Chord Recognition with Deep Learning

Pierre Lardet

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

2025

Abstract

Progress in automatic chord recognition has been slow since the advent of deep learning in the field. In order to understand why, I conduct experiments on existing methods and test hypotheses enabled by recent developments in generative models. Findings show that chord classifiers still have poor performance on rare chords and that pitch augmentation provides the biggest boost in accuracy. Features extracted from generative models do not help but synthetic data presents an exciting avenue for future work. I conclude by improving the interpretability of model outputs with beat detection, reporting some of the best results in the field and providing qualitative analysis. Much work remains to be done to solve automatic chord recognition but I hope that this thesis charts a path for others to try.

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

Thank you to my two lovely supervisors, Prof. Amos Storkey and Dr. Alec Wright, for allowing me to fulfil my desire for such a personally motivating project. They offered guidance on this project and research at large which I will take forward. I would like to thank Andrea Poltronieri for sharing the dataset used in this work. Finally, I would like to thank my friends and family for providing me with reasons to take a break.

Table of Contents

Chapter 1

Introduction

Chords form an integral part of music. Part of how musicians understand music is through harmonic structure. Chord annotations allow music to be easily shared, performed, improvised and analysed. Not all chord annotations available online are free or of a high enough quality. This is because creating high quality chord annotations requires a trained musician.

To this end, I investigate the use of deep learning for automatic chord recognition. Data-driven methods have dominated the field of all automatic music transcription for over a decade. However, the significant progress of early models has not continued in recent years while the problem remains far from solved.

In this work, I first aim to understand why. I implement a common benchmark model and conduct a thorough analysis of its behaviour. This involves looking at common mistakes the model makes, performance on rarer chords and how its predictions relate to time. I then use these observations to study the various ways models have been improved and compare them.

I also conduct novel research on the use of generative models as both feature extractors and a source of new data. This is enabled by chord-conditioned generative models developed in recent years. I conclude by re-thinking how the model predicts chords in time by incorporating beat estimation.

The analysis of existing models, exploration of improvements and discussion of new research directions constitutes a novel contribution to the field of automatic chord recognition. Despite lack of performance improvements in recent years, I hope that this work provides motivation for others to continue pursuing research aiming to solve the problem posed by chord recognition. All code is available on Github.[1] Data can be made available upon request.[2]

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

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

1.1 Outline

The thesis is structured as follows:

- **Chapter 2** provides background information on harmony, chord recognition and musical data. I then discuss existing literature on the subject, pointing out trends in the field and the most exciting avenues for research. In particular, I highlight motivation for the research carried out in this work.
- **Chapter 3** describes the datasets, evaluation metrics and training procedure used.
- **Chapter 4** contains the implementation of a convolutional neural network from the literature followed by an analysis of its properties and predictions. I observe behaviours which provide opportunities to improve the model.
- **Chapter 5** extends this work by studying various methods of improving on this model. Some of these experiments analyse existing improvements while others present novel avenues of research.
- **Chapter 6** concludes the thesis 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 and models that are commonly used in ACR, the challenges that are faced in this field and future directions.

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. Chords can be thought of as a collection of at least two notes built from a root note and scale. Any notes from the scale can be present but the most common are the third, fifth and seventh. A chord's *quality* is determined by the intervals between notes in the chord irrespective of the root note. The most common qualities are major and minor. Many other qualities exist such as diminished and suspended. Chords can be played in *inversion*, where the root note is not the lowest Note. In this work, chords are represented using Harte notation [1].

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 share many notes. For example, `G:maj` and `E:min` are related in this way. It is possible for chords to share the same set of notes like `G:maj6` and `E:min7`.

Chords are an important part of music. They provide harmonic context for a melody and can be used to convey emotion, tension and release [2]. They are also important for improvisation where musicians will play notes that fit the chord progression [3]. Contemporary guitar music is often represented by a chord sequence. Chords are also important for songwriting and production where a chord progression can form the basis of a song. Music analysis also makes heavy use of chords. The harmonic structure of a piece can be analysed to understand the composer's intentions and why we enjoy the music [4].

2.1.2 Chord Recognition

Chord recognition is the task of identifying which chord is playing at any moment in a piece of music. This can be useful for creating notated versions of songs for musicians, musicologists and music recommendation. Those wishing to learn a song may visit 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. Music recommendation systems can recommend songs based on their harmonic content as similar music will often have similar harmonic content [5]. 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.

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 [6]. The task of annotating chords is time-consuming and requires a trained musician [7]. Automatic chord recognition systems have the potential to alleviate these problems by providing a fast, accurate and scalable solution.

Unfortunately, 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 many ways on different instruments. Precisely when a chord starts and ends can be imprecise. Whether a melody note is part of a chord and whether a melody alone is enough to imply harmonic content are both ambiguous. In order to identify a chord, data across time must be considered. For example, a chord may be vamped or arpeggiated. Audio also contains many unhelpful elements for chord recognition such as reverb, distortion and unpitched percussion.

2.1.3 Music Features

Recorded music can be represented in a variety of ways on a computer. The simplest is as a raw time-series of amplitudes, referred to as the audio's waveform. Data in the raw audio domain has been applied in generative models such as Jukebox [8] and MusicGen [9] and autoencoders [10].

Spectrogram: A spectrogram is a transformation of a waveform into the time-frequency domain calculated via a short-time Fourier transform (SFTF). Spectrograms are commonly used in many audio processing tasks such as audio search [11] and music transcription [12]. As of yet, linear spectrograms computed using the STFT have not been used in ACR tasks [13].

CQT: A common version of the spectrogram used in music transcription is the constant-Q transform (CQT) [14]. 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

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

[2] `https://www.youtube.com/watch?v=oOlDewpCfZQ` accessed 25th February 2025

ACR [15, 16]. An example CQT from the dataset used in this work is shown in Figure 2.1. As Korzeniowski and Widmer [17] 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 shift of the CQT bins. Another logarithmic variant of the spectrogram is the mel-spectrogram, based on the mel-scale [18]. It is intended to mimic the human ear's perception of sound and is commonly used in speech recognition [19] but has also been used in music transcription tasks [20].

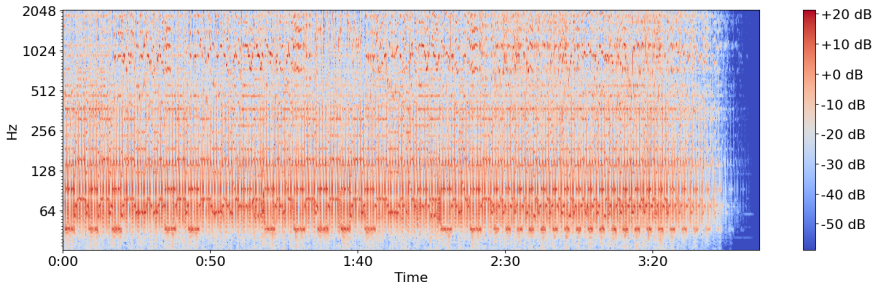

Figure 2.1: CQT of 'Girls Just Wanna Have Fun' by Cyndi Lauper. We can see the log-spaced frequency bins on the y-axis. There is clear structure and repetition in the song, particularly in the lower frequencies, which can be attributed to a regular drum groove. Such structure gives an idea of the patterns 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. Such features have been generated by deep learning methods [21] or by hand-crafted methods [22, 23] and have seen use in recent ACR models [24]. 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 [25]. Chris Donahue and Liang [20] use features from JukeBox [8] to train a transformer [26] 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 nor with CQTs.

2.2 Related Work

The field of ACR has seen considerable research since the seminal work of Fujishima [27] 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. [7]
- *Isophonics*: 300 annotations of songs from albums by The Beatles, Carole King and Zweieck. [28]
- *RWC-Pop*: 100 pop songs with annotations available[3] for chords. [29]
- *USPop*: 195 annotations of songs chosen for popularity. [30]
- *JAAH*: 113 annotations of a collection of jazz recordings. [31]
- *HookTheory*: 50 hours of labelled audio in the form of short musical segments, crowdsourced from an online forum called HookTheory[4]. [20]

Many of these have been compiled together into the *Chord Corpus* by de Berardinis et al. [6] with standardised annotation formats. However, audio is scarce, in part due to copyright issues. 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.

Another problem is that the existing data is imbalanced, with a large number of major and minor chords and fewer instances of chords with more obscure qualities. This can lead to models that are biased towards predicting major and minor chords. Attempts to address such an imbalanced distribution have been made by weighting the loss function [32], adding 'structure' to the chord targets [16, 32], re-sampling training examples to balance chord classes [21] and curriculum learning [33].

Pitch Augmentation: Due to the lack of labelled data, data augmentation via pitch shifting has been applied to ACR. Input features are pitch shifted while chords are transposed. McFee and Bello [16] note the large increase in performance using pitch shifting directly on the audio. Other works have since used pitch shifting directly on the CQT [32]. No work has compared the two methods.

Synthetic Data Generation: Data has been scaled up using augmentation and semi-supervised learning with some success [34]. Research has been done into the use of synthetic data [35, 36] and self-supervised learning [37] for other MIR tasks but not for ACR. With the advent of new models which accept chord-conditioned input [38], the possibility of generating synthetic data for ACR is an exciting avenue of research.

2.2.2 Models

Model Architectures: Since the work of Humphrey and Bello [39], chord recognition has been predominantly tackled by deep learning architectures. The authors used a convolutional neural network (CNN) to classify chords from a CQT. CNNs have been combined with recurrent neural networks (RNNs) [40, 32, 16] with a CNN performing

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

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

feature extraction from a spectrogram and an RNN sharing information across frames. More recently, transformers have been applied in place of the RNN [20, 24, 41, 42, 33, 43].

Despite increasingly complex models being proposed, performance has not improved significantly. In fact, Park et al. [43] found that their transformer performed marginally worse than a CNN. Humphrey and Bello [44] 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. [38] use the model developed by Park et al. [43] 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. [21] 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 [32]. 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. [45] 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 [46] 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 [17] 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 [16] found that similar qualities are often confused with each other and that the model favours the most common chord qualities. Park et al. [43] 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.

2.2.3 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 [32] up to 4096 [16]. Which hop length works best remain unclear.

More recently, Chris Donahue and Liang [20] 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. [45] and Cho and Bello [47] 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. Durán and de la Cuadra [48] jointly estimate beats and chords but use a different jazz-specific dataset and do not conduct analysis on how beat-wise predictions affect performance.

2.2.4 Future Directions

Pauwels et al. [13] provide an overview of ACR up to 2019 since the seminal work of Fujishima [27] 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 [16]. 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

The initial period of this project was spent finding a suitable dataset for training and testing. Eremenko et al. [31] use the *JAAH* dataset while Chris Donahue and Liang [20] use the *HookTheory* dataset. Many works use a combination of the *McGill Billboard*, *Isophonics*, *RWC-Pop* and *USPop* datasets. However, none have audio available. Furthermore, annotations come from different sources in different formats. I spent time looking at scraping audio data, 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, one of the authors of the chord corpus or 'ChoCo' for short [6]. 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 several weeks into the project.

The dataset used in this work is referred to as *pop*. It consists of songs from the *Mcgill Billboard*, *Isophonics*, *RWC-Pop* and *USPop* datasets introduced in Section 2.2.1. This collection was originally proposed in work by Humphrey and Bello [44] in order to bring together some of the known datasets for chord recognition. The dataset consists of subsets from the above sources filtered for duplicates and selected for those with annotations available. In total, there are 1,213 songs. The dataset was provided with obfuscated filenames and audio as `mp3` files and annotations as `jams` files [49].

3.1.1 Data Integrity

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

Duplicates: Files are renamed using provided metadata identifying them by artist and song title. This is done to identify duplicates in the dataset. There is only one: Blondie's 'One Way or Another' which has two different recordings. It is removed from the dataset. Automatic duplicate detection is conducted by embedding each audio using mel-frequency cepstral coefficients (MFCC) [50]. This function is commonly used to embed audio into low dimensions. This provides a fast and easy way of quantifying similarity. Audio is passed through the `mfcc` provided in `librosa` [23] with 20 coefficients. A song's embedding is calculated as the 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. Misaligned annotations could make training impossible. 10 songs are manually investigated for alignment issues by listening to the audio and comparing it to the annotations directly. The annotations are all well-timed with detailed chord labels.

Automatic analysis of the alignment of the audio and chord annotations is also done using the cross-correlation between the derivative of the CQT over time and the chord annotation. 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 is estimated using the `delta` function from `librosa`. The chord annotations are 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 are normalised by subtracting the mean and dividing by the standard deviation. Finally, cross-correlation is computed using the `correlate` function from `numpy`. A typical cross-correlation for a song is visualised in Appendix A.3. The cross-correlation is periodic and repeats every 20 frames or so. Listening to the song, the period of repetition is a fraction of a bar-length.

To check alignment across the dataset, a histogram of the lag of the maximum cross-correlations over songs is plotted in Figure 3.1. Under the assumption that the annotations are not incorrect by more than 5 seconds, the region of possible maximal lags is restricted to a window of 50 frames either side of 0. 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 is 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 is convincing enough to leave the annotations as they are for training.

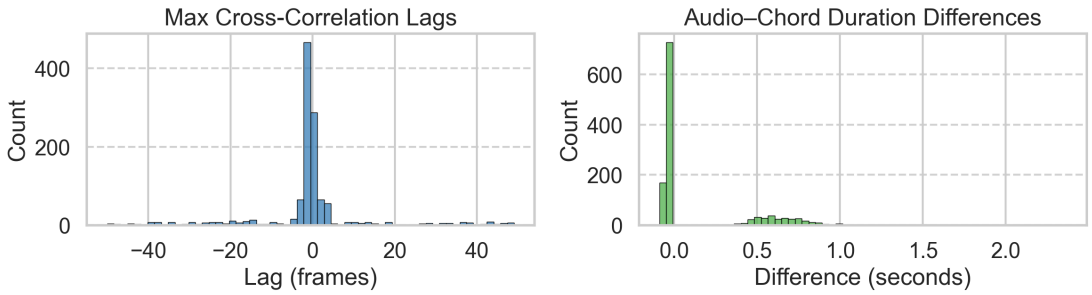

Figure 3.1: Histograms of the maximum cross-correlation lags and the difference in length between the audio and chord annotations. Both show results close to 0 suggesting good alignment between audio and annotations.

