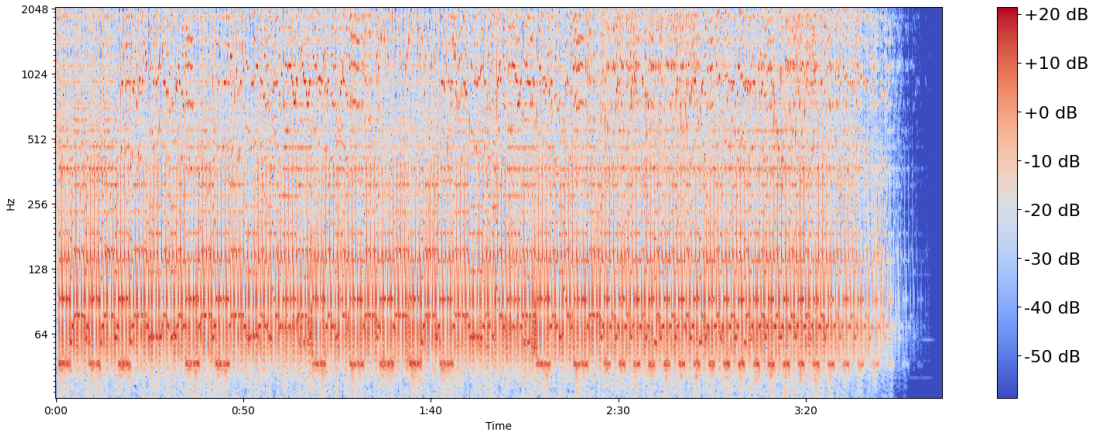

Figure 2.2: A sample CQT of 'Girls Just Wanna Have Fun' by Cyndi Lauper from the dataset used in this work. We can clearly see the log-spaced frequency bins. We also observe clear structure and repetition in the song, especially in the lower frequencies, which can be attributed to a regular drum groove and bass instruments. This is typical of pop songs in this dataset.

Chroma Vectors: Chroma vectors are a 12-dimensional time-series representation, where each dimension corresponds to a pitch class. Each element represents the presence

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) and have seen use in recent ACR models (Chen and Su, 2019).

Generative Features: More recently, features extracted generative models have been used as input. The proposed benefit is that the vast quantities of data used to train these models allows for rich representations of the music. 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 and combine these into a lead sheet generation model. They found that these features outperformed hand-crafted features in melody transcription tasks but did not perform experiments in ACR.

2.2 Related Work

2.2.1 Automatic Chord Recognition

Chord recognition is a difficult task. Which chord is playing when is inherently ambiguous. Different chords can share the same notes, and the same chord can be played in many different ways. Furthermore, the same chord can be played in different contexts, such as a different key or time signature, and can sound different. Whether a melody note is part of a chord or not is ambiguous and whether a melody alone is enough to imply harmonic content is also ambiguous. In order to identify a chord, information across time must be considered as the chordal information may be spread over time. For example, a chord may vamped or arpeggiated. Audio contains many unhelpful elements for chord recognition such as reverb, distortion and percussion. Combined with the lack of labelled data, this makes chord recognition a challenging task.

Pauwels et al. (2019) provide an overview of ACR since the seminal work of Fujishima (1999) in 1999 up to 2019, and provide suggestions for future avenues of research.

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

- *Mcgill Billboard*: 890 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 a larger dataset with artists chosen for popularity. (Berenzweig et al., 2004)
- *JAAH*: 113 annotations of a collection of jazz recordings. (Durán and de la Cuadra, 2020)

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

- *HookTheory*: 50 hours of labelled audio in the form of short musical segments, crowdsourced from the HookTheory website[4]. (Chris Donahue and Liang, 2022)

Other small datasets also exist. Many of these have been compiled together into the *Chord Corpus* by de Berardinis et al. (2023), with standardised annotation formats.

A big problem frequently encountered in ACR is the lack of labelled data. This work uses 1200 labelled songs with audio. This is due to the difficulty and time associated with labelling data aligned in time and the legal sensitivity of the data involved.Data has been scaled up using augmentation and semi-supervised learning (Hung et al., 2023) with some success. Research has been done into the use of synthetic data (Kroher et al., 2023; Sato and Akama, 2024) and supervised learning (Li et al., 2024) for MIR tasks, but not for ACR.

Another problem is that the existing data is often imbalanced, with a large number of common chords like major and minor chords and fewer chords like diminished and augmented chords, or chords with upper extensions and inversions. 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 re-weighting classes (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).

Models: A variety of machine learning methods have been applied to such datasets. Older methods such as the work by (Lee and Slaney, 2006) use hidden Markov models (HMMs) but more recent methods use deep learning. Convolutional neural networks (CNNs) and recurrent neural networks (RNNs) have been used together in many works (Wu et al., 2019; Jiang et al., 2019; McFee and Bello, 2017) with CNNs performing feature extraction from a spectrogram feature and an RNN sharing information across frames. More recently, transformers have been applied to the entire process Chris Donahue and Liang (2022) and Chen and Su (2019, 2021).

Decoding: HMMs still see use as a decoder operating on the frame-wise probability distributions over chords generated by the model (Miller et al., 2022). Other works have used conditional random fields (CRF) in their place to model the dependencies between chords (Jiang et al., 2019).

Evaluation: Evaluation is typically done using accuracy and recall of correct chord predictions, or more music-aware measures such as the correct root note, third or mirex, metric which measures that chords have at least 3 notes in common. These are all implemented by Raffel et al. (2014) in the `mir_eval` library [5]. Qualitative evaluation is also often carried out.

2.2.2 Synthetic Data Generation

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

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

Chapter 3

Experimental Setup

This chapter outlines 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 used throughout.

3.1 Datasets

Two ACT datasets are used in this work. The first dataset is simply referred to as the *Pop* dataset, as much of the music in the dataset comes from the Billboard Hot 100 charts, or other popular bands. The second dataset is the *JAAH* (Jazz Annotations and Analysis of Harmony) dataset mentioned in Section 2.2.1. While the chords annotations are publicly available on Github[1], the audio was kindly given to me by Andrea Poltronieri, a PhD student at the University of Bologna and the author of 'ChoCo' (the Chord Corpus) (de Berardinis et al., 2023). I spent the first weeks of the project contacting authors of several papers which have used these or other datasets and he was the only one helpful enough to provide me with the audio.

The rest of this chapter explains 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 2.1.2. The CQT is computed using the `librosa` library (McFee et al., 2015), using the `.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`. This returns a complex-valued matrix containing phase, frequency and amplitude information. Phase information is discarded by taking the absolute value, before being converted from amplitude to dB, equivalent to taking the logarithm. These

[1] https://github.com/smashub/choco

default parameters were chosen to be consistent with previous works (McFee and Bello, 2017). However, the hop size was later varied in order to investigate the effect of different frame lengths on performance. An alternative choice of frame length could be to use some fraction of a bar. For example, Chris Donahue and Liang (2022) use 1/16th of a bar. However, most work in the literature opts for a constant hop size. We follow this convention, as a hop size which varies over songs introduces a new source of error in identifying the beats of a song, which the authors note as sometimes being problematic to their method.

This leads to a CQT matrix of a song having dimensions $216 \times F$ where 216 is the number of frequency bins and F is the number of frames in the song. For context, 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 hop size in samples. A 3 minute song has just under 2000 frames. To save on computational time, the CQT was pre-computed into a cached dataset rather than re-computing each CQT on every run. This was done for each hop size used.

3.1.1.2 Chord Annotations

The chord annotations are represented as a sorted list of dictionaries, each containing the chord, start time and duration. The chord itself 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`. However, the majority chords in the dataset are not this complicated. The notation also includes `N` which signifies that no chord is playing and `X` which represents any out of gamut chord.