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 [44]. 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 [16]. 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 Preprocessing

3.1.2.1 Audio to CQT

I first convert the audio to a constant-Q transform (CQT) representation explained in Section 2.1.3. This feature is common in ACR and is used as a starting point for this work. The CQT was computed using `librosa`, using the built-in `cqt` function. A sampling rate of 44100Hz was used, with a hop size of 4096, 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 [16] 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.

3.1.2.2 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 [1]. 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 [16, 44, 32]. Jiang et al. [32] use a more detailed vocabulary by also including inversions but I decide to remain consistent with previous works. As McFee and Bello [16] 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. The method for converting from Harte notation to a symbol in the chord vocabulary is similar to that used by McFee and Bello [16] and is detailed in Appendix A.7.

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.6. 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.

Frames are allocated a chord symbol based on which chord is playing at the middle of the frame. While this may not be a perfect solution, frames are ≈ 93ms long which is

shorter than the minimum duration of a chord in the dataset. This guarantees that the chord label for every frame plays for at least half the frame. Furthermore, only 4.4% of frames include a chord transition.

3.1.3 Chord Distribution

Much of the recent literature has focused on the long tail of the chord distribution, using a variety of methods to address the issue. It is first helpful to understand the distribution of chords in the datasets, illustrated in Figure 3.2. 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 two or three orders of magnitude less common. The distribution over roots is far less skewed, although there is a preference for chords in keys with common roots like A, C and D.

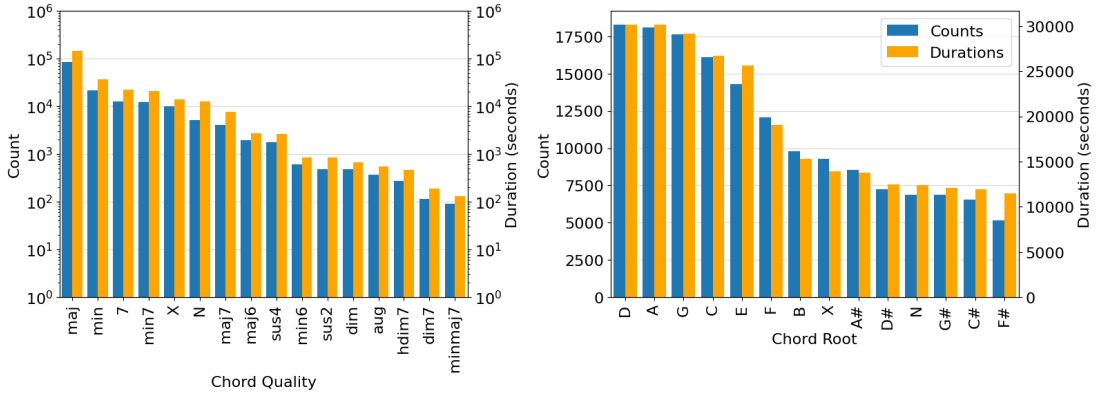

Figure 3.2: Chord distributions over qualities (left) and roots (right) in the *pop* dataset. The plots show both the raw counts in frames and the duration in seconds for each chord root/quality. Note that the y-axis over qualities is in a logarithmic scale. We observe that the qualities are very imbalanced, with `maj` as the most popular. Conversely, roots are relatively balanced.

3.2 Evaluation

As is standard for ACR, I use weighted chord symbol recall (WCSR) to evaluate chord classifiers. Simply put, WCSR measures the fraction of time that a classifier's predictions are correct. I include a formal definition in Appendix A.2. Correctness can be measured in a variety of ways such as `root`, `third` and `seventh` which compare along roots, thirds, or sevenths respectively. I also use 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`. Finally, I use `acc` to denote the overall accuracy where symbols must match exactly.

Other measures of correctness are sometimes used. These include `majmin`, a measure of correctness over only major and minor qualities. I use this measure only to substantiate the use of the larger vocabulary in Appendix A.6. 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 verified empirically on preliminary experiments which are omitted.

All metrics all implemented in the `mir_eval` library [51] which also provides utilities for calculating WCSR from frame-wise chord predictions. The mean WCSR is computed over all songs in the evaluation set. 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 [44]. I report only the mean throughout this work. This was chosen for being more commonly used in recent literature and because it is important for a metric to detect if the model performs poorly over certain genres.

For some experiments, two further metrics are calculated. 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. [32]. I redefine $\texttt{acc}_{\text{class}}$ here in terms of WCSR and introduce $\texttt{median}_{\text{class}}$. The definitions can be found in Equations 3.1.

$$\text{acc}_{\text{class}} = \frac{1}{C}\sum_{c=1}^{C} WCSR(c) \qquad \text{median}_{\text{class}} = \text{median}_{c=1}^{C}[WCSR(c)] \tag{3.1}$$

C denotes the number of chord classes. $WCSR(c)$ is the WCSR considering only time when chord c is playing. A formal definition in included in of class c is also included in Appendix A.2.

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 interpret.

The justification for redefining $\texttt{acc}_{\text{class}}$ this way is that metrics calculated over discrete frames are not comparable across different frame lengths and are dependent on the method of allocating chords to frames. Continuous measures more closely reflect what we truly desire from the model. To illustrate this, imagine a very long frame length. The model could have perfect scores on these frames but be making terrible predictions for much of the song. Through preliminary experiments, it became clear that with sufficiently small hop lengths, there are negligible differences with continuous measures. Nevertheless, I propose that the field of ACR adopts a continuous measure of class-wise accuracy.

I do not also compute *quality*-wise accuracies as computed by Rowe and Tzanetakis [33]. 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

select models in Section 5.7.

Other evaluation tools are used such as confusion matrices and the number of chord transitions per song. Note that confusion matrices are 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 two for short 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 as performance was found to be marginally better. 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 a 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 Eddie, The University of Edinburgh's research compute cluster. Experiments on Eddie are run on CPUs due to the lack of availability of GPUs.

Training code is implemented in `PyTorch` [52]. Unless stated otherwise, models are trained with the Adam optimiser [53] 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 and train for 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 time. The model is saved 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 [44]. 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. For final testing, models are re-trained on the combined training and validation sets and tested on the test set.

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 [16], referred to as the *CRNN*. It remains competitive with state-of-the-art, is often used as a comparative baseline and is 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 [54] before being fed through two convolutional layers with a rectified linear unit (ReLU) after each one. The first convolutional layer has a 5×5 kernel and outputs one channel. It is intended to smooth out noise and spread information 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. The output is passed through a bi-directional gated recurrent unit (GRU) [55], with hidden size initially set to 256 and a final fully connected layer with softmax activation. This produces a vector of length C for each frame. The chord with the maximum probability is taken as the model's prediction for each frame.

The authors of the model propose using a second GRU as a decoder before the final fully connected layer, called 'CR2'. In brief empirical tests, the results with and without 'CR2' were very similar. I therefore do not include this in model. Results are left to Appendix A.8 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.

Detailed metrics are left to Appendix A.4 but results are summarised here. The best performing model was found to be with `lr=0.001`. Performance over different schedulers is very similar. I proceed with a learning rate of 0.001 and `cosine` scheduling for the rest of the experiments. Scheduling with `cosine` is chosen as it provides a consistent learning rate reduction. It empirically was found that plateau would sometimes only reduce the learning rate in the final epochs. Only saving on validation loss improvement provides the regularisation effect of early stopping regardless of choice of scheduler.

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. Figures supporting this conclusion can be also be found in Appendix A.4. 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 proceed with training for 150 epochs without early stopping.

Results from Reddi et al. [56] suggest that stochastic gradient descent (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.9 for lack of interest.

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.10.

A sample of the results are shown in Table 4.1. Results across hyperparameters are relatively stable. 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 where increased model complexity does not help. It also seems that the choice of hyperparameters is somewhat arbitrary. The models do 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 [16] 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 [16] 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 4.1: A subset of *CRNN* model results on the large vocabulary with different hyperparameters. Best results for each metric are boldfaced. 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 A.1, it seems that the parameters suggested by McFee and Bello [16] 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. 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. Saving on validation loss improvement also provides a regularisation effect.

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 is placed between each layer. These are followed by a 36-channeled $1 \times I$ convolutional layer and fully connected layer as in the *CRNN*.

I test models of increasing depths, kernel sizes and channels. In general, the deeper

models perform 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 the *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 perform a grid search over learning rates and schedulers for these baselines to ensure that convergence was reached. Convergence results are 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 First Results

Table 4.2 shows the results of the *CRNN* compared with the baseline models. The *CRNN* performs the best out of these models. The GRU layer improves accuracy by 5.2%. However, similar performance increases can be achieved by adding convolutional layers as in *CNN5* as opposed to 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 roughly 60%. Indeed, the best models trained by Park et al. [43] and by Akram et al. [42] never achieve an accuracy of more than 66%. Humphrey and Bello [44] refer to this as the 'glass ceiling' which the field of ACR is still struggling to break through. The problem posed by ACR remains far from solved.

Korzeniowski and Widmer [46] train a deep CNN which remains competitive with state-of-the-art to this day. It contains 8 layers. Park et al. [43] find that the performance of this deep CNN is very similar to that of the *CRNN*, both reaching accuracies of around 65%. Training much deeper convolutional networks was found to be more computationally expensive than training the *CRNN* with little performance gain to be had. Therefore, I proceed with the *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 4.2: Results for *logistic*, *CNN1*, *CNN5* and *CRNN*. We see that the *CRNN* performs the best on nearly all metrics. The *CNN5* performs almost as well. This suggests that shallower CNNs can reach similar performance as the deep CNN trained by Korzeniowski and Widmer [46].

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 seek to understand behaviour of the model by answering a series of questions.

4.4.1 Qualities and Roots

How does the model deal with imbalanced chord distribution? The class-wise metrics in Table 4.2 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 4.1. 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 the model consistently predicts major for similar chord qualities like major7 and sus4. A similar effect is observed with minor chords and qualities like minor7. The model frequently confuses diminished 7 chords for diminished chords. This explains the median class-wise accuracies of nearly 0 for all models.

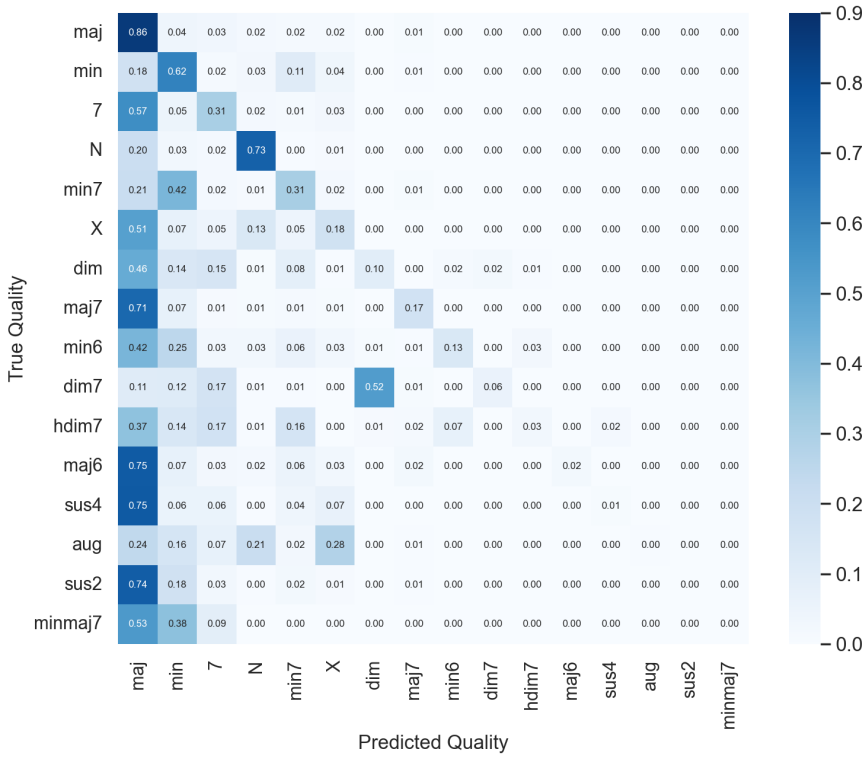

Figure 4.1: Row-normalised confusion matrices over qualities of the *CRNN* model. Rows are ordered by frequency of chord quality. We observe that the model struggles with the imbalanced distribution. It frequently confuses `dim7` and `dim` qualities, consistently predicts `maj` for `sus2, sus4, maj6, maj7` and struggles with the rare qualities like `minmaj7` and `aug`.

I also produce a confusion matrix over roots. This is left to Appendix A.11 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. 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 4.4.6.

Performance is 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 completely different sounds mapped to it. All of the chords mapped to `X` will share many notes with at least one class in the known portion of the vocabulary. It is therefore 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.

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, the *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 at most 1%.

Through qualitative evaluation discussed in Section 4.4.6, 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.12. 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 would be valuable to make the chords more interpretable.

4.4.4 Performance Across 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.13. 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.14. 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 evaluate over the entire song at once.

4.4.5 Consistency Over Songs

Does the model have consistent performance over different songs? The set of accuracies over songs of the *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 4.2. 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. These 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 4.4.6.

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.

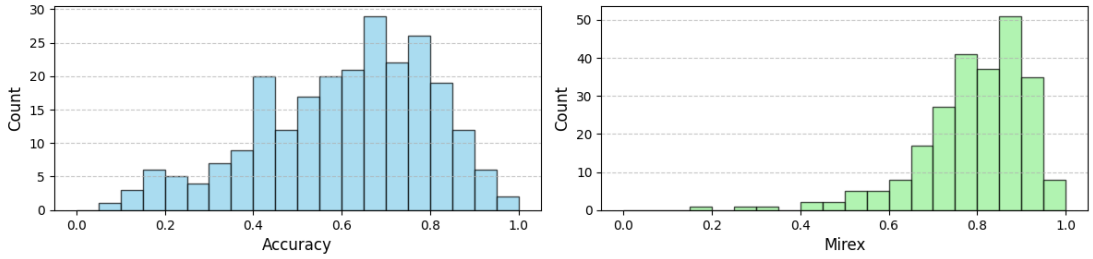

Figure 4.2: Histogram of accuracies and mirex scores over songs in the validation set. Accuracies are mixed, with 15% of songs below 40%, and 69% between 0.4 and 0.8. However, with the more generous `mirex` metric, we find that there are almost no songs below a score of 40% and only 7% below 0.6. This 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 3.1.1.

4.4.6 Four Illustrative Examples

Let us now inspect a few songs to see how the model performs. I choose four examples showing different behaviours and failure modes of the model. Predictions are illustrated frame-by-frame and coloured by correctness, as measured by both accuracy and `mirex` score in Figure 4.3.

In *Mr. Moonlight*, there are few differences between the accuracy and `mirex` score. 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 rapidly alternates between `F:maj` and `F:sus2`. The confusion matrix in Figure 4.1 suggests this mistake is very fairly common on qualities like `sus2` which are similar to `maj`.

In *Ain't No Sunshine*, the `mirex` score is significantly higher than the accuracy. This is because the majority of the mistakes the model makes are missing 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 predicting `E:min7` when the chord is `G:maj`. All of these mistakes occur frequently in this song. The mean difference between the accuracy and mirex is 18.7%, with one song reaching a difference of over 70%. 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` throughout this section. This is a sensible label as when this melody is sung elsewhere in the song, it is labelled as `A:min7`.

In the next two songs, *Brandy* and *September*, the model's mistakes are less interpretable.

While performance is okay on *Brandy* with a `mirex` of 75.6%, 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.

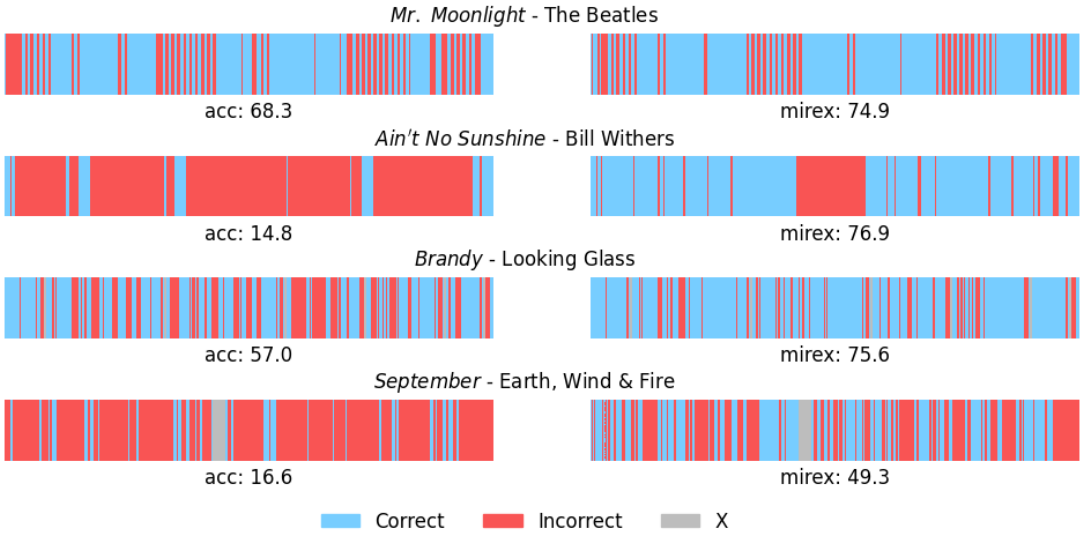

Figure 4.3: Chord predictions of the *CRNN* model on four songs from the validation set (blue: correct, red: incorrect, gray: `X`). This allows us to understand some of the behaviour of the model. We can see regular repeated errors in *Mr. Moonlight*, which are mostly mistaking two similar qualities. 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 *September*, 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.

4.5 Takeaways and Next Steps

Below, I summarise the main takeaways from this section and motivate further improvements to the model.

Performance on rare chord classes is poor. There are few instances of chord classes with complex qualities and upper extensions. The model ends up predicting major and minor classes for these rare chords. 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 5.2.1. I also look at a 'structured' loss function

which exploits similarity between chords in Section 5.2.2. Performance can also be improved by improving the data. I explore the use of data augmentation in Section 5.4 and synthetic data generation in Section 5.5.

Predictions are not smooth. While it is unclear whether or not smoothness will improve performance, a good chord recognition model's predictions would be smooth. This motivates the exploration of a 'decoding' step in Section 5.1.

The model does not use long-range context. *CNN1* only shares context a maximum of 5 frames either side as this is its kernel size. It achieves an accuracy of 54.5%, just 5% less than the *CRNN*. This suggests that the majority of the performance gain associated with including contextual information is not complex nor far-reaching. I conclude that while a little context improves performance, the *CRNN* does not use context in a complex manner.

The model is simple. The analysis of feature maps by Korzeniowski and Widmer [46] corroborate up with this idea. Their analysis suggests that the model is detecting the presence of individual notes and deciding which chord is present based on these notes. This is why many more parameters do not help. Unfortunately, this results in many similar chords being confused. The root note can be wrong, similar qualities are often mistaken and predictions often miss upper extensions. This explain the large discrepancy between the accuracy and `mirex` metric with average values of 60% and 79% over the validation set respectively. This problem is exacerbated by the imbalance in the dataset discouraging the model from being sensitive to indicators of rare chord classes.

Performance is song-dependent. Accuracies over songs vary widely. `mirex` scores are more consistent but still vary. Detailed analysis of properties of songs which cause the model would be valuable work but I do not explore this further here.

Chords are not interpretable in time. The model struggles a little on transition frames. Solely improving performance on such frames is unlikely to improve metrics by much. A much better reason to segment chords is to give the output of the model a far interpretable meaning. The frame-wise correctness illustrated in Figure 4.3 are not musically interpretable. Even if chord symbols were added, this would not constitute good musical notation. Musicians do not operate over 93ms frames. They think of music as existing in beat-space. Poltronieri et al. [57] explore the related task of finding chord boundaries in audio given the chord sequence. Instead, I take inspiration from Chris Donahue and Liang [20] and use a beat detection model in order to task the model with predicting chord over beats rather than frames in Section 5.6.

Chapter 5

Improving the Model

In this chapter, I use the insights from Chapter 4 to improve the *CRNN* and address questions raised by the literature. I perform a series of experiments to test improvements to the model, evaluate a selection of models on the test set and perform qualitative analysis of the model's outputs.

Many of these experiments introduce new hyperparameters. I choose these hyperparameters in a greedy fashion and keep them as specified unless stated otherwise. While the assumption of independence between hyperparameters is certainly wrong, it is computationally infeasible to perform a full hyperparameter search.

I first conduct experiments verifying that CQTs are the best features for ACR and that a hop length of 4096 is appropriate. These experiments are left to Appendix A.5 for brevity. To summarise, CQTs achieve 10% greater accuracy than other spectrogram variants and any hop length less than 4096 achieves similar results. Thus, I proceed with CQTs and a hop length of 4096.

5.1 Decoding

As observed in 4.4.3, the *CRNN* predicts 168 transitions per song as opposed to the 104 seen in the ground truth data. I implement a decoding step over the frame-wise probability vectors to smooth predicted labels. Common choices for decoding models include a conditional random field (CRF) [32, 43] and a hidden Markov model (HMM) [21].

I first implement a HMM. The HMM treats the frame-wise probabilities as emission probabilities and the chord labels as hidden states. O'Hanlon and Sandler [58] note that using a transition matrix with homogeneous off-diagonal entries in the transition matrix performs similarly to using a learned transition matrix. I adopt such a transition matrix for this HMM, with a parameter β denoting the probability of self-transition and all other transition probabilities equal to $\frac{1-\beta}{C-1}$. Decoding then follows the Viterbi algorithm [59] over the summed forward and backward pass.

A plot of the effect of β 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 the performance of the model while successfully reducing the number of transitions per song to that of the true labels. I choose $\beta = 0.15$ for the remainder of experiments as it results in 102 transitions per song while maintaining high performance.

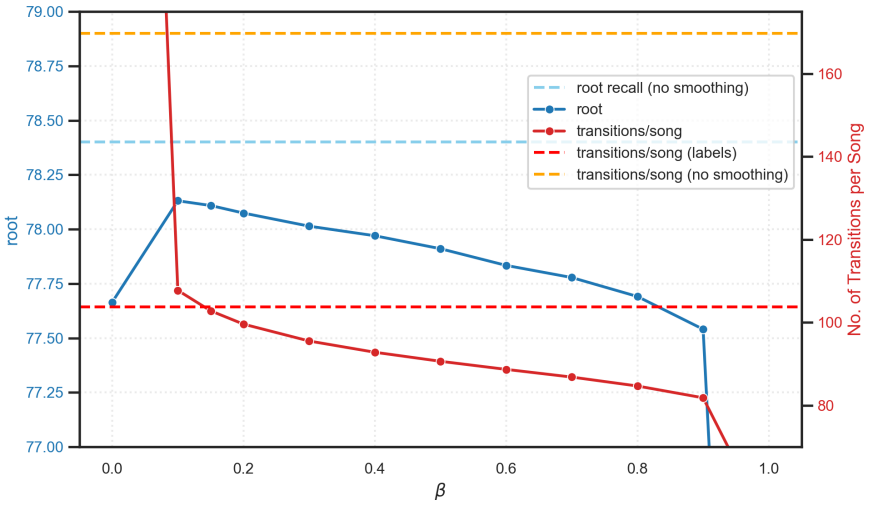

Figure 5.1: Effect of the HMM smoothing parameter β on the *CRNN* model. As we increase β, the number of transitions per song decreases. I choose $\beta = 0.15$ as it results in 102 transitions per song, very close to the 104 of the ground truth. Performance is stable across β with a slight degradation 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.12. 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. This is illustrated in Appendix A.15.

I also implement a linear chain CRF using `pytorch-crf`.[1] In contrast to the HMM, the CRF uses a learned transition matrix. Results comparing the HMM, CRF and no smoothing can be found in Table 5.1. Both the CRF and HMM reduce the number of transitions per song to a similar level. The HMM outperforms the CRF, with 3.5% greater accuracy. The HMM has almost identical performance to the model with no smoothing. I hypothesise that the learned transition matrix allows the model to overfit to the chord sequences in the training set. Regardless of the explanation, I proceed with HMM smoothing.

5.2 The Loss Function

5.2.1 Weighted Loss

One of the biggest problems with the *CRNN* is the low recall on rarer chord qualities. Two common methods for dealing with long-tailed distributions are weighting the

[1] `https://github.com/kmkurn/pytorch-crf`

smoother	acc	root	third	seventh	mirex	transitions/song
none	**60.0**	**78.1**	**75.0**	**62.3**	**79.2**	167
HMM	**60.0**	**78.1**	**75.0**	**62.3**	**79.2**	**102**
CRF	56.5	75.5	72.5	58.6	76.2	100

Table 5.1: Results for the HMM and CRF smoothing methods with the *CRNN*. The HMM has almost identical performance to the model with no smoothing. However, it drastically reduces the number of transition per song to an acceptable level. The CRF performs notably worse than the HMM.

loss function and over-sampling. Rowe and Tzanetakis [33] also explore the use of curriculum learning as form of re-sampling which we do not explore here because they report only minor performance gains. Sampling is explored by Miller et al. [21] 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 to explore but is left as future work. It would require a complex sampling scheme as frames cannot be sampled independently.

Weighting has been explored by Jiang et al. [32]. We employ a similar but simpler implementation here. A standard method of weighting is to multiply the loss function by the inverse of frequency of each, with a parameter controlling the strength of the weighting. This is defined in Equation 5.1.

$$w_c = \frac{1}{(\text{count}(c) + 10)^{\alpha}} \tag{5.1}$$

Where w_c is the weight for chord c, 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. I add 10 in the denominator to avoid dividing by 0 and to diminish the dominating effect of chords with very few occurrences. I then define normalised weights w_c^* in Equation 5.2 so that the learning rate can remain the same.

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

Where $\mathcal{C}$ is the set of all chords in the vocabulary. This keeps the expected weight over samples at 1 such that the effective learning rate remains the same. These values are calculated over the training set. I test values of α in the set $\{0, 0.05, 0.1, \ldots, 0.95, 1\}$. The plot in Figure 5.2 illustrates the effect of the weighting on the model's performance. I find that increasing α improves $\texttt{acc}_{\text{class}}$ but decreases `root` accuracy. Choosing $\alpha = 0.3$ maximises $\texttt{acc}_{\text{class}}$ without hurting root accuracy which I carry forward to subsequent experiments.

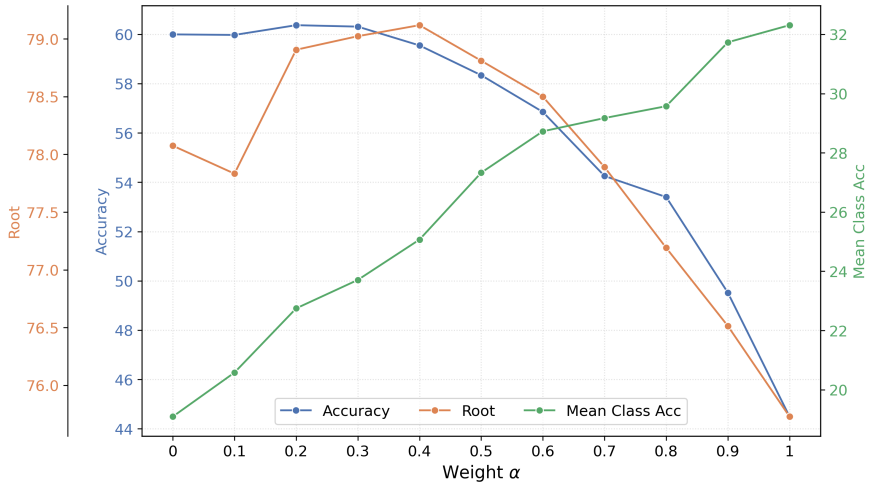

Figure 5.2: Effect of weighted loss on the *CRNN* model with varying α. As we increase α, $\texttt{acc}_{\text{class}}$ improves accuracy and `root` decreases. I claim there is a sweet-spot where very little overall performance is sacrificed for better class-wise accuracies. I choose this to be $\alpha = 0.3$. The `root` and `third` metrics improve and less than 3% is lost on other metrics while mean class-wise accuracy improves by 6%.

For further insight, a plot of the differences between a confusion matrices with and without weighted loss can be found in Appendix A.16. Notably, recall on most qualities increases, with recall on major7 doubling to 0.34. The weighted model predicts 2.2 times fewer `X` symbols which may explain how it increases recall on these rarer qualities without sacrificing on accuracy.

Weighting the loss function also increased the number of transitions predicted per song slightly. This may be because occasional sharp gradient updates cause more extreme probability outputs. I increase the HMM smoothing parameter β to 0.2 to bring the number of transition per song to 104.