This annotation is far too flexible to be used as targets for a machine learning model trained on limited data. This would lead to thousands of classes, many of which would appear only once. Instead, we define two chord vocabularies. The first is a simple chord vocabulary which contains only major and minor for each root and a no chord symbol `N`. Chords outside the vocabulary are mapped to `X`. For example, `C:maj7` is mapped to `C:maj` while `A:hdim7/5` is mapped to `X`. Let V be the size of the vocabulary, then $V = 26$ for the small vocabulary. The second is a more complex vocabulary, which contains 14 qualities for each root: major, minor, diminished, augmented, minor 6, major 6, minor 7, minor-major 7, major 7, dominant 7, diminished 7, half diminished 7, suspended 2, suspended 4. Additionally, we retain the `N` and `X` symbols totalling $V = 12 \cdot 14 + 2 = 170$ chord labels. This vocabulary or similar vocabularies have been used by much of the literature (McFee and Bello, 2017; Humphrey and Bello, 2015; Jiang et al., 2019).

Chords in Harte notation are mapped to this vocabulary by first converting them to a tuple of integers using the Harte library. These integers represent pitch classes and are in the range 0-11 inclusive. These are transposed such that 0 is the root. These pitch classes are 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 is 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 is still not matched, it is 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`.

3.1.2 Pop

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 most of the known datasets for chord recognition. The dataset consists of 1,217 songs, filtered for duplicates and selected for those with annotations available. 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 were investigated.

Duplicates: Files were renamed using provided metadata identifying them by artist and song title. This was done to identify duplicates in the dataset. The dataset was first filtered for duplicates, of which there was one - Blondie's 'One Way or Another', which had two different versions. The duplicate was removed from the dataset. Further duplicates may exist under different names but the songs were listened to throughout the project and no other duplicates were found. Automatic analysis of the audio could help, although cannot be guaranteed to find different versions of the same song, as above.

Chord-Audio Alignment: 10 songs were manually investigated for alignment issues. This was done by listening to the audio and comparing it to the annotations directly at various points in the song, with a focus on the beginning. It became apparent that precise timings of chord changes are ambiguous. The annotations aired on the side of being slightly early but were all certainly good annotations, with detailed chord labellings including inversions and upper extensions. This observation was borne in mind later when analysing performance of the models at transition frames in Section [XX].

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, varying a 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 changes. First, the CQT of the audio was computed following the procedure defined in Section 3.1.1.1. The derivative of the CQT in the time dimension was then 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 this as one bar-length, where drum transients will be highly correlated with the chord changes.

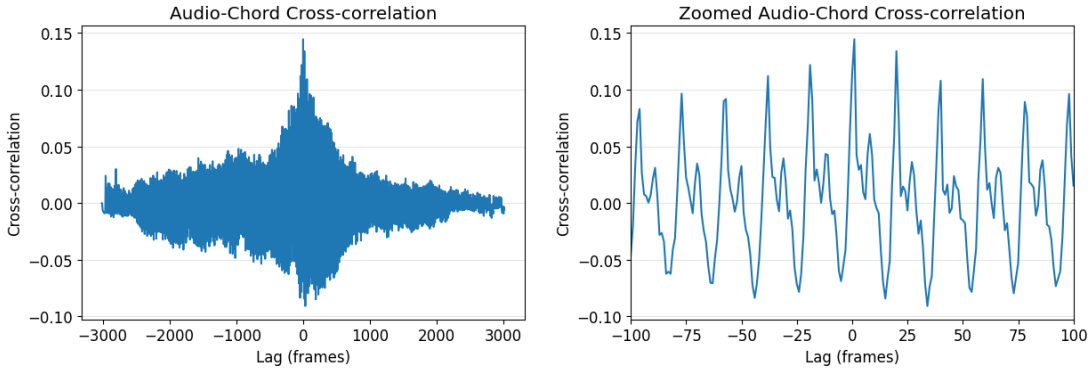

Figure 3.1: Cross-correlation of the derivative of the CQT of the audio and the chord annotations for a single song. We can see correlation peaking in regular intervals of around 20 frames, which corresponds to one bar length in this song. On a larger scale, it is highest on around 0.

To check alignment across the dataset, we can plot the lag of the maximum cross-correlations as a histogram. If we further assume that the annotations are not incorrect by more than approximately 10 seconds, this would mean we could 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 100. This reduction does not change the shape of the picture. Rather, focusing on a reduced set of frames allows more detail to be visible. A final simple check was done by looking at the lengths of the audio and chord annotations. A histogram of differences in length is also shown in Figure 3.2. The majority of songs have a maximum lag in the region of 0, with a few outliers. This can be attributed to noise. Furthermore, 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.

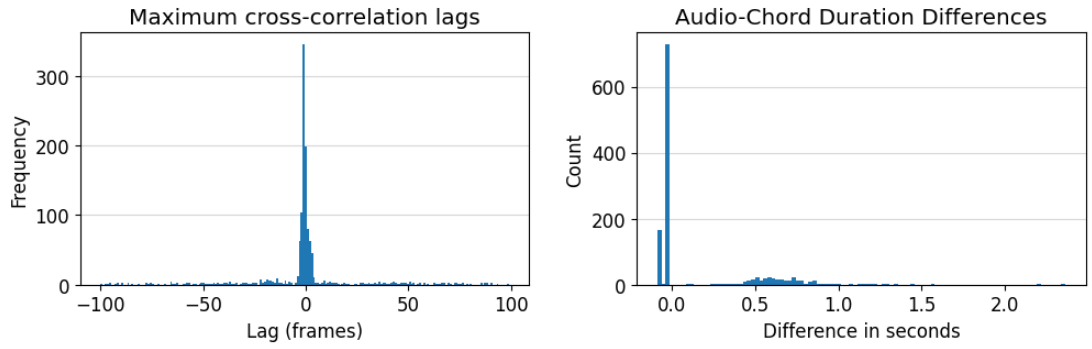

Figure 3.2: Cross-correlation of the derivative of the CQT of the audio and the chord annotations for a single song. The x-axis is the lag in frames and the y-axis is the correlation. The plot repeats every 100 frames, which corresponds to 4 bars.

Incorrect and Subjective Annotations: Throughout manual listening, no obviously wrong annotations were found. However, looking at songs which the first trained models perform the worst on using the `mirex` metric, a 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 A440 for the first two songs and the key of the annotation was wrong for the other. These songs were kept in to maintain consistency with previous works and are unlikely to skew reported metrics much as explained in Section 3.2. 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).

3.1.2.2 Chord Distribution

Much of the recent literature has focused on the long tail of the chord distribution (Miller et al., 2022; Rowe and Tzanetakis, 2021), 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 by quality, only for the larger chord vocabulary as the smaller vocabulary does not have the same long-tailed distribution. The plots clearly 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 having two to three orders of magnitude fewer frames. Another helpful display of chord qualities can be found in the work by Jiang et al. (2019). The distribution over roots is far less skewed, although there is unsurprisingly a preference for chords from simpler keys like C, D or E and fewer in C# or F#.

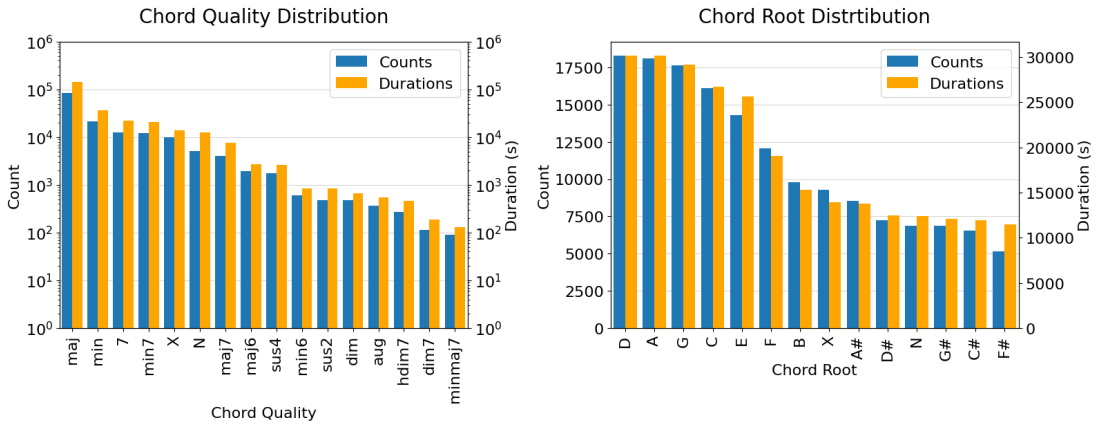

Figure 3.3: Chord distributions in the *Pop* dataset. The plots show both the raw counts in terms of frames and the duration in seconds for each chord root/quality. Note that the scales on the qualities are in log-scale. We observe that the qualities are very imbalanced, with `maj` the most popular,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