5.2.2 Structured Loss

McFee and Bello [16] propose a structured loss function which they claim improves performance on the *CRNN* model. They introduced additional targets of the root, bass and pitch classes. I follow a similar method but do not include the bass as the current chord vocabulary does not consider inversions. The idea behind this loss term is to explicitly task the model with identifying the components of a chord that we care about. This can allow the model to exploit structure in the chord vocabulary such as shared roots and pitch classes, rather than all symbols being predicted independently.

The root can be any of the 12 notes in the Western chromatic scale, `N` or `X`, creating a 14-dimensional classification problem. The 12 pitch classes each represent a single binary classification problem. Two fully connected layers calculate a 14-dimensional vector and 12-dimensional vector from the hidden representation outputted from the GRU for the root and pitch classes respectively. Finally, these representations are concatenated with the original output of the model and fed to the final fully connected

layer to predict the chord symbol.

The mean cross-entropy loss is calculated in each case. These are then summed to from the *structured loss*. Finally, a linear combination of the structured loss and the original loss is calculated. The final loss is a convex combination of the original loss and the structured loss as defined in Equation 5.3.

$$L = \gamma L_{chord} + (1-\gamma)(L_{root} + L_{pitch}) \tag{5.3}$$

Where L is the overall loss, L_{chord} is the cross-entropy loss over chords symbols, L_{root} is the cross-entropy loss targeting the root, L_{pitch} is mean binary cross-entropy over each of the pitch classes. γ is a hyperparameter controlling the weighting of the original loss.

I test models with $\gamma \in \{0, 0.1 \dots 0.9\}$. Choosing $\gamma = 0.7$ improves accuracy by 1.3% while `mirex` worsens by 0.3%. Accuracy with `third` increases by 1.7% and on `seventh` by 1.3%. Generally, greater γ improves accuracy metrics while `mirex` results are noisy. A plot of the trend can be found in Appendix A.10 but does not provide further insight. I keep $\gamma = 0.7$ going forward based on peak accuracy.

5.3 Generative Features

Chris Donahue and Liang [20] use generative features extracted from Jukebox [8] to improve performance for melody transcription. They also produce a chord transcription model using the same methodology but do not report results. I decide to test generative features using MusicGen [9] as a feature extractor. This was for several reasons. MusicGen is a newer model. It has several different sizes of model which could be tested against each other as an experiment on the complexity of the model. It has a fine-tuned variant called MusiConGen [38] which is used for synthetic data generation in Section 5.5. All model weights are available on the HuggingFace Hub.[2] Finally, its training data was properly licensed, unlike Jukebox. I leave the results of Jukebox for future work. Details of the feature extraction process are left to Appendix A.18 for the interested reader.

As the representations are 2048-dimensional, it is computationally infeasible to feed this directly into the GRU. Instead, I project these vectors down to a power of 2, from 16 to 1024, using fully connected layers. The best representation is with 64 dimensions although results show no clear trend. Results are left to Appendix A.20.

I test different variants of MusicGen including musicgen-large (3.3B parameters), musicgen-small (330M), musicgen-melody (1.5B) and MusiConGen (1.5B). I also test different reductions of the four 'codebooks' outputted by the language model including taking each codebook on its own, taking the element-wise mean across codebooks or the concatenation of all four. For further details on codebooks, see the work of Copet et al. [9]. The best model was found to be musicgen-large and the best reduction to average over the four codebooks. However, results were all close and are left to Appendices A.21 and A.22 for lack of importance.

[2] `https://huggingface.co/docs/hub/en/index`

features	accuracy	root	third	seventh	mirex
gen·CQT	**61.0**	**80.3**	**77.0**	**63.3**	78.4
gen	58.7	77.6	74.3	60.9	77.5
CQT	**61.0**	79.8	76.8	63.2	**79.3**

Table 5.2: Comparison of CQT and generative features with a concatenation of the two, denoted as gen·CQT. The concatenation performs the best on most metrics, but not by enough to claim that it is meaningfully better.

It is surprising that the concatenated representation performs worse than the averaged representation as it contains at least as much information. However, if the information provided by each of the codebooks is largely the same then there are no reasons that the concatenated representation should perform better and training with Adam may simply find a worse minima. Training on 8192-dimensional vectors is also computationally expensive.

To test whether or not these features help when compared with a CQT, I test with the CQT only, generative features only and a concatenation of the two. The results are shown in Table 5.2. Although the generative features perform worse than the CQT, they clearly contain information useful for chord recognition with an accuracy of 58.7%. When the CQT and generative features are used together, performance remains largely the same. This experiment was run multiple times, with similar results each time. There is no clear evidence that the generative features provide any benefit over just using the CQT.

This is surprising as Chris Donahue and Liang [20] claim that generative features are better than hand-crafted features for the related task of melody recognition. However, they only compare to mel-spectrograms which may perform as well as CQTs for melody recognition. Observations here cast doubt on the validity of their claims. It may be the case that features extracted from other generative models such as Jukebox [8] or MusicLM [60] perform better. The comparison is left for future work.

Given the lack of improvement and the drastically increased computational cost associated with extracting features and training the model, I do not proceed with using generative features.

5.4 Pitch Augmentation

Pitch augmentation has been done in other works on chord recognition, either on the CQT [32] or on the audio [43, 16]. Although similar, these are not identical transformations. Shifting the CQT takes place after discarding phase information and leaves empty bins behind whereas audio pitch shifting can introduce other artefacts intended to preserve harmonic structure and maintain phase information. I implement both methods and compare.

When a sample is drawn from the training set, it is shifted with probability p. The shift is measured in semitones in the set $\{-5,-4\ldots-1,1,\ldots 6\}$ with equal probability of

each shift. This means there is 12 times as much training data. Convergence was still reached in 150 epochs. Shifting the CQT matrix is done by moving all items up or down by the numbers of bins corresponding to the number of semitones in the shift. The bins left behind filled with a value of -80dB. Audio shifting is done with `pyrubberband`.[3]. CQTs are then calculated on the shifted audio. A plot of the effect of the shift probability p on the model's performance can be found in Figure 5.3.

Results show a clear trend that increasing p improves performance. Shifting the audio provides a very similar effect to simply shifting the CQT. Choosing $p = 0.9$ results in a 2.1% increase in accuracy. The `mirex` score breaks the trend with performance varying over different values for p. $\texttt{acc}_{\text{class}}$ also improves by over 2%. This can be explained by the model becoming becoming root-invariant. With $p = 0.9$, all potential roots become close to equally likely. I proceed with pitching shifting with $p = 0.9$ on the CQT for the remainder of the experiments as it is computationally cheaper than shifting the audio.

McFee and Bello [16] claim an increase of 5% on the median across most metrics. I do not find such a large effect here. Nonetheless, it is clear that pitch shifting is a useful augmentation.

Note that the weights for the weighted loss are calculated based on *expected* counts, taking the shift probability p into account. I also test shifting on both the CQT and audio but results are not any different than either method alone. Unfortunately, it was not computationally feasible to test pitch shifting with generative features as the feature extraction over $12 * 1210 = 14,520$ songs is too expensive.

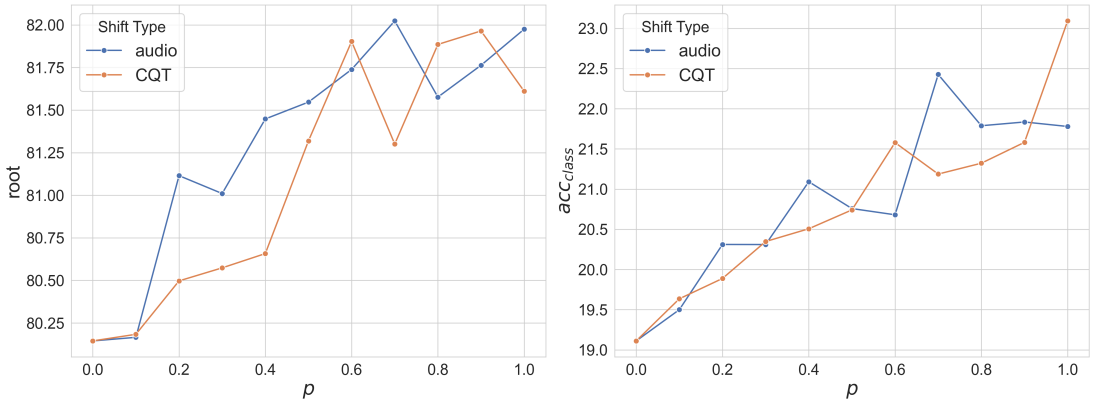

Figure 5.3: Effect of pitch shift probability p on `root` (left) and $\text{acc}_{\text{class}}$ (right). Pitch shifting provides a clear increase in performance. Shifting the audio or the CQT has a similar effect. I choose $p = 0.9$ which is close to making the model root-invariant.

5.5 Synthetic Data Generation

Given the success of pitch augmentation for ACR, it is sensible to look for other sources of data. Further augmentation is possible such as adding noise and time-stretching. However, these do not provide new harmonic structure or create new instances of

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

rare chords so I do not explore these here. Instead, I look to generate new data. Generation would be possible through automatic arrangement, production and synthesis. However, this is a complex task, requires a lot of human input and is unlikely to produce sufficient variety of timbres, instrumentation arrangement. Instead, I use a recent chord-conditioned generative model called MusiConGen [38]. I use this over CocoMulla [61] as the method of chord-conditioning has a simpler interface, doesn't require reference audio and the authors claim that it adheres more closely to its conditions. Indeed they feed the outputted audio through BTC [43] and find a `triads` score of 71% using the chord conditions as the ground truth. While far from perfect, it suggests the model is able to generate audio that mostly lines up with the chord conditions.

I generate 1210 songs, each 30 seconds long, to mimic the size of the *pop* dataset. I refer to this dataset as *synth*. This is split into a train, validation and test split in the same fashion as for the *pop* dataset. To generate a larger order of magnitude of songs would require a lot of compute. While the model does support auto-regressive generation for longer audio, I find that its outputs become incoherent using the provided generation functions. It also sometimes produces incoherent output with 30 seconds but this was much less common.

To generate a song, I sample a BPM from a normal distribution with mean 117 and standard deviation 27, clipped to lie in the range $[60, 220]$. These values were calculated from the training set. I then sample a song description from a set of 20 generated by ChatGPT. The descriptions outline a genre, mood and instrumentation. Descriptions include only jazz, funk, pop and rock which were all part of the fine-tuning training set for MusiConGen. Note that the model does not output melodic vocals, owing to the lack of vocal music in the pre-training and fine-tuning data. Finally, I generate a jazz chord progression using the theory of functional harmony Steedman [62]. Details of this process can be found in Appendix A.23. The process generates a very different chord distribution than the one in *pop* with many more instances of upper extensions and rare qualities. This is intended to provide the model with many more instances of rare chords.

To offset this distribution shift, I calibrate the probabilities outputted by the model. To encourage root-invariance, calibration terms are averaged over roots for each quality. The details of calibration are described in Appendix A.24 with a figure showing the calibration terms for each quality. This figure shows that the rarer qualities are much more common in the synthetic data.

Outputs from MusiConGen were manually inspected. In general, the outputs are good. They consistently stick to the provided BPM and usually stick to the chord conditions. Outputs are occasionally musically strange with jarring drum transients and unrealistic chord transitions. They also did not have an enormous variety in terms of timbre and instrumentation. Nonetheless, the majority of examples have sensible annotations that one would expect a human to be able to annotate well.

I compare a model trained on only *pop*, trained on only synthetic data and trained on both. While the latter results in more training data per epoch, convergence is always reached so these are fair comparisons. I test the models on both the *pop* and synthetic data validation splits. Given the increased instances of rare chords in the synthetic data,

I remove the weighting on the loss function for the models trained on synthetic data.

The results are shown in Table 5.3. The model trained on both datasets performs very similarly to the model only trained on *pop*. That is, the model has not overfitted to the synthetic data. However, it has also not improved performance. Training on any synthetic data drastically improves the accuracy on the *synth* validation split. This is likely due to the unique chord distribution of the generated jazz progressions and the consistent instrumentation. The model trained on both datasets performs the best, with an accuracy of 51.0%.

training set	*pop* acc	*synth* acc
pop	**62.1**	24.2
synth	48.6	44.8
both	**62.1**	**51.0**

Table 5.3: Results for models trained on the Pop dataset, synthetic data and both together. Unsurprisingly models trained on *pop* perform better on *pop* and models trained on *synth* perform better on *synth*. The model trained on both does not to any better on the *pop* validation split, but performance is increased on the *synth* validation split. Training only on *pop* results in an accuracy of just 24.2% on *synth*, much lower than reported by Lan et al. [38]. However, this may be due to the unrealistic distribution of chords in the generated sequences.

For further insight, the difference in confusion matrix over qualities is plotted in Appendix A.25. It does not show any clear trends. Recall on some rare qualities increases while it decreases for others. In general, the model predicts `maj` less often for rare chord qualities but replaces this with other erroneous predictions.

While performance does not improve, the lack of overfitting and the improvement on the synthetic data provides hope that with further work and improved generative models, synthetic data could prove a useful tool for ACR. Given the lack of performance increase, I do not continue to train on the synthetic data.

5.6 Beat Synchronisation

Chords exist in time. Musicians interpret chords in songs as lasting for a certain number of beats, not a fixed length of time. In its current form, the model outputs frame-wise predictions. While these could be stitched together to produce a predicted chord progression or beat-wise predictions could be made as a post-processing step, I decide to implement a model that outputs beat-wise predictions directly. This allows the model to use information from the entire duration of the beat to make its prediction.

Following the methodology of Chris Donahue and Liang [20], I first detect beats using `madmom` [63]. This returns a list of time steps where beats have been detected. I first verify that these beats are plausible. I perform a cross-correlation analysis with the chord transitions in a similar manner to Section 3.1.1. A histogram of maximum lags within a window of 0.3 seconds can be found in Appendix A.26. I find that almost all

maximum lags occur within a window of 0.1 seconds. To provide further evidence, I compute the maximum accuracy a model could attain if predicting chords at the beat level. This is done by iterating over each beat interval and assigning the chord with maximum overlap with the ground truth. This yields an accuracy of 97.1%. With these observations combined, I am satisfied that the estimated beats are accurate enough to be used.

To calculate features for a beat interval, I average all CQT features whose centre is contained within the interval. The CQT is calculated using a hop length of 1024. The shorter hop length is used to minimise the effect of CQT frames with partial overlap with two beat intervals and ensures that each beat has many CQT frames associated with it. These representations have the added benefit of decreased computational cost as beats have a lower frequency than frames.

I test the model with different divisions and groupings of beats. This is to test the assumption that the whole beats are fine-grained enough for the model to make good predictions. I include tests where beat intervals are sub-divided in two, in four, or beat intervals are joined into groups of two. I refer to this as the *beat division*. I also test a *perfect* beat division where the true chord transitions are taken as the beats. This is not a fair comparison as the model should not have access to true transition timings. However, it does provide an idea of how the model would fare if it could perfectly predict chord transitions. Note that HMM smoothing is removed for beat-wise models as it is no longer necessary.

Results are shown in Table 5.4. Using beat-wise predictions does not affect performance compared to frame-wise predictions. The model also performs just as well with a beat division of 1 as with a beat division of $1/2$ or $1/4$. The model performs worse with a beat division of 2 though only loses 6% accuracy. This suggests that most chord transitions occur at frequencies smaller than the beats produced by `madmom` but that sometimes the two are misaligned.

The model with 'perfect' beat intervals performs slightly worse on accuracy metrics but attains a very high `mirex` score of 90.4% which is the highest of any seen in the literature. This is a very promising result. Data from the CQT producing a `mirex` score of 90.4% suggests provides hope for significant improvements in the field. Why the model performs worse on accuracy metrics is not clear. It may be because averaged features from longer time periods provide better information as to the pitch classes present but dampens signal regarding the root note. Indeed, the mean chord duration is 1.68 seconds while a CQT frame is 0.093 seconds. Why this effect is not observed when predicting at the beat-level is also not clear. Further analysis may lead to significant improvements on ACR.

beat division	acc	root	third	seventh	mirex
1/4	62.3	81.3	78.3	64.5	79.6
1/2	**62.8**	81.5	**78.6**	**65.1**	79.9
1	62.3	81.3	78.1	64.6	80.0
2	56.4	74.7	71.6	58.5	73.4
none	62.5	**81.7**	78.2	64.7	80.2
perfect	61.1	79.6	76.1	63.4	**90.4**

Table 5.4: Results for different beat divisions. The beat division of 'none' refers to a frame-wise *CRNN* with a hop length of 4096 and HMM smoothing applied and 'perfect' refers to intervals that are calculated from the labels. Note that this 'perfect' model is not a fair test. Beat-wise predictions with beat intervals of 1 and below see no decrease in performance compared to frame-wise predictions. Beat intervals greater than 1 increasingly suffer from being forced to assign predictions to larger periods of time that may not line up with true chord transitions. Interestingly, a beat interval of 2 still performs relatively well. A notable result is the `mirex` score of the 'perfect' model. A score of 90.4% is the highest of any in the literature. Despite this, its accuracy does not improve.

5.7 Final Results

For final results, I retrain select models on the combined training and validation splits and test on the held out test split. This is an 80/20% train/test split. I consider the original *CRNN* with no improvements, the *CRNN* with a weighted and structured loss and HMM smoothing, concatenating generative features with CQTs, pitch augmentation, beat-wise predictions, 'perfect' beat-wise predictions and training on synthetic data.

Results are shown in Table 5.5. Observations are largely similar to those found previously. Weighted and structured loss with smoothing improve accuracy by 1.2% with pitch shifting improving by a further 1%. Generative features do not help and synthetic data improves performance by a further 0.6%. This alone is not a clear enough signal that training on synthetic data is better but provides hope for further work. Finally, beat-wise prediction maintain the same performance. The 'perfect' model achieves the highest performance across all metrics. The `mirex` of 90% found on the validation set has reduced to 88.7% and the gap with accuracy has narrowed when compared with previous results in Table 5.4. This is still a significant results and suggests that there is hope for breaking through the 'glass ceiling'. The mean class-wise accuracy does not improve past 19.7% without 'perfect' beats. The median in this case is 6%. The model's performance on the long tail remains poor. Using a greater α in weighting may improve the picture but would require sacrificing accuracy.

Another observation is that the model's accuracy did not improve on the test set with the additional training data. Accuracy of the *CRNN* with pitch shifts trained on only 60% of *pop* attained an accuracy of 64.3%, while the model trained on the full 80% training split attains 63.7%. This discrepancy can be explained by the stochastic nature of the training process but it still suggests that more data from the same distribution may not improve performance.

	acc	root	third	seventh	mirex	$\text{acc}_{\text{class}}$
CRNN	61.6	79.3	76.5	63.3	80.6	18.9
+ weighted/structured/HMM	62.8	80.9	78.3	64.6	80.6	18.6
+ gen features	62.7	80.4	77.9	64.6	**80.6**	19.5
+ pitch shift	63.8	**82.4**	79.4	65.6	80.3	**19.7**
+ synthetic data	**64.4**	82.2	**79.9**	**66.3**	**81.5**	18.3
+ beats	63.7	82.4	79.7	65.6	80.9	**19.7**
+ perfect beats	65.8	84.5	81.7	67.6	88.7	21.2

Table 5.5: Final results from various experimental setups on the test set. The 'perfect beats' model assumes oracle beat tracking and achieves the highest results across all metrics but is excluded when considering the best results for each metric as it is not a fair comparison. Beat-wise models do not use synthetic data. Observations echo those found previously on the validation set. Adding the weighted loss, structured loss term and an HMM smoother improves accuracy by 1.2%. Generative features do not help while pitch shifting improves metrics by 1% and synthetic data by 0.6%.

5.8 Qualitative Analysis

To conclude this section, I present two examples of the model's predictions from the test set using the '+ synthetic data' model. Finally, I compare beat-wise predictions with frame-wise predictions on songs not contained the dataset.

The first song is *Don't Stop Me Now* by Queen. With stable vocals and a clear piano part, the model fares well. The model's predictions are shown in Figure 5.4. Transitions are close to the ground truth timings. Almost all chords have the correct root and third while sevenths are often missed. The `root` and `mirex` on this song are both $\approx 87\%$ while the overall accuracy is only 56.2%.

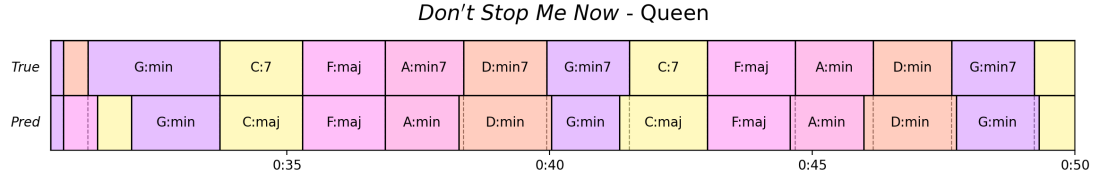

Figure 5.4: Comparison of predictions with ground truth on a section of *Don't Stop Me Now* by Queen. Predictions have the correct root and third and are well timed but have discrepancies in sevenths.

The second song is *Roxanne* by The Police, illustrated in Figure 5.5. Syncopation, ambiguous bass, and sliding vocals make this song harder to annotate. Root recall is almost 80% but `mirex` is only 64%. Sevenths are often omitted and thirds are sometimes wrong as well. The model confuses major and minor, and sus4 and major qualities. There are also some chord transitions which are not present in the ground truth and chord transitions are not all well-timed.

Overall, outputs are a lot smoother than found previously in Section 4.4.6 with more cohesive predictions of the same chord for a series of frames. However, the problem associated with identifying rarer chord qualities remains. Performance is also still highly song dependent. The model's accuracies over songs in the test set have a standard deviation of 20%.

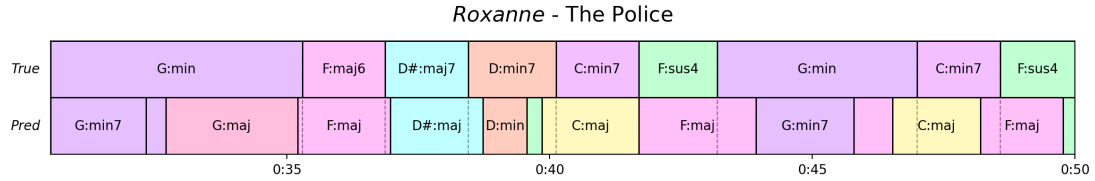

Figure 5.5: Comparison of predictions with ground truth on a section of *Roxanne* by The Police. Predictions have the correct root but are not always well timed and do not always have the correct quality. The model's annotation is not very good on this song.

As a final example, frame-wise and beat-wise predictions are compared on two songs not part of the dataset in Figure 5.6: *Someone Like You* by Adele and *Misty* by Ella Fitzgerald. In both cases, similar chord information is visualised differently. From a musicians' perspective, frame-wise predictions lead to ambiguity over how long a chord lasts in beats. Frame-wise block lengths hint at beats but are not uniform. With time changes, this would become more problematic. In regions with rapid chord transitions, chords are not musically interpretable. In contrast, beat-wise predictions resemble musical notation more closely, and longer beat intervals prevent rapid chord changes. However, when predictions occur part-way through bars, beat-wise output can still be hard to interpret.

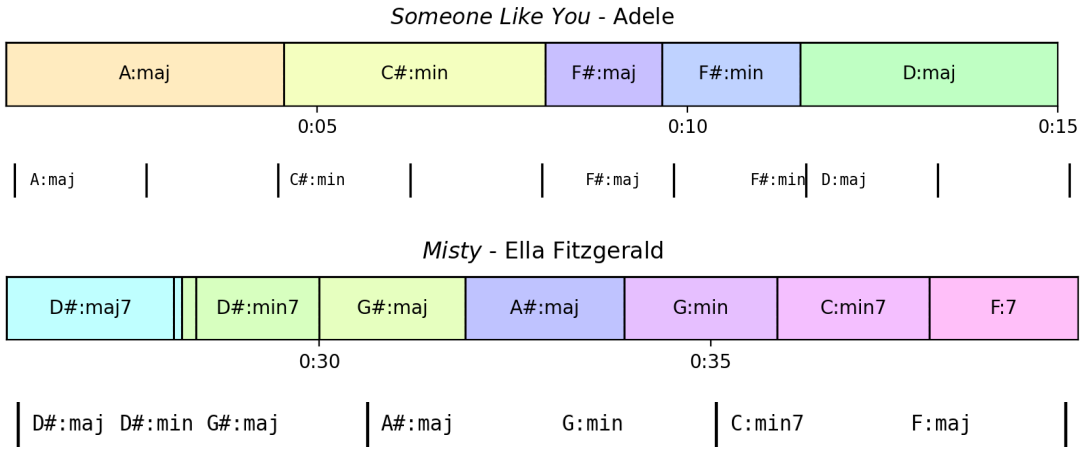

Figure 5.6: Comparison of frame-wise and beat-wise predictions on *Someone Like You* by Adele and *Misty* by Ella Fitzgerald. Beat-wise predictions are visualised over bars with a meter of 4. For both methods, adjacent predictions of the same chord are grouped together. While beat-wise predictions are more musically meaningful and easily parsed, they can be messy and hard to interpret when chord transitions are late or off the beat.

Chapter 6

Conclusions and Further Work

6.1 Conclusions

In this thesis, I have presented a thorough analysis of deep learning in automatic chord recognition. There are a few key takeaways.

ACR models are not complex. Good performance relative to state-of-the-art can be achieved with few parameters. It is likely that the task of determining which pitch classes are present from a CQT is a relatively simple operation for a neural network to learn. Performance does not increase with model size past a low threshold. Without smoothing, frame-wise predictions result in too many chord transitions. Of the smoothing methods tested, fixed transition penalties or transition matrices are preferred. Weighting the loss function allows control over performance on rare qualities, but requires sacrificing a little overall accuracy. Introducing structured representations of chords as additional targets provides a small performance gain. Features extracted from MusicGen contain information relevant to ACR but not any more than is already contained in the CQT.

Pitch augmentation works well to encourage root-invariance and improve accuracy. The use of synthetic data provides an exciting avenue for future research. Results presented here show signs that with newer models and more careful construction, synthetic data could provide many new training examples with a customisable chord distribution.

Predicting chords over beats instead of frames does affect performance while improving the interpretation of the model's outputs. Predicting chords over the true chord intervals results in the highest `mirex` score seen in the literature suggesting that there remains room for improvement in the way chord transitions are detected.

While deep learning models are powerful chord recognisers, there is still much work to be done before the problem is solved. The 'glass ceiling' is yet to be broken but the work presented here provides a solid foundation for future research and hope that the true ceiling is much higher.

6.2 Future Work

Many of the experiments conducted would benefit from further analysis. Implementing a sampling method which prioritises rare qualities may yield improved results over a weighted loss function. Looking at alternative methods of structuring chords beyond simply the pitch classes present may improve results, like the work of Jiang et al. [32]. The use of larger generative models trained on a wider variety of data may improve the richness of its representations.

Work presented here highlights also suggests new avenues of research.

Multiple Data Sources. Results on synthetic data show enough promise to continue. A more closely controlled chord sequence generation process may help. For example, constructing examples specifically to highlight the differences between different sevenths and observing the model's recall on sevenths. Other datasets also exist such as *HookTheory*. I was not able to obtain audio from this source. However, results here also suggest that gathering more data from the same distribution may not lead to improvements. A better source of data may be *JAAH* which would enable comparisons across genres and chord distributions.

Finding better chord transitions than beats. The high `mirex` score found in Section 5.6 suggests two things. First, targeting the problem of identifying chord transitions rather than beats may yield better results. Durán and de la Cuadra [48] jointly estimate beats and chords but to the best of my knowledge, no modern work has jointly estimated chord transitions and chord. Second, current models are missing information regarding presence of pitch classes that is present in the CQT. Perhaps this information is spread out in time or obscured by nearby frames that are irrelevant to the current chord. Understanding this effect may lead to new insights.

Subjective annotations. Inter-annotator agreement of the root of a chord is estimated at lying between 76% [64] and 94% [65] but these metrics are calculated using only four and two annotators respectively. Humphrey and Bello [44] and Harte and Sandler [66] posit that agreement between annotations can be far lower than that for some songs. Analysis of such an effect on commonly used datasets would provide a valuable contribution to the field. Such analysis could be used to inform the design of more subtle chord annotations which take into account multiple annotations and degrees of uncertainty.