3.1.3 JAAH Dataset

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

We evaluated models using the standard `mir_eval` library (Raffel et al., 2014). This library provides a variety of metrics for evaluating the performance of chord recognition models. The metrics used in this work follow standard practice, in using `root`, `third`, `seventh` and `mirex` accuracies. For a given hypothesis and reference chord, these metrics each return 1 if the root/third/seventh/three notes of the chord match and 0 otherwise. They return -1 if the true label is `X`. `X` is a highly ambiguous class with a wide variety roots and qualities that are mapped to it. From a musician's perspective, it would be better to have a reasonable guess at the chord rather than `X` which provides little information. Therefore, the model should not be penalise on frames with true label `X` and as such, these are ignored for all evaluation.

Finally, we looked at the overall frame-wise accuracy, referred to as `frame` or simply accuracy. We choose not to use `majmin`, `triad` or `tetrad` as these are highly correlated with the previous metrics. This makes sense from a theoretical standpoint as the third and seventh are strong indicators of the triad and tetrad of the chord. This was also verified empirically on some initial results, where the `third` and `majmin` accuracies were always within 1% of one another.

For the above metrics, the mean is computed for each song, and both the mean and median of the mean across songs are returned. The mean and median were found to within $\pm 0.01\%$ on all values observed, although the median was consistently slightly higher, likely due to there being more negative outliers than positive outliers. Some of these outliers were those previously identified as having incorrect annotations or could be those identified as being unsuitable for chordal analysis (Humphrey and Bello, 2015). Because of the similarity between the two, we report only the mean throughout this work, chosen as it is more commonly used in recent literature. These metrics can be formally defined as follows:

$$M(f) = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{|\{t \in T : y_{i,t} \neq \mathtt{X}\}|} \sum_{t \in T : y_{i,t} \neq \mathtt{X}} f(y_{i,t}, \hat{y}_{i,t})$$

where M is the metric in question, f is a function which calculates the metric for a pair of reference and hypothesis chord labels, defined for each metric as above. N is the number of songs, T is the set of frames in a song, $y_{i,t}$ is the true chord at frame t of song i, and $\hat{y}_{i,t}$ is the predicted chord at frame t of song i.

For some experiments, we looked at three more metrics. The first is the overall frame-wise accuracy, calculated across the entire evaluation dataset rather than on a per-song basis, still ignoring `X` chords, called $\texttt{frame}_{\text{all}}$. The second and third are the mean and median class-wise accuracies, called $\texttt{class}_{\text{mean}}$ and $\texttt{class}_{\text{median}}$ respectively. These are the mean/median accuracy over each chord class. These metrics are formally defined as follows:

$$\text{frame}_{\text{all}} = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{|\{t \in T : y_{i,t} \neq \mathtt{X}\}|} \sum_{t \in T : y_{i,t} \neq \mathtt{X}} \mathbb{1}_{(y_{i,t} = \hat{y}_{i,t})}$$

$$\text{class}_{\text{mean}} = \frac{1}{C}\sum_{c=1}^{C} \text{acc}(c), \quad \text{class}_{\text{median}} = \text{median}_{c=1}^{C}[\text{acc}(c)]$$

where $\text{acc}(c)$ is the accuracy of class c defined as:

$$\text{acc}(c) = \frac{\sum_{i=1}^{N}\sum_{t \in T : y_{i,t} \neq \mathtt{X}} \mathbb{1}_{(y_{i,t} = c \wedge \hat{y}_{i,t} = c)}}{\sum_{i=1}^{N}\sum_{t \in T : y_{i,t} \neq \mathtt{X}} \mathbb{1}_{(y_{i,t} = c)}}$$

where C is the number of chord classes, and N, T, $y_{i,t}$ and $\hat{y}_{i,t}$ are as defined before.

This metric is intended to measure the model's performance on the long tail of the chord distribution. It is important to measure both the mean and median as the distribution of accuracies is skewed. These last two metrics are the only ones take into account the entire evaluation dataset at once. Some songs may contain very few chords of a particular chord or quality, making the per-song metrics very noisy.

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 performances of selected models.

Finally, other evaluation tools were used, such as confusion matrices and accuracies over particular chord qualities and classes. A small amount of qualitative analysis was also done, looking at the predictions of the model on a few songs. This was done to understand the model's behaviour and to identify any systematic errors.

3.3 Training

Three variants of the dataset were used for training, validation and testing. For training, an epoch consisted of randomly sampling a patch of audio from each song in the training set. The length of this sample was kept as a hyperparameter, set to 28 seconds for the majority of experiments. For testing, the entire song was used was performance was found to be better if the model was allowed to see the entire song at once. This is later discussed in Section 4.2.4.4. Because songs are of variable length, songs were split into patches of the same length as the training patches when validating mid-way through training. This meant that computation could be parallelised without many padded frames. For all variants, frames in the batch were padded to the maximum length of the batch and ignored for loss or metric calculation.

The models were trained using the `PyTorch` library (Paszke et al., 2019). All final experiments were run on the ML Teaching Cluster with a variety of Nvidia GPUs, mostly GTX 1060's or GTX 1080's. Unless state otherwise, models were 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 were trained to minimise the cross entropy loss between the predicted chord and the true chord distributions. The models were trained with a batch size of 64 for a maximum of 150 epochs unless stated otherwise. This batch size was based on brief experiments testing the speed of an epoch. Validation was conducted every 5 epochs in order to save on computation time. Optionally, training was stopped early if the validation loss did not improve for 25 epochs. The model was saved whenever the validation loss improved. Each training run took approximately 30 minutes of GPU time.

For the majority of experiments, a random 60/20/20% training/validation/test split was used. This contrasts much of the literature which uses a 5-fold cross validation split, as introduced by Humphrey and Bello (2015). We do not maintain this split in order to obtain clean estimators of the generalisation error using the held-out test set. The split was kept constant across experiments. Later, models were re-trained on the combined training and validation sets and tested on the test set. In Chapter 5, the models were trained on the entire *Pop* dataset and tested on the *JAAH* dataset.

Chapter 4

Model Comparison

In this section, we compare various model on the *Pop* dataset. We start by describing a baseline model. We then implement a model from the literature and investigate its behaviour. We look at weighting the loss function, training on a more structured loss function, and performance of a transformer model. Finally, we investigate the use of generative features as input, experiment with decoding methods and compare the best models on the held-out test set.

4.1 Logistic Baseline

As a simple baseline, we consider a single layer NN which treats each frame independently. Then the layer takes input of size 216 and outputs a V-dimensional vector, where V is the cardinality of the chord vocabulary. The outputs are then passed through a softmax layer to get the probability of each chord and the cross-entropy loss is calculated. We call this model *Logistic*.This can effectively be seen as a logistic regression model trained using stochastic gradient descent (SGD). We could have used a logistic regression model implemented in `sklearn` for example which may have found better minima but implementing it as a neural network was fast and easy and unlikely to yield significantly different results.