A statement on the limitations of the conclusions presented and ethics of musical machine learning models can be found in Appendix A.1.

Bibliography

[1] 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.

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

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

[4] 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.

[5] 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.

[6] 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.

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

[8] 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`.

[9] 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`.

[10] Alexandre Défossez, Jade Copet, Gabriel Synnaeve, and Yossi Adi. High fidelity neural audio compression, 2022.

[11] Avery Wang. An industrial strength audio search algorithm. 01 2003.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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, 2016.

[18] 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.

[19] 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.

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

[21] 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.

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

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

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

[30] 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.

[31] Vsevolod Eremenko, Emir Demirel, Baris Bozkurt, and Xavier Serra. Audio-aligned jazz harmony dataset for automatic chord transcription and corpus-based research. In *Proceedings of the 19th International Society for Music Information Retrieval Conference (ISMIR)*, pages 488–495, Paris, France, 2018. doi: 10.5281/zenodo.1290736. URL https://ismir2018.ircam.fr/doc/pdfs/206_Paper.pdf.

[32] 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.

[33] 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.

[34] 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.

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

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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, 2016. doi: 10.1109/MLSP.2016.7738895.

[47] 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.

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] 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/.

[56] 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.

[57] Andrea Poltronieri, Valentina Presutti, and Martín Rocamora. Chordsync: Conformer-based alignment of chord annotations to music audio. In *Proceedings of the Sound and Music Computing Conference (SMC)*, 2024. URL https://arxiv.org/abs/2408.00674.

[58] 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.

[59] Andrew J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2): 260–269, 1967.

[60] Andrea Agostinelli, Timo I. Denk, Zalán Borsos, Jesse Engel, Mauro Verzetti, Antoine Caillon, Qingqing Huang, Aren Jansen, Adam Roberts, Marco Tagliasacchi, Matt Sharifi, Neil Zeghidour, and Christian Frank. Musiclm: Generating music from text. *arXiv preprint arXiv:2301.11325*, 2023. URL https://arxiv.org/abs/2301.11325.

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

[62] Mark Steedman. A generative grammar for jazz chord sequences. *Music Perception: An Interdisciplinary Journal*, 2(1):52–77, 1984. doi: 10.2307/40285276.

[63] Sebastian Böck, Filip Korzeniowski, Jan Schlüter, Florian Krebs, and Gerhard Widmer. madmom: a new Python Audio and Music Signal Processing Library. In *Proceedings of the 24th ACM International Conference on Multimedia*, pages 1174–1178, Amsterdam, The Netherlands, 10 2016. doi: 10.1145/2964284.2973795.

[64] 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.

[65] 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.

[66] 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.

[67] 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.

Appendix A

Appendix

A.1 Limitations and Ethics Statement

It is worth bearing in mind the limitations of the work presented here and its conclusions. The dataset used focuses largely on pop and rock. Cross-genre generalisation is not considered. These models expect music in standard tuning in the Wester chromatic scale. Chords themselves are also a highly Western concept. Not all music can be well described by the harmony structures considered in the chord vocabulary in this work.

There are also ethical issues relevant to training musical machine learning models worth mentioning. The audio data in this work is subject to copyright. Research on chord recognition falls under fair use but distribution of such data should be carefully controlled.

The generative models also present an ethical dilemma. Such models can be trained on copyrighted data without proper legal agreements with rights holder. The authors of the MusicGen models used in this work claim to have addressed such concerns. I do not support the use of work in chord recognition as a basis for advancing music generation models without proper legal and ethical issues being addressed.

A.2 Weighted Chord Symbol Recall Definitions

A formal definition of WCSR is provided in Equation A.1. The WCSR is a measure of the percentage of time that the model's predictions are correct.

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

$$Z = \sum_{i=1}^{N} \int_{t=0}^{T_i} \mathbb{I}_M(y_{i,t}) dt \tag{A.2}$$

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.

I also define WCSR for a single class c in Equation A.3. This is useful for understanding the performance of the model on a specific chord class.

$$\text{WCSR}(c) = \frac{1}{Z_c} \sum_{i=1}^{N} \int_{t=0}^{T_i} M(y_{i,t}, \hat{y}_{i,t}) \cdot \mathbb{I}_c(y_{i,t}) dt \tag{A.3}$$

$$Z_c = \sum_{i=1}^{N} \int_{t=0}^{T_i} \mathbb{I}_M(y_{i,t}) \cdot \mathbb{I}_c(y_{i,t}) dt \tag{A.4}$$

where N, T, M, $y_{i,t}$, $\hat{y}_{i,t}$ and $\mathbb{I}_M(y_{i,t})$ are defined as before in Equation A.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 A.1.

A.3 Cross Correlation for Alignment Verification

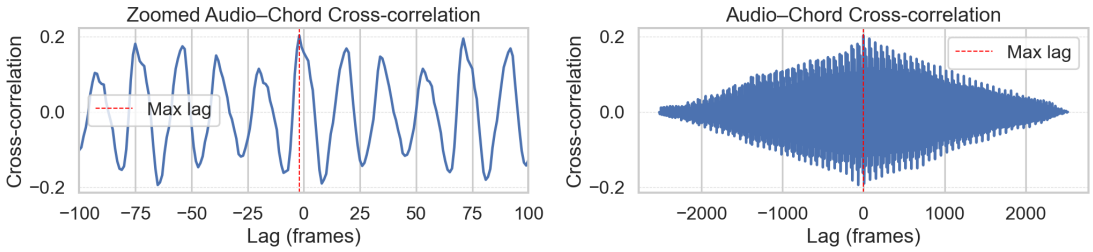

Figure A.1: Cross-correlation of the derivative of the CQT of the audio and the chord annotations for a single song. We can see correlation peaking in regular intervals of around 20 frames. 1 frame is 93ms so 20 frames ≈ 1.86 seconds. Zooming out, we observe peaks in correlation centred around 0.

A.4 Learning Rate and Scheduler Experiment Results

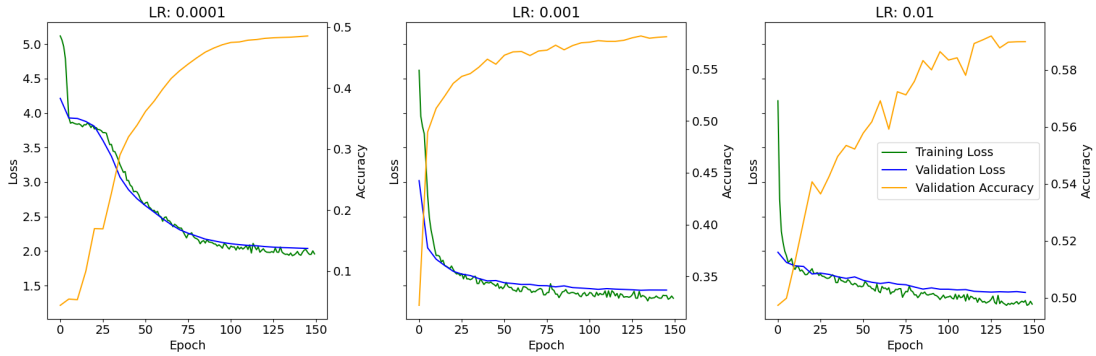

Figure A.2: Training graphs for the *CRNN* with different learning rates and the `cosine` scheduler. The learning rate of 0.001 seems to be the best, as it converges in a reasonable time and the validation accuracy increases in a stable fashion. While it may seem that running for more epochs may increase 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 A.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.

A.5 Revisiting the Spectrogram

A.5.1 Spectrogram Variants

It is standard practice in ACR to use a CQT as input. However, Pauwels et al. [13] raise the question of whether the CQT is truly the best choice. They suggest that the pitch-folding of the CQT may distort the harmonic structure of notes. By contrast, Chris Donahue and Liang [20] use a mel-spectrogram in place of a CQT.

I test four spectrogram variants in Table A.2. These include the standard CQT, mel-spectrogram and linear spectrogram. I also calculate a chroma-CQT to test whether the model is using information from multiple octaves better than a hand-crafted algorithm. The chroma-CQT is calculated by summing CQT values across octaves. Spectrogram calculations are all implemented in `librosa` [23]. I use 216 bins for the CQT and mel

spectrograms and 2048 fast Fourier transform (FFT) bins for the linear spectrogram with a hop length of 4096 for all.

Results show that CQTs are the best choice. This raises questions as to the validity of the conclusions drawn by Chris Donahue and Liang [20]. They claim that their generative features are better than hand-crafted features. However, they only compare to mel-spectrograms which may not perform as well as CQTs for the related task of melody recognition. The CQT is also better the chroma-CQT. We can be confident that the model is using information from multiple octaves more efficiently than the simply summing across octaves.

spectrogram	acc	root	third	seventh	mirex
CQT	**60.2**	**78.4**	**75.3**	**62.5**	**79.5**
chroma-CQT	50.1	71.4	65.7	52.0	69.8
mel	52.7	69.1	66.3	54.6	70.6
linear	51.2	66.1	63.0	53.1	73.8

Table A.2: Results for CQT, chroma-CQT, mel and linear spectrograms. The CQT is certainly the best feature. The other all perform similarly on accuracy and mirex, but the chroma-CQT does comparatively at identifying thirds and sevenths.

A.5.2 Hop Lengths

Different hop lengths have been used to calculate the CQT ranging from 512 [32] up to 4096 [16]. In previous experiments I have used a hop length of 4096 as is used by the authors of *CRNN* [16]. Shorter frames would reduce the number of transition frames but require more computational cost. If frame lengths are too short, the Fourier transform may not be able to capture the harmonic structure of the audio.

In Table A.3, I test the effect of different hop lengths on the model's performance. I use a CQT with 216 bins and a hop length of 512, 1024, 2048, 4096, 8192 and 16384. Results indicate that performance is similar for hop lengths of 4096 and below. Performance suffers for greater hop lengths. While it could be argued that 2048 does better than 4096, this difference is small enough that it is not worth the increased computational cost. Models trained with a hop size of 2048 take at least twice as long to train and evaluate as those trained on a hop size of 4096.

A.6 Small vs Large Vocabulary

Some initial experiments were conducted over a smaller vocabulary with $C = 26$. This vocabulary includes a symbol for major and minor over each root and two special symbols, `N` and `X` for no chord and unknown chord respectively. This contrasts the much larger vocabulary with 14 chord qualities for each root which is used for the majority of the experiments. With this larger vocabulary, $C = 170$.

Table A.4 shows the results of the *CRNN* trained over the smaller and larger vocabulary evaluated on the small vocabulary. The predictions of the model and the reference

hop length	acc	root	third	seventh	mirex
512	60.1	78.3	**75.5**	62.4	**80.0**
1024	60.2	**78.7**	75.2	62.5	79.6
2048	**60.3**	78.5	75.2	**62.6**	79.6
4096	60.0	78.1	75.0	62.2	79.2
8192	57.9	76.2	72.9	60.1	79.3
16384	53.3	71.7	68.0	55.4	77.9

Table A.3: Results over different hop lengths for CQT calculation. Any hop length in the range 512 to 4096 has similar performance. For frames that are any longer, performance suffers. This is likely caused by the requirement for the model to assign a single chord class to each frame. The longer the frame, the greater potential there is for multiple chords to be playing during the frame.

labels are all mapped to the small vocabulary before being evaluated in the same was as described in Section 3.2. This test was to verify that the model trained on the larger vocabulary performs well competitively on the smaller vocabulary. If the model trained on the larger vocabulary performed poorly on the smaller vocabulary, it may be prudent to first try to improve performance on this smaller vocabulary. It may also be a sign that the larger vocabulary is too complex or that the more detailed annotations are inconsistent.

However, the table shows very similar performance between both models. This allows us to proceed with the larger vocabulary for the rest of the experiments. The larger vocabulary is also more consistent with the literature and allows for a model to produce far more interesting chord predictions than simply minor, major and root.

Vocab	C	acc	root
small	26	76.7	80.1
large	170	76.0	79.1

Table A.4: *CRNN* with a small and large vocabulary. Metrics show similar performance between the two. Training on the large vocabulary does not prevent the model from learning how to classify the smaller vocabulary. Thus, I proceed with the larger vocabulary.

Note that the `mir_eval` package also includes a `majmin` evaluation metric that compares chords over just the major and minor qualities. However, this is not quite the same as the test above due to subtleties in how `mir_eval` chooses whether or not a chord is major or minor. It ends up ignoring many chords that could be mapped to these qualities in the smaller vocabulary. Coincidentally, the *CRNN* with the default parameters attains a `majmin` accuracy of 76.0% over the larger vocabulary. This further confirms that we need not continue to test on the smaller vocabulary. The `majmin` metric is not used in the rest of the thesis as it is not as informative as the other metrics and the `third` metric is highly correlated with it.

A.7 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 [16]. 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 [67] 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 [16] but gives more meaningful labels to roughly one third of the chords previously mapped to `X`.

A.8 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.5: *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 the *CRNN*.

A.9 A Long Run with SGD

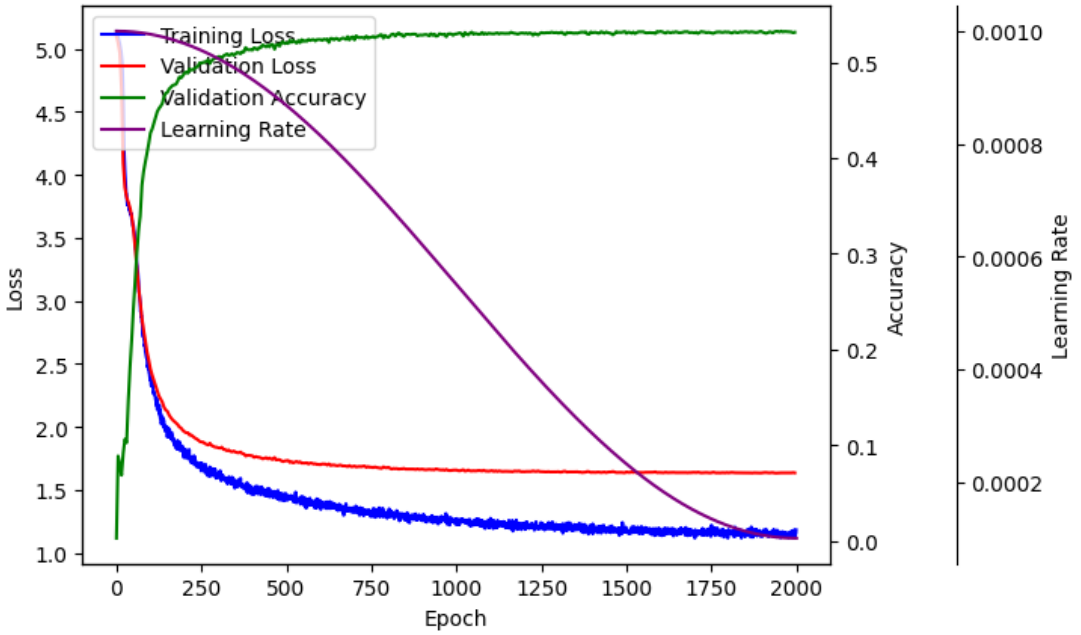

Figure A.3: Training graphs for the *CRNN* trained with SGD, momentum 0.9, a learning rate of 0.001 and the `cosine` scheduling for 2000 epochs. Convergence is reached but performance does not exceed that which is achieved by Adam over 150 epochs. Furthermore, there is signifcant computational cost associated with running for 2000 epochs. I proceed with Adam for the remainder of experiments.