A grid search on learning rates and learning rate schedulers was conducted on the sets `[0.1, 0.01, 0.001, 0.0001]` and `[Cosine, Plateau, None]` respectively. The `Plateau` scheduler halves the learning rate when the validation loss hasn't improved for 10 epochs and `Cosine` is as described in Section 3.3. The best model was found to be a learning rate of 0.01 with a `Cosine` scheduler. This best model was chosen for having the highest score on the most of the metrics in validation set. All models with learning rates of 0.01 or 0.001 converged within 150 epochs. Although the best model had learning rate 0.01, a learning rate of 0.001 over 150 epochs had a more stable validation accuracy. The model's results can be seen in Table 4.1. Full results are omitted as they are not relevant to the main discussion. The model serves simply as a baseline to compare the more complex models to. These results give us the first empirical evidence that the task is non-trivial. The model is only able to predict the root of the chord with a mean frame-wise accuracy of 0.64 and a mirex of 0.65. The model

identifies both the root and the third with an accuracy of 0.56 but struggles more with the seventh with an accuracy of 0.44. The lowest scores are on class-wise accuracies able to predict the class of the chord with $\texttt{class}_{\text{mean}} = 0.13$ and $\texttt{class}_{\text{median}} = 0.03$. This gives us the first insight into each of the evaluation metrics and what we can hope from more complex models and other improvements.

Model	frame	root	third	seventh	mirex	$\text{class}_{\text{mean}}$	$\text{class}_{\text{median}}$
Logistic	0.42	0.64	0.56	0.44	0.65	0.13	0.03

Table 4.1: Baseline model results

4.2 CRNN

4.2.1 Model Description

We implement a convolutional recurrent neural network (CRNN) as described in McFee and Bello (2017). The model takes as input a matrix of size $I \times F$ where I is the number of input features and F is the number of frames. The model passes the input through a layer of batch normalisation, before being fed through two convolutional layers with ReLU after each one. The first convolutional layer has a 5×5 kernel, and outputs only one channel the same size as the input. It is intended to smooth out noise and spread some information across adjacent frames about sustained notes. The second layer has a kernel of size $1 \times I$, and outputs 36 values. This essentially acts as a linear layer across frames with shared inputs. 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 the softmax activation. This produces a vector of length V for each frame, where V is the size of the chord vocabulary.

The authors of the model also propose using second GRU as a decoder before the final dense layer, called 'CR2'. However, we believe that a similar effect could be achieved with more layers in the initial GRU. Furthermore, both in the paper and in brief empirical tests of our own, the results with 'CR2' were indistinguishable from the model without it. We therefore do not include it in our final model. Results are omitted as they are neither relevant nor interesting.

4.2.2 Small to Large Vocabulary

Initial experiments were conducted on the simpler chord vocabulary with $V = 25$. Only if the model could somewhat accurately classify the smaller vocabulary and if performance did not decrease on the smaller chord vocabulary when trained on the larger vocabulary, would we proceed to using the larger vocabulary. In keeping with with the methodology in McFee and Bello (2017), we initially run with a learning rate of 0.001. We reduce the learning rate to half its previous value if the validation loss hasn't improved in 10 epochs and stop training if it has not improved after 25 epochs, with a maximum of 100 epochs. Training samples were set to 10 seconds long. Model convergence was manually checked using the validation and training losses over epochs.

Results are shown in Table 4.2. For comparison, the table also shows the performance of the same model trained with the large vocabulary, $V = 170$ and its predictions mapped back to the smaller vocabulary. A confusion matrix over chord roots of the model trained on $V = 26$ is shown in Figure 4.1. The model performs better than the baseline on the larger vocabulary, which is to be expected given the nested nature of the models, and the harder task on the larger vocabulary. From the confusion matrix, it becomes clear that many of the mistakes the model is making lie in the `X` symbol, which constitutes just over 7% of the smaller vocabulary dataset. Chords with qualities like `sus4` could be confused with `major` by a reasonable model but are represented with `X` in the smaller vocabulary. Interestingly, the model trained with $V = 170$ performs nearly as well on all metrics as the model trained with $V = 25$. This implies that training with $V = 170$ allows the model to learn almost all the relevant information about the smaller vocabulary, and gives it the chance to learn something about the larger vocabulary as well. Therefore, we proceed with the larger vocabulary for the rest of the experiments.

While some other works continue to measure performance on the smaller vocabulary (Park et al., 2019), we believe more metrics distract from the main goal of increasing performance across a wider range of chords. Additionally, the `third` metric capture much of the information we would look for in the evaluating with $V = 25$. We therefore only measure performance on the larger vocabulary from now on.

Model	V for training	root	third	$\text{class}_{\text{mean}}$	$\text{class}_{\text{median}}$
CRNN	26	0.79	0.77	0.74	0.74
CRNN	170	0.78	0.74	0.72	0.73

Table 4.2: CRNN model results on the small vocabulary with $V = 26$. The other metrics are omitted as they are identical to `third` for classification with $V = 26$.

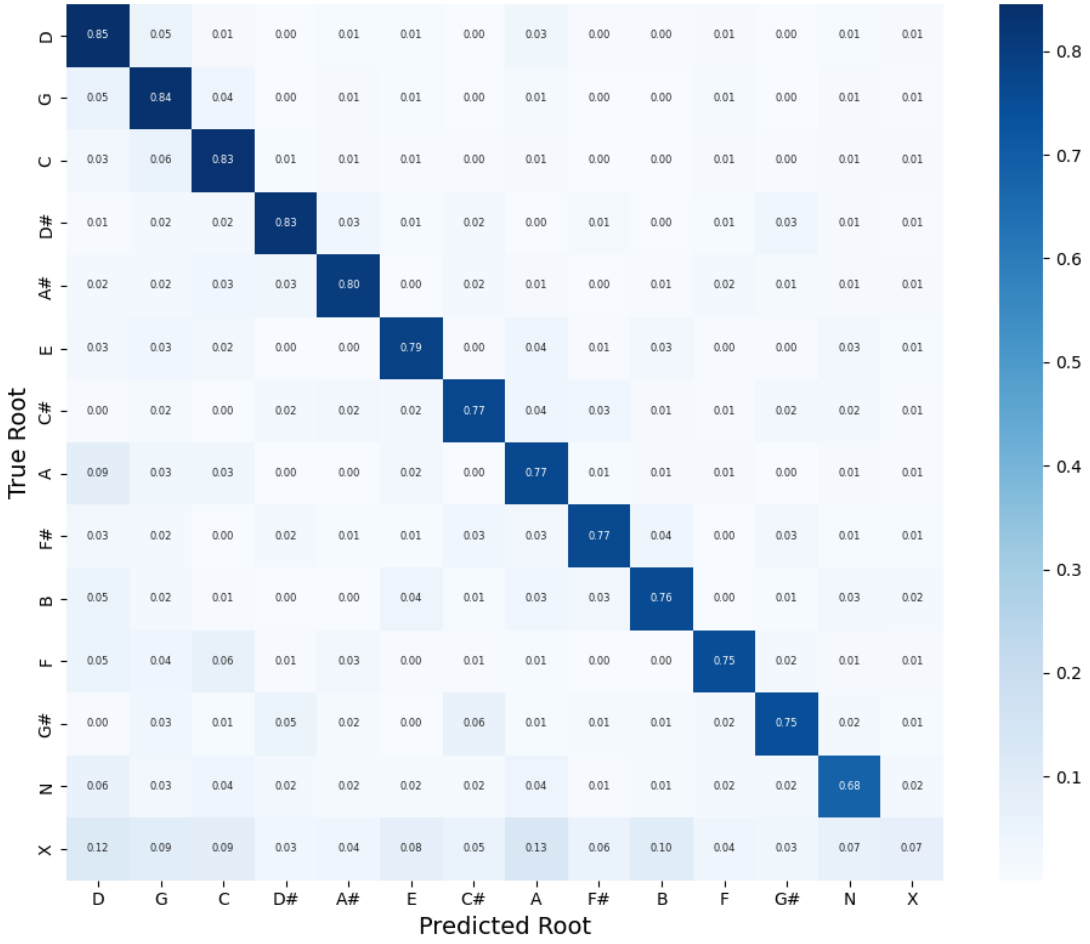

Figure 4.1: Confusion matrix over roots of the CRNN model trained on the small vocabulary. The values have been normalised over rows such that the values on the diagonals are recall metrics. There is a clear outlier in the model's recall with the label `X`, at just 0.07. It also performs poorly on `N`.

4.2.3 Hyperparameter Tuning

After progressing to the larger vocabulary, we thought it a good time to conduct some hyperparameter tuning.

4.2.3.1 Learning rates

We perform grid search on the same learning rates and learning rate schedulers as for the *Logistic* model. The learning rates were in the set of `[0.1, 0.01, 0.001, 0.0001]` and the learning rate schedulers to `[Cosine, Plateau, None]`. We remove early stopping in order to check for convergence and overfitting without the possibility of a pre-emptive stop. Judging by training graphs seen in 4.2, the best learning rate is 0.001. Any lower and we do not converge fast enough; any higher and gradient updates cause the validation accuracy to be noisy. These figures also show that the validation loss does not get worse after convergence. We conclude that the model is not quick to overfit, perhaps due to the random sampling in the training process. Combined with the

fact that training is relatively quick and we only save on improved validation loss, we decided to remove early stopping. We therefore conduct future experiments without early stopping and with `lr=0.001` and `Cosine` scheduling.

SGD has been shown to be a good optimiser for a longer run over many epochs if the model is able to converge [XX]. We ran an experiment over 2000 epochs with the above hyperparameters and `momentum=0.9` and found that the model was able to converge, reaching its best validation loss at around 1000 epochs and remaining flat thereafter. The results with this model did not improve on the best model trained with Adam, with 3% lower root accuracy and 2% lower third accuracy, but with 2% better mirex. Due to the much longer training time and the lower root accuracy, we decided to stick with Adam for the rest of the experiments.

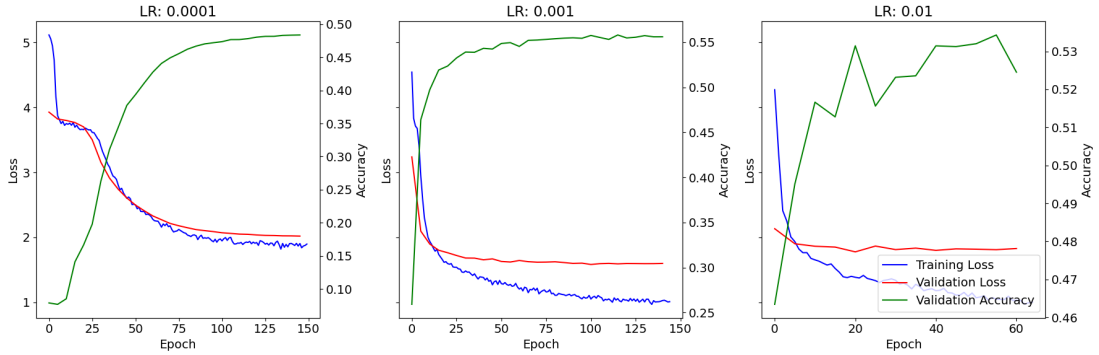

Figure 4.2: Training graphs for the CRNN model with different learning rates. The learning rate of 0.001 seems to be the best, as it converges in a reasonable time and the validation accuracy increases in a stable fashion.

We report a subset of metrics in Table 4.3. The best performing model by validation metrics was found to be with `lr=0.001` and `Cosine` scheduling. However, there were no large differences in performance between the learning rate schedulers. We proceed with these hyperparameters as defaults for the rest of the experiments.

lr	scheduler	root	third	seventh	mirex	$\text{class}_{\text{mean}}$	$\text{class}_{\text{median}}$
0.01	Cosine	0.71	0.68	0.55	0.76	0.13	0.00
0.001	Cosine	**0.77**	**0.73**	**0.60**	0.80	0.17	**0.01**
0.0001	Cosine	0.70	0.65	0.54	0.70	0.10	0.00
0.001	Plateau	0.74	**0.73**	**0.60**	0.80	**0.18**	**0.01**
0.001	None	0.71	0.70	0.58	**0.82**	0.17	0.00

Table 4.3: CRNN model results on the large vocabulary with different learning rates and schedulers. Overall, a learning rate of 0.001 and a scheduler of `Cosine` performs the best in many metrics, though a scheduler of `Plateau` performs just as well or better on many metrics. We prioritise the performance of the model on the root as this is more important than the `mirex` metric.

4.2.3.2 Model Hyperparameters

With this learning rate and learning rate scheduler fixed, we perform a random search on the number of layers in the GRU, the hidden size of the layers in the GRU and the training patch segment length. The search is performed by independently and uniformly randomly sampling 32 points in the sets `hidden_size` $\in \{64, 65, \ldots, 512\}$, `num_layers` $\in \{1, 2, 3\}$ and `segment_length` $\in \{10, 11, \ldots, 60\}$. A sample of the results are shown in Table 4.4. The models were then ranked according to each metric and their ranks for each metric added up. The models were ordered by this total rank. The best model was found to have a hidden size $h = 201$, a single layer GRU and a segment length of $L = 28$, although the differences between models were relatively small. Such small differences might indicate that the model is learning something relatively simple and that increased model complexity will not help. We proceed with this model as the default for the rest of the experiments, referred to simply as the *CRNN*.

L	layers	h	frame	root	third	seventh	mirex	$\text{class}_{\text{mean}}$	$\text{class}_{\text{median}}$
28	1	201	**0.58**	**0.78**	0.75	0.62	**0.79**	0.18	0.01
23	2	295	0.58	0.78	**0.75**	**0.62**	0.78	0.19	0.02
14	2	374	0.57	0.77	0.74	0.61	0.79	0.19	**0.02**
56	1	463	0.58	0.77	0.74	0.61	0.78	0.19	0.02
42	3	222	0.57	0.77	0.74	0.60	0.79	0.16	0.00
...	...	...	...	...	...	...	...	...	

Table 4.4: CRNN model results on the large vocabulary with different hyperparameters. Best unrounded metrics are bolded. Results across all hyperparameters were very similar. We proceed with the best model found with hidden size of 201, a single layer GRU and a segment length of 28, although any configuration would work similarly. This also implies that model is learning something simple because the increased complexity of larger models did not help with performance. The variance within the data may be entirely due to the stochastic nature of SGD. Regardless of what makes the results different, the effect size is small enough that we conclude it does not make a difference.

Hyperparameters for CQT computation were chosen to be the same as in McFee and Bello (2017). However, the hop length was chosen to be 4096 samples. Other works have used 512 samples Jiang et al. (2019) or 2048 samples Rowe and Tzanetakis (2021). It should be noted that the metrics are not directly comparable across hop sizes as we are changing the number of frames and so the likelihoods are different. Nonetheless, if drastically different results are obtained, it may be worth using a different hop size. A plot of accuracy against log hop size is shown in teh Appendix A.3. The plot shows that performance is not affected by hop size much at all. This may be because the hope sizes used are all granular enough such that every chord has at least one frame associated with it, but not so granular that features per frame become less meaningful. We proceed with the hop size of 4096 samples in keeping with the literature and to keep computational cost low.

4.2.4 Model Analysis

4.2.4.1 Qualities and Roots

How does the model deal with the long tail of the chord distribution? Two of the metrics give some indication that the performance is poor. We use a confusion matrix over qualities of chords to provide more granular detail. The confusion matrix is shown in Figure 4.3.

We also looked at confusion matrices over roots. We do not illustrate these matrices as the model performs similarly over all roots with a recall between 0.74 and 0.82, approximately increasing with commonality of the chord. This aligns with the fact that the roots do not represent a long-tailed distribution as with the qualities, as seen in Figure 3.3. However, the two special symbols `N` and `X` have poorer performance, with recalls of 0.63 and 0.24 respectively. Many of the `N` chords are at the beginning and end of the piece. Clearly, the model struggles a little with understanding when music begins and ends. An example where the model mistakenly thinks chords are playing part-way through a piece is discussed in Section 4.2.4.6. The performance on `X` is to be expected. It is a highly ambiguous class with many possible inputs that are mapped to it, all of which will be fairly close to some symbol in the true vocabulary. It is unreasonable to expect the model to be able to predict this class well which further supports the argument for ignoring this class for evaluation.