A.10 Random Hyperparameter Search Sets

The random hyperparameter search for the *CRNN* was done over the following variables and values:

- `hidden_size` $\in \{32, 64, 128, 256, 512\}$
- `num_layers` $\in \{1, 2, 3\}$
- `segment_length` $\in \{5, 10, 15, 20, 25, 30, 35, 40, 45\}$
- `kernel_size` $\in \{5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}$
- `cnn_layers` $\in \{1,2,\ldots,5\}$
- `cnn_channels` $\in \{1,2,\ldots,5\}$

For each run, a value was selected for each hyperparameter, with each possible value equally likely.

A.11 Confusion Matrix of CRNN over Roots

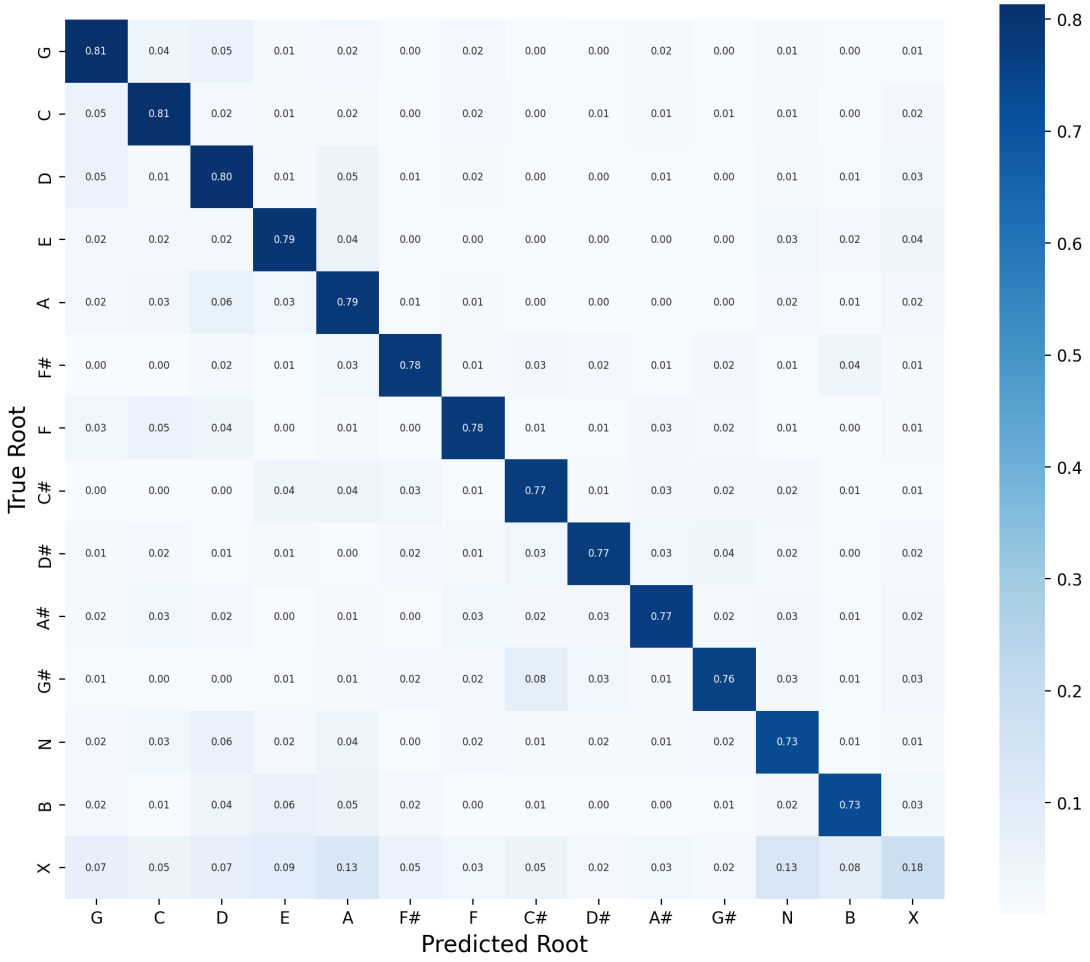

Figure A.4: Performance is relatively stable across roots. The only outlier is the unknown chord symbol X. This is to bexpected given the ambiguous nature of the chord.

A.12 Incorrect Region Lengths With/Without Smoothing

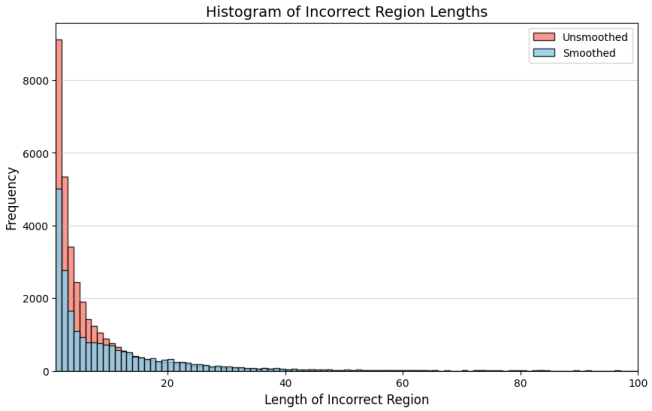

Figure A.5: Histogram over incorrect region lengths for the *CRNN* with and without smoothing. An incorrect region is defined as a sequence if incorrect frames with correct adjacent of either end. Both distributions have a long-tail, with 26.7% regions being of length 1 without smoothing. This raises concerns over the smoothness of outputs and requires some form of post-processing explored in Section 5.1. The distribution is more uniform with smoothing, with approximately half the very short incorrect regions.

A.13 Accuracy over the Context

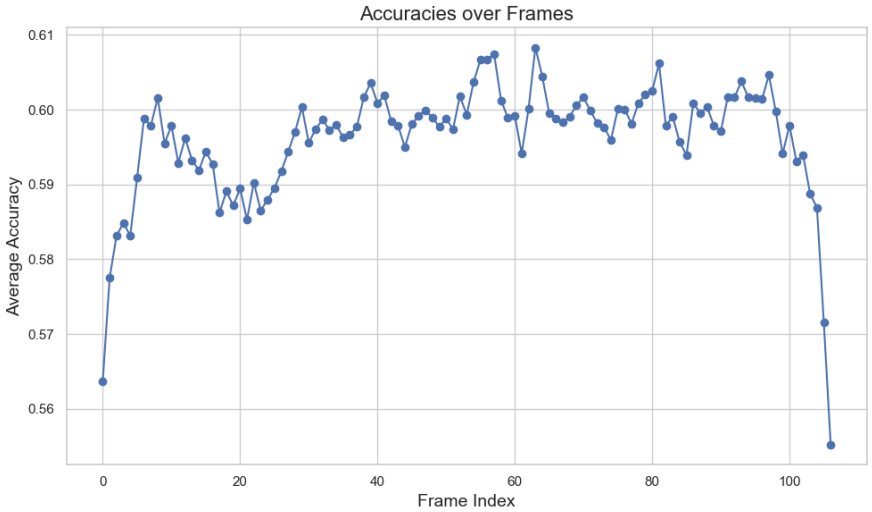

Figure A.6: Average frame-wise accuracy of the *CRNN* model over the patch of audio. The model performs worse at the beginning and end of the patch of audio, as expected. However, the differences are only 0.05. We propose that the context on one side is enough for the model to attain the vast majority of the performance attained with bi-directional context. This plot supports our procedure of evaluating over the entire song at once.

A.14 Accuracy vs Context Length of Evaluation

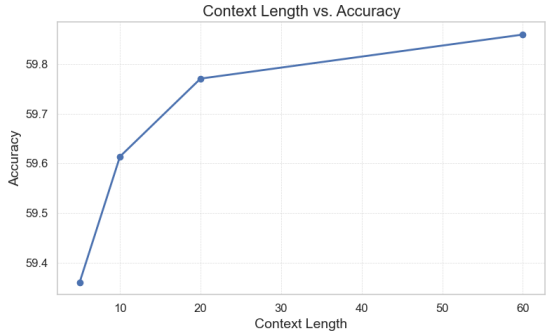

Figure A.7: Accuracy with increasing segment length of validation set. The accuracy increases very slightly. I choose to continue evaluating over the entire song at once.

A.15 HMM Smoothing Effect

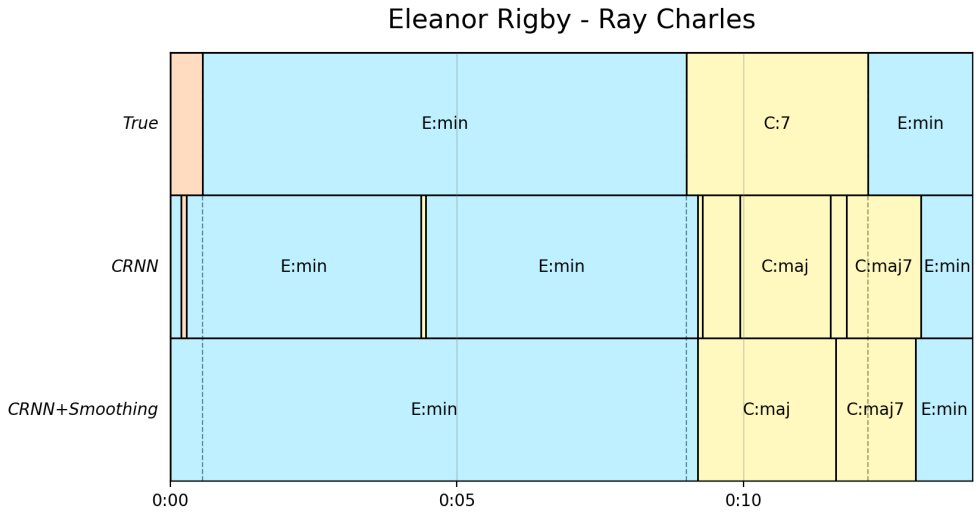

Figure A.8: An example of the effect of the HMM on the predictions of the *CRNN* model. The top plot shows the ground truth. The middle plot shows frame-wise predictions of the *CRNN* without smoothing. The bottom plot shows the predictions after smoothing. Chords are coloured by their equivalent chord 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.

A.16 Weighted Loss Confusion MAtrix

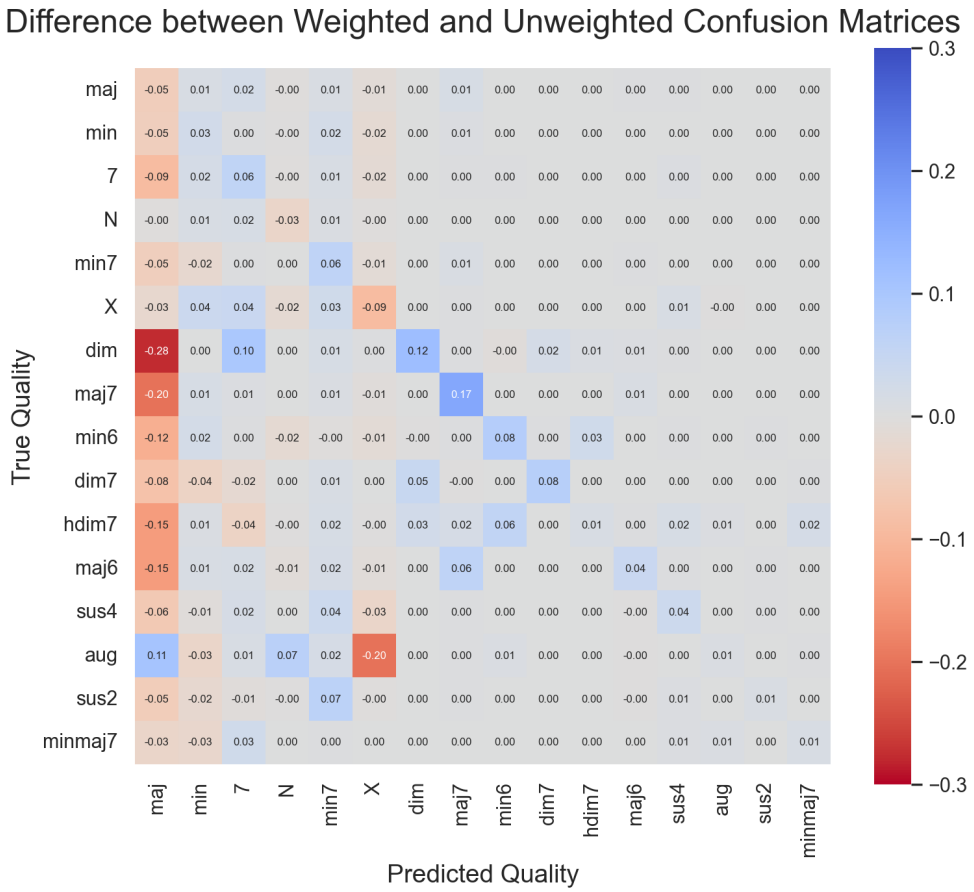

Figure A.9: Most of the diagonal entries in the confusion matrix increase. Recall on major7 qualities increases by 0.17. The only qualities to decrease in recall are major, N and X. I conclude that weighting the loss does improve the model. The weighted model predicts X 2.2 times less often. This may be how the weighted model improves class-wise metrics without sacrificing too much overall accuracy since X frames are ignored for evaluation.