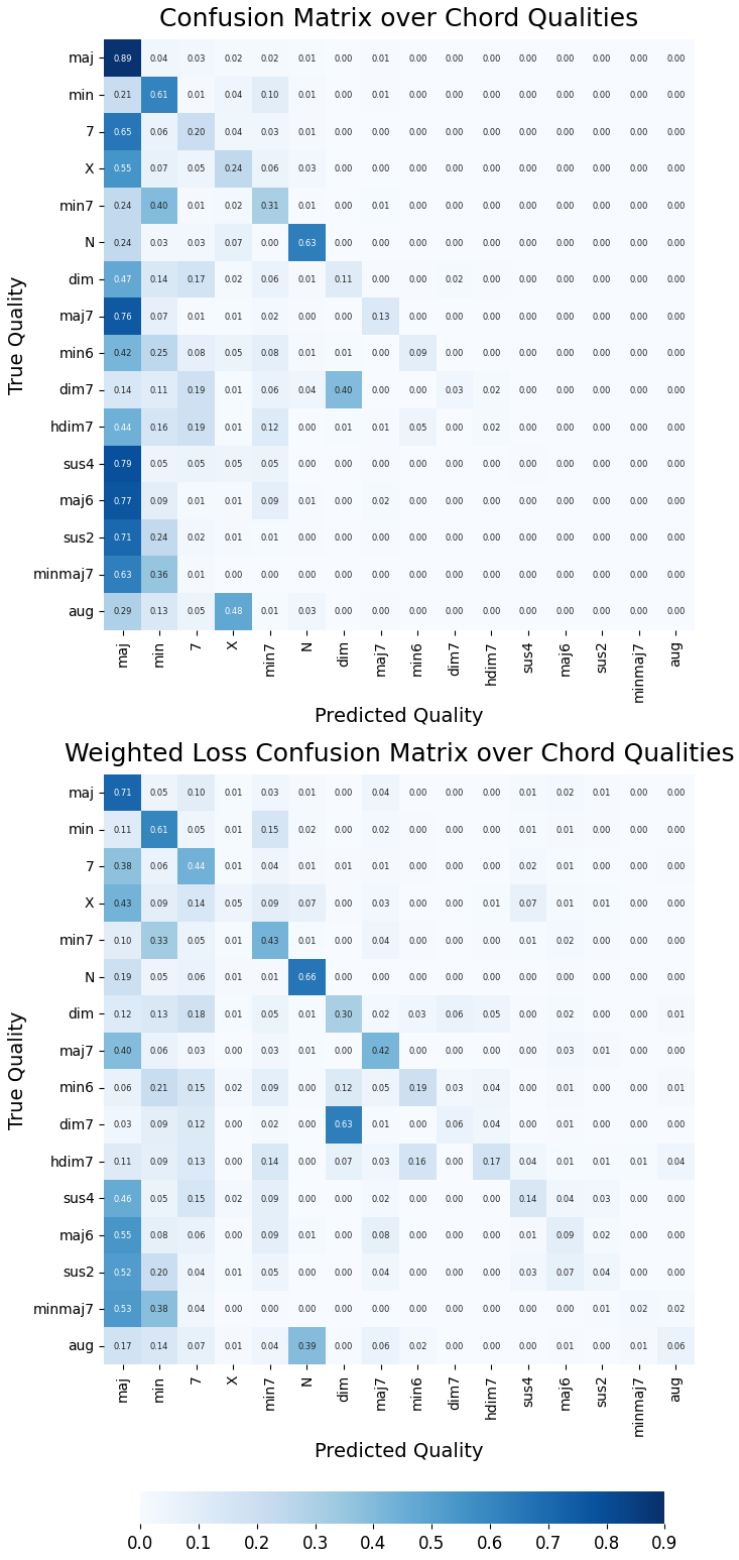

Figure 4.3: Row-normalised confusion matrices over qualities of the *CRNN* model without (above) and with (below) weighted loss. The weighting is with with $\alpha = 0.55$ as in Equation4.2. Rows are ordered by frequency of chord quality. We can see that both models struggle with the long tail. However, weighting the loss does improve the model, notably on `7` and `maj7` qualities and predicts `maj` less often. Recall on the `maj` worsens by 0.18 and recall on `X` decreases from 0.24 to 0.05. The weighted model predicts `X` approximately four 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. We can also see that both models frequently confuse `dim7` and `dim` qualities, consistently predict `maj` for `sus2, sus4, maj6, maj7` and `minmaj7` and struggles with `aug`.

4.2.4.2 Transition Frames

We hypothesise that the model is worse at detecting chords 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. To test this, we isolate the transition frames and compare the accuracies for transition and non-transition frames separately. We found that with a hop length of 4096, 4.4% of frames are transition frames. On the *CRNN* model with a `frame` of 0.59, the model achieves a `frame` of 0.36 on the transition frames and 0.60 on non-transition frames. Therefore, the model is certainly worse at predicting chords on transition frames. However, improving performance on these frames to the level of non-transition frames would only increase the overall frame-wise accuracy by less than 0.01.

Through manual investigation explained in Section 4.2.4.4 we found that on some songs the model did struggle to correctly identify the boundary of a chord change. This was not captured by the above metrics as if the boundary is ambiguous enough to span multiple frames, there may be a larger impact in accuracy than a single frame. Furthermore, some songs will have more obvious boundaries than others. Though the average may be low, for some songs this may be the main limiting factor in performance. However on such songs, the boundaries are likely to be highly subjective and the model would have to learn to detect the beat to consistently predict the boundaries. As previously discussed, we did not wish to introduce another failure mode by using a separate model for beat detection. Some work has attempted to simultaneously segment and predict This is a related task which we do not address here.

Given the above evidence that few frames are affected, and the difficulty of directly addressing the beat detection problem, we do not further investigate transition frames in this work.

4.2.4.3 Smoothness

Are the models outputs *smooth*? Most chords in the dataset last longer than a second but there are more than 10 frames per second. If many of the model's errors are due to rapid hesitation in chord probability, some post-processing smoothness could be applied. We use a crude measure of smoothness by looking at the number and length of incorrect regions. Such a region is defined as a sequence of frames with the same predicted chord where the model predicts the frames incorrectly. We find that of all incorrect regions, 26.9% are one frame wide. However, only 1.7% of frames that are incorrect are one frame wide. Thus, predicting correctly for all one-frame-wide regions would only improve the metrics by 0.017. A histogram over region lengths can be found in Appendix A.1.

We conclude that further work on the model to improve the smoothness would not improve performance significantly. However, when being used by a musician, smoothing predictions is valuable to make the chords more interpretable, and perhaps more correct. This is explored as a post-processing step in Section 4.8.

4.2.4.4 Performance Across the Context

How does the model perform across its context? We hypothesise that the model is worse at predicting chords at the beginning and end of a patch of audio as it has less context on either side. Analysing this will also help to understand if the model is putting its bi-directional context to good use.

To test this, we evaluate the model using the same fixed-length validation conducted during training as described in Section 3.3. We then calculate average frame-wise accuracies over the context and plot them in Figure 4.4. We use a segment length of 10 seconds corresponding to $L = 108$ frames. We can see performance is worst at the beginning and end of the patch, as expected, but not by much. Performance only dips 5%, 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.

We conducted a further experiment, measuring overall accuracy with increasing segment lengths used during evaluation. Indeed accuracy increases, but only by a tiny 0.005 from 5 seconds to 20 seconds and is flat thereafter. Results can found in Appendix A.2. We conclude that the context length does not make a big difference and continue to evaluate the model over the entire song at once.