A.17 Structured Loss Experiment Results

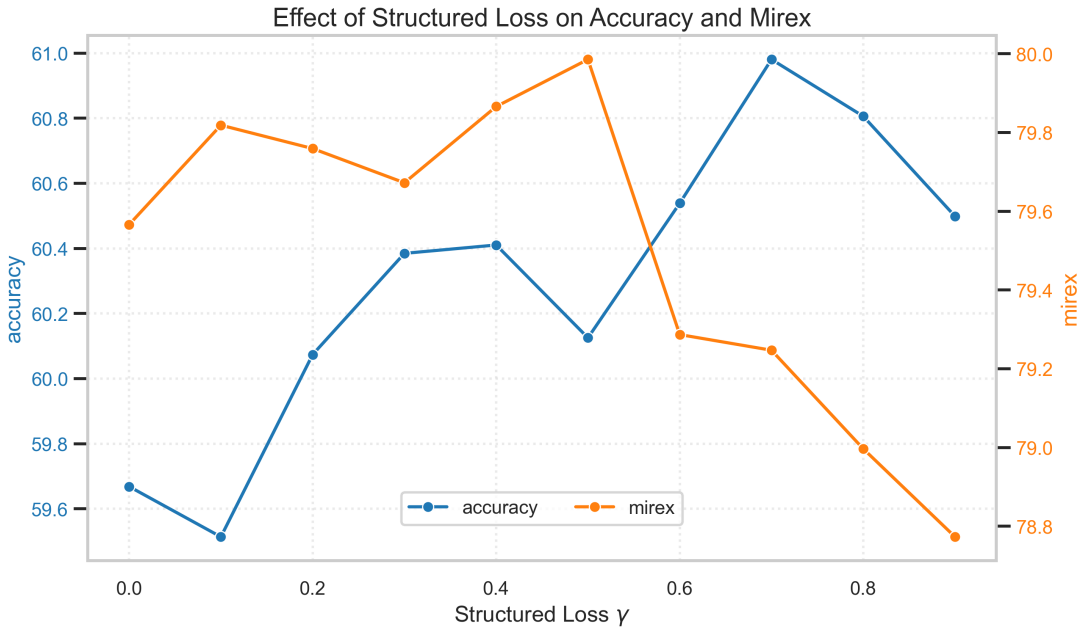

Figure A.10: Effect of structured loss on the *CRNN* model with varying γ. As we increase γ, accuracy improves but `mirex` behaves erratically, worsening at the higher end. The other metrics behave similarly to accuracy. I choose $\gamma = 0.7$ based on peak accuracy.

A.18 Details of Generative Feature Extraction

Overlapping 5 second chunks of audio were fed through MusicGen in a batched fashion. This first requires passing the audio through the pre-trained Encodec audio tokeniser [10]. These are then fed through the language model. I take the output logits as the representation for each frame. The model outputs logits in four 'codebooks', each 2048-dimensional vectors, intended to represent different granularities of detail in the audio. Audio segments are overlapped such that every frame has context from both directions. The multiple representations for each frame are averaged. Finally, these representations are upsampled. The model operates at a frame rate of 50Hz. To compute a representation with the same frame length as the CQT, I take the mean over the frames outputted by the model closest to the centre of the CQT frame. In case averaging over frames dampened the signal, I also tried linearly interpolating between the two closest frames outputted by the model. However, this was empirically found to perform slightly worse. Results are left to Appendix A.19. This feature extraction required the use of NVIDIA RTX A6000 GPUs. The extraction process takes 4 hours for each model over the entire dataset.

A.19 Upsampling Methods in Generative Feature Extraction

upsample	accuracy	root	third	seventh	mirex
area	**59.4**	**77.8**	**74.6**	**61.7**	**78.1**
lerp	58.4	77.7	74.0	60.6	78.2

Table A.6: Comparison of generative feature extraction with linear interpolation and area averaging for the musicgen-large and a linear projection down to 64 dimensions and averaging over the four codebooks. The results are very similar, but the area averaging method is slightly better in all metrics. I therefore choose to continue averaging over model frames in order to upsample to CQT frames.

A.20 Generative Feature Low-Dimensional Projection

d	accuracy	root	third	seventh	mirex
16	58.4	77.6	**74.5**	60.6	77.2
32	57.8	77.5	73.6	60.0	77.8
64	**59.2**	77.5	74.3	**61.4**	**77.9**
128	58.1	77.0	73.6	60.3	77.4
256	58.7	77.6	74.3	60.9	77.5
512	58.3	**77.9**	74.3	60.6	77.6
1024	58.4	76.7	73.5	60.7	76.7

Table A.7: Generative feature extraction with different projection dimensions, d. All results use the musicgen-large model and average across the four codebooks. There are no large differences between the different dimension reductions. I take $d = 64$ as it performs the best, but there is no evidence that this is not due to randomness in the optimisation process.

A.21 Generative Features with Different Models

model	acc
large	**60.2**
small	59.8
melody	59.9
chord	59.7

Table A.8: Results for generative features extracted from MusicGen [9] and MusiConGen [38] models with different musicgen-models. Only accuracy is reported as the other metrics are qually similar. These models are musicgen-large, musicgen-small, musicgen-melody and MusiConGen, referred to as large, small, melody and chord respectively. There are very few differences between the models. This suggests that the small model has just as much information in its internal representations that is useful for identifying chords as the other models. Another point of note is the comparison between chord and melody. The chord model is a chord-conditioned fine-tuned version of melody. It is surprising that the chord-conditioning did not help compared to the non chord-conditioned model.

A.22 Generative Features with Different Codebook Reductions

reduction	accuracy
concat	58.9
codebook_2	58.9
codebook_3	57.5
codebook_1	59.0
codebook_0	58.1
avg	**59.4**

Table A.9: Accuracy for different codebook reduction methods. Other metrics are omitted as they do not provide more information. All results are for musicgen-large with reduction down to 64 dimensions. Reductions of the form 'codebook_n' refer to training on codebook of index n from the model. Performance is similar across reductions except for codebook_0 and codebook_3 which perform worse. I choose the averaging reduction based on maximum accuracy.

A.23 Jazz Chord Progression Generation

The theory of functional harmony is a set of rules that govern the relationships between chords in a piece of music. While these rules are not always followed, many chord progressions can be parsed and broken down into the rules that have been followed

to create them. The rules are based on the relationships between the chords and the keys they are in. For example, a chord progression that moves from a tonic chord to a dominant chord is said to be following the rule of *dominant function*. This is a common rule in jazz music and is often used to create tension and resolution in a piece of music.

I first decide whether we are in major or minor, each with probability 0.5. I then uniformly sample a tonic from the set of notes in the Western chromatic scale. From this tonic, seven functional chords are decided before sequence generation. These are all probabilistic. For example, the tonic chord is always the tonic, but the dominant chord can be of `maj`, `7`, `sus4`, `aug` or `dim7` qualities. The probabilities are user-tuned but do not matter very much to the funcionality of the synthetic dataset.

For chord sequence generation, various rules are followed in a probabilistic manner. Progressions have a random length, uniformly sampled in the range $[4, 10]$.

- Tonic (I) may move to predominant chords (ii, IV, vi) or occasionally mediant (iii).
- Predominant chords (ii, IV) resolve to the dominant (V).
- Dominant (V) usually cadences back to tonic (I) or sometimes moves to vi.
- Tonic substitute (vi) leads to ii or iii.
- Mediants (iii) feed into vi.
- Unspecified or fallback transitions are routed toward ii to maintain forward motion.

A time-aligned chord sequence is then calculated in a similar format to that provided by the `jams` package. This assumes that each chord is played for one bar, that the BPM is always followed, and that MusiConGen simply loops over the chord progression if the end is reached. These assumptions were found to hold on manually inspected examples.

For further details and exact probabilities used, please refer to the provided code.[1]

A.24 Calibration to Handle Distribution Shift

To correct for the distribution shift between synthetic training data and the *pop* train split, I estimate the empirical class probabilities in each domain—$P_{\text{train}}(y)$ and $P_{\text{pop}}(y)$—and rescale the model's logits by the ratio

$$r(y) = \frac{P_{\text{pop}}(y)}{P_{\text{train}}(y)}. \tag{A.5}$$

In order for calibration to be root invariant, I take the mean ratio over chords that share the same quality, and a single calibration factor r_q is applied to every chord with that quality.

[1] `https://github.com/PierreRL/LeadSheetTranscription/blob/main/src/data/synthetic_data/chord_sequence.py`

Note that the model's outputs are in logits so the log ratio is added in implementation.

Figure A.11 shows the calibration of the model's outputs to account for distribution shift.

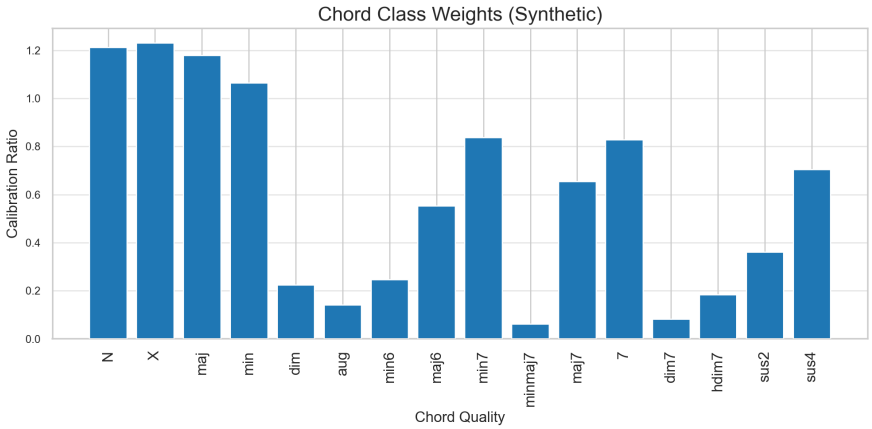

Figure A.11: Calibration ratios r_q of the model's outputs to account for distribution shift. The high ratio on rare chord qualities like `majmin` show that these qualities are much more common in the synthetic dta.

A.25 Difference in Confusion Matrixces with Synthetic Data

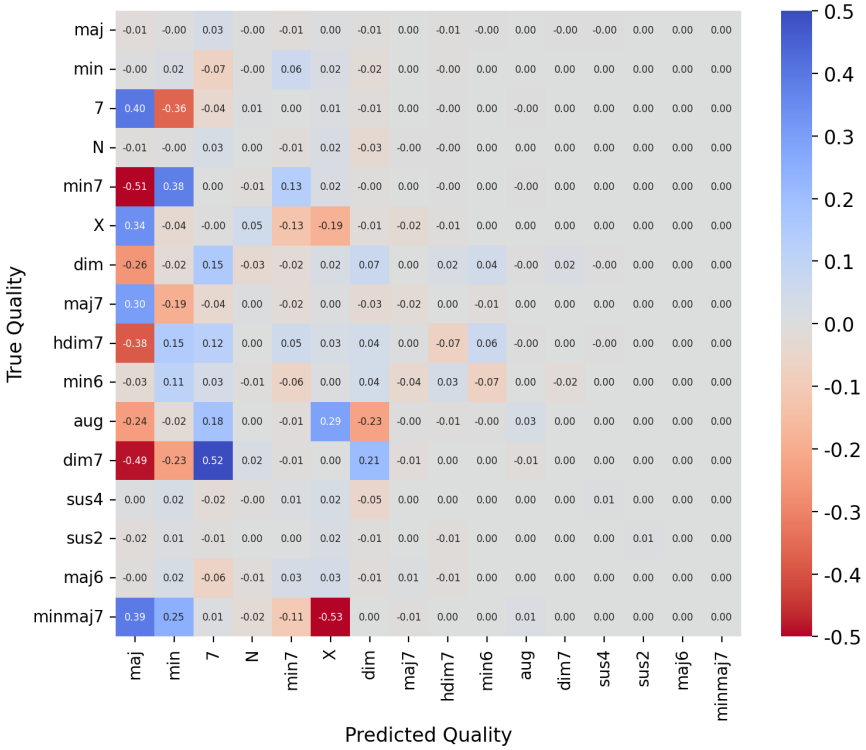

Figure A.12: Element-wise difference in normalised confusion matrix of the model trained on synthetic data and *pop* data versus just *pop* data with a weighted loss function. There are some notable differences. Training on synthetic data strongly discourages the model from predicting `X` chords, which are not present in the synthetic data. Recall on `min7` qualities increases by 13% and by 7% on `dim` qualities. However, recall `hdim7` and `min6` worsens. In general, the model predicts `maj` less often for rarer qualities. Another interesting observation is that the synthetic data corrects many of the predictions on `7` qualities from erroneous predictions of `min` to erroneous predictions of `maj`. Something similar happens with `min7` qualities. It is hard to say which model is better. Indeed, their overall accuracies are the same.

A.26 Maximum Lag Cross Correlation of Chord Transitions with Beats

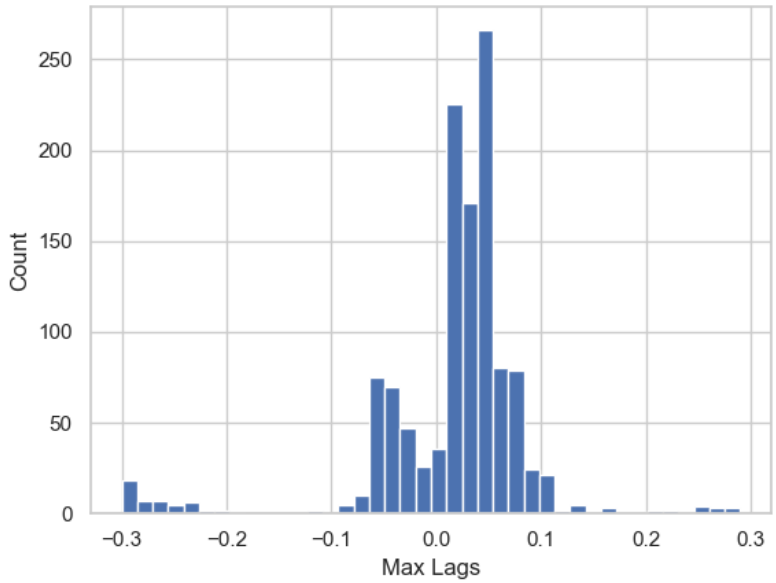

Figure A.13: Maximum lag cross correlation of chord transitions with beats within a window of 0.3 seconds. Almost all maximum lags occur between -0.1 and 0.1 seconds.