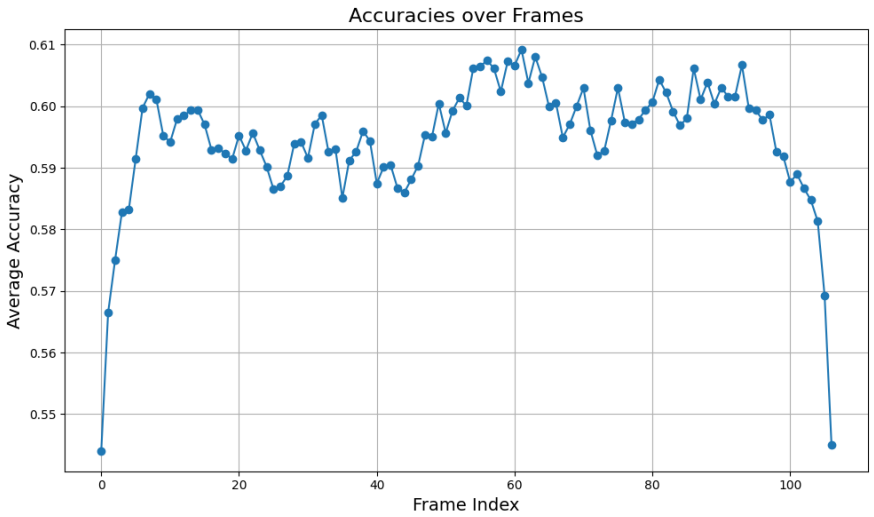

Figure 4.4: Average frame-wise accuracy of the *CRNN* model over the patch of audio. The model performs worse at the beginning and end of the patch of audio, as expected. However, the differences are only 5%. We propose that the context on one side is enough. This plot supports our procedure of evaluating over the entire song at once.

4.2.4.5 Generalising Across Songs

Does the model do well consistently over different songs? We plot a histogram of accuracies and mirex scores over songs in the validation set in Figure 4.5. We find that the model has very mixed performance with accuracy, with 17% of songs scoring below 0.4. However, when we use the more generous `mirex` metric, almost all of the scores below 0.4 are improved, and only 6% below 0.6. Many of the mistakes that the song

makes are a good guess in the sense that it may have omitted a seventh or mistaken a major 7 for its relative minor. Examples of such mistakes are discussed in Section 4.2.4.6. We conclude that, in general, the model's outputs are reasonable predictions, but lacks the detail contained in good annotations like correct upper extensions of the chords.

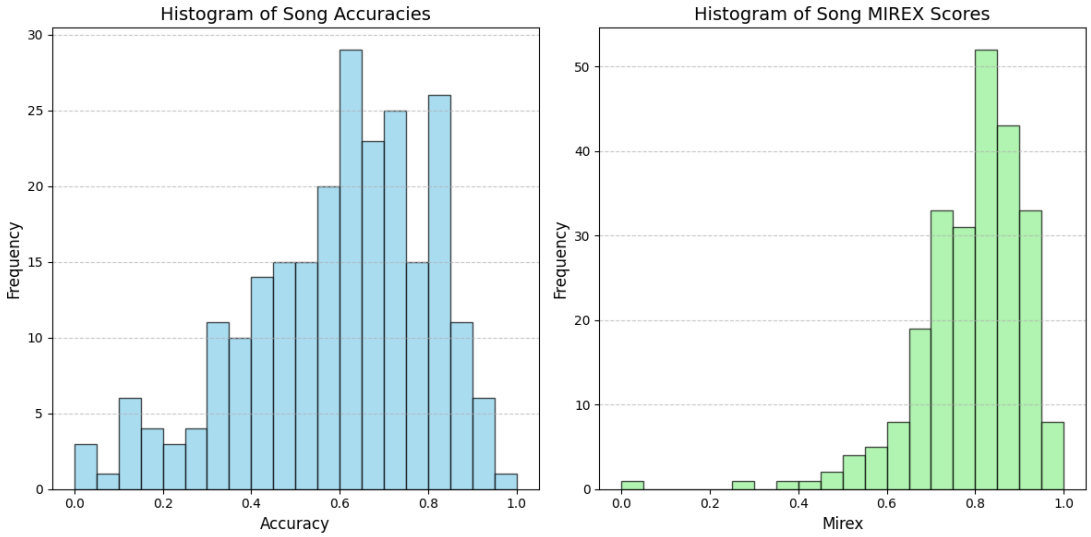

Figure 4.5: Histogram of accuracies and mirex scores over songs in the validation set. Accuracies are mixed, with 17% of songs below 0.4, and 64% between 0.4 and 0.8. However, with the more generous `mirex` metric, we find that there are almost no songs below 0.4 and only 6% below 0.6. Many of the mistakes the model makes are small, like predicting `C:maj` instead of `C:maj7`. Such examples are discussed in more detail in Section 4.2.4.6. The very low outliers in the `mirex` score were found to be songs with incorrect annotations found in Section 3.1.2.1.

4.2.4.6 Four Illustrative Examples

Let us now inspect a few songs to see how the model performs. We choose four examples showing different typical behaviours. We show illustrations of frame-by-frame correctness as measured by both accuracy and `mirex` in Figure 4.6.

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`, which is an understandable mistake to make, especially after hearing the song. As evidenced by the confusion matrices in Figure 4.3, this mistake very fairly common on qualities like `sus2`. 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. In this song, many of these are a missing seventh such as `A:min` 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`. 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 the 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 often predicts completely wrong chords. This is a song where the model's outputs are less sensible. 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 confuses qualities that can be easily mistaken for major or minor chords. Sometimes the model makes mistakes in the boundaries of chord changes, and sometimes it predicts completely wrong chords, although these are on songs which are more difficult to annotate.

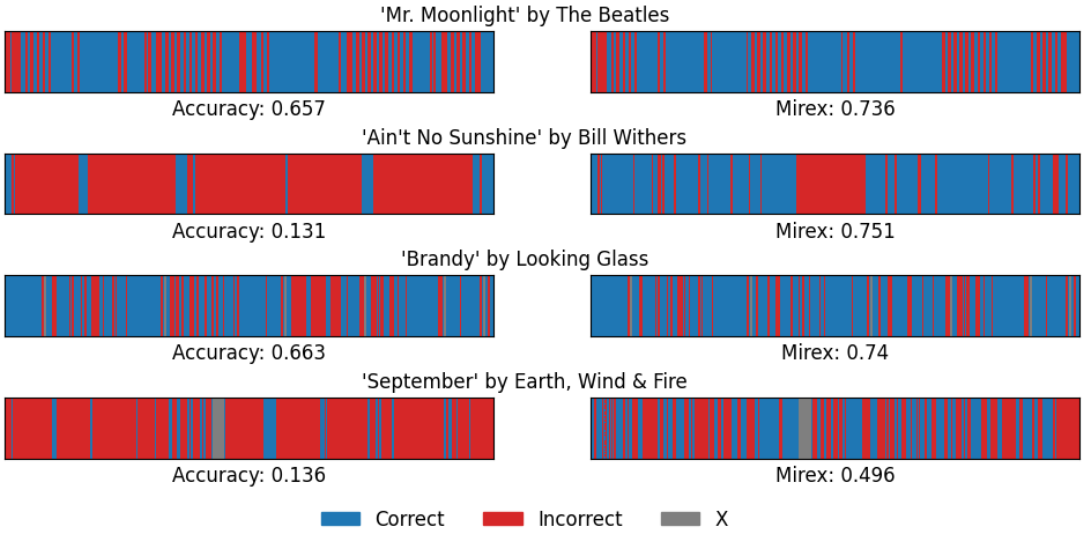

Figure 4.6: Chord predictions of the *CRNN* model on four songs from the validation set (blue: correct, red: incorrect, gray: `X`). This allows to understand some of the behaviour of the model. We can see regular repeated errors in 'Mr. Moonlight', which are mostly mistaking two similar qualities. The discrepancy between accuracy and `mirex` on 'Ain't No Sunshine' can be explained by missing sevenths in many predictions in 'Ain't No Sunshine'. 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.

4.3 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. Sampling is explored by Miller et al. (2022). However, they use a different re-weighting scheme 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), and we employ a similar, but simpler implementation here. Rowe and Tzanetakis (2021) explore the use of curriculum learning as form of re-sampling which we do not explore here.

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.

$$w_c = \frac{1}{(\text{count}(c)+1)^{\alpha}} \tag{4.1}$$

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

$$w_c^* = \frac{w_c}{s} \text{ where } s = \frac{\sum_{c \in C} \text{count}(c) \cdot w_c}{\sum_{c \in C} \text{count}(c)} \tag{4.2}$$

Where 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 α in the set $\{0, 0.05, 0.1, \ldots, 0.95, 1\}$. The plot in Figure 4.7 shows the effect of the weighting on the model's performance.

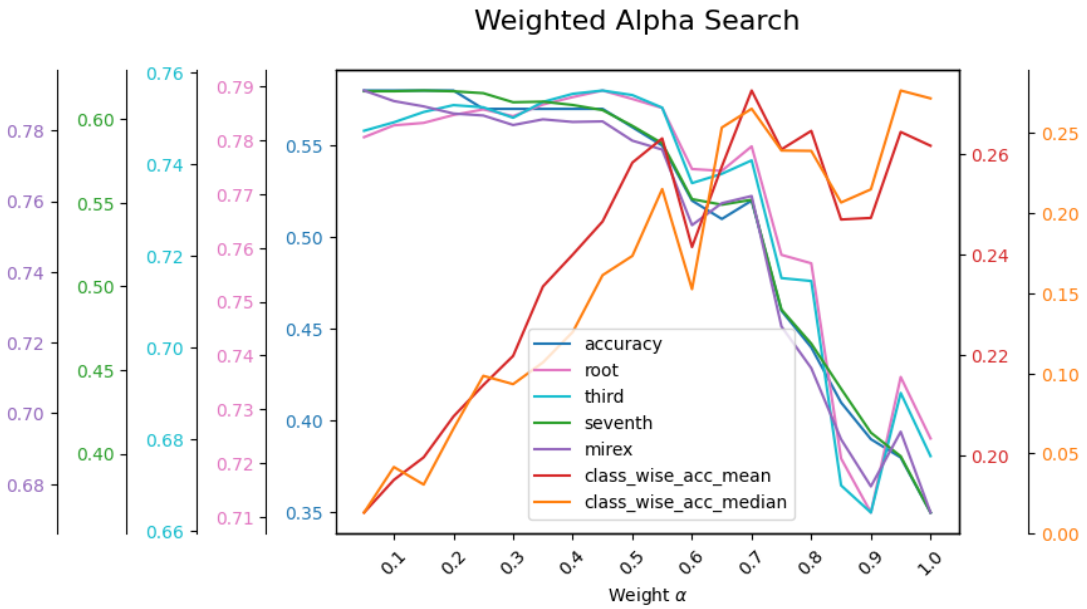

Figure 4.7: Effect of weighted loss on the *CRNN* model with varying α. As we increase α, class-wise metrics improve but accuracy-based metrics worsen. We claim a sweet-spot in the middle where we trade only a little overall performance for better class-wise recall. We choose this to be $\alpha = 0.55$. 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.

4.4 Pitch Augmentation

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

4.5 Structured Training

- Add structured loss and retrain. From literature. Should (?) further improve accuracies on sevenths etc.

4.6 Transformer

- Make own transformer architecture, similar to BTC or HarmonyTransformer, or Curriculum learning. How to train and evaluate?

4.7 Using 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.

4.8 Decoding

- HMM - CRF - Something ad-hoc?

4.9 Results on Test Set

- Directly compare CRNN, weighted loss, pitch augmentation, structured, transformer, generative features on the test set.

Chapter 5

Synthetic Data Generation

5.1 Performance on JAAH

- The performance of existing models on JAAH

5.2 Generation method

- Generation method.

5.3 Experiments

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

5.4 Results

- Comparison on normal data - Comparison on JAAH

5.5 Qualitative Analysis

- Qualitative analysis of the results

Chapter 6

Conclusions, Limitations and Further Work

6.1 Conclusions

- What do the results say?

6.2 Limitations

Limitations: - Genre - Standard tuning, Western - No lyrics - No beat/segmentation detection - No chord inversion - Size of vocabulary lacking leading to tricky X chord

6.3 Further Work

Further work: - More data e.g. HookTheory - Larger vocabulary? Inversion as in? Why didn't I do this? - Make into a useful tool by outputting chords in a more useful format - Investigate other methods of synthetic data generation? - Stuff I didn't have time to incorporate e.g. beat/segmentation detection, transformers, structured loss, curriculum learning

Bibliography

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.

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/.

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

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.

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.

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.

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, 2012. 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.

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.

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.

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

Kyogu Lee and Malcolm Slaney. Automatic chord recognition from audio using a hmm with supervised learning. pages 133–137, 01 2006.

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.

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.

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.

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.

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.

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.

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 Histogram over Incorrect Region Lengths

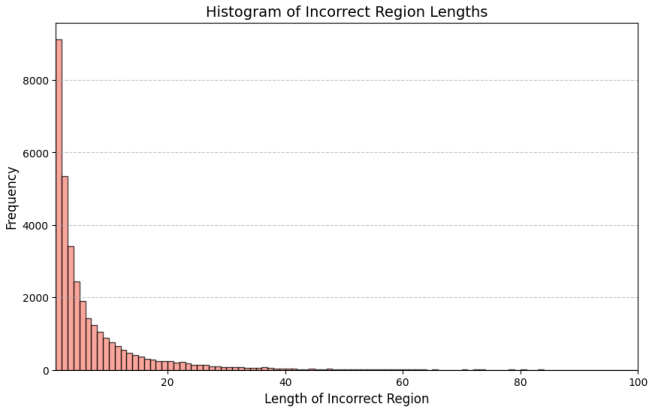

Figure A.1: Histogram over incorrect region lengths. The incorrect region distribution has a long-tail, with 26.7% regions being of length 1. This raises concerns over the smoothness of outputs and requires some form of post-processing explored in Section 4.8.

A.2 Accuracy vs Context Length of Evaluation

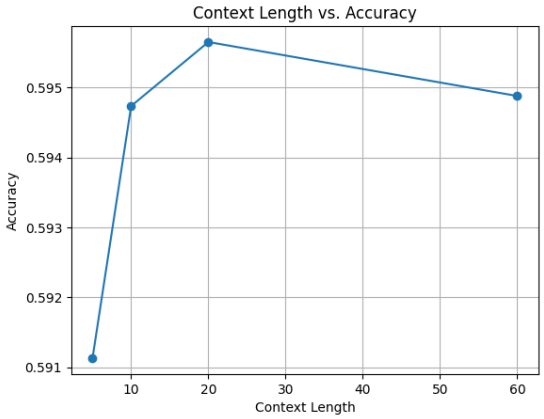

Figure A.2: Accuracy vs context length of evaluation. The accuracy increases very slightly. The effect size is so small that we conclude it does not make a difference, and choose to evaluate over the entire song at once.

A.3 Accuracy vs Hop Length

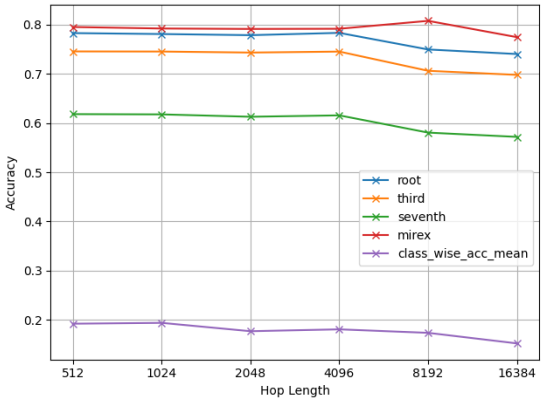

Figure A.3: Accuracy vs hop length. Metrics are not directly comparable over hop lengths due to different likelihoods. However, the metrics are fairly consistent over different hop lengths, certainly over the region explored by the literature $[512, 2048, 4096]$. Every hop length tested is short enough to be more granular than chords, but not so short that the computed CQT is too noisy. We continue with the default hop length of 4096, to be consistent with some of the literature while keeping computational cost low.