Jazz Piano Lead Sheet Arrangement via Deep Learning

Elliot Hogg
200817750

David Herbert
Mark Turner

27/08/2021

19300 Words

Declaration

I hereby declare that this dissertation represents my own work except where otherwise stated.

Abstract

Over the past 10 years, deep learning has been extensively researched and used in the task of music generation. However, one area that remains mostly untouched is the task of lead sheet arrangement. In the genre of jazz, most piano music is notated in the lead sheet format – which is an abbreviated representation of a song. The interpretation and arrangement of the music must be decided and undertaken by the performer. To date, the literature has not presented a lead sheet arrangement system that meets the needs of jazz pianists. This paper presents a novel lead sheet arrangement system (LSAS) that uses a conditional adversarial generative network (cGAN) to create full arrangements of jazz piano lead sheets. It also presents the Jazz-Chords dataset – a novel dataset of chord label-chord voicing pairs. Technical evaluation of the LSAS indicate that it can generate unique arrangements that are both accurate and musically pleasing. To the best of our knowledge, this work represents the first successful attempt at using deep learning to arrange full length piano lead sheets.

Acknowledgments

I would firstly like to thank my supervisor David Hebert, whose support and advice has been invaluable to this project.

I would also like to thank Mark Turner for his continued support throughout the project.

Contents

1. Introduction

Lead sheet arrangement is a task whereby a musician takes an abbreviated representation of a song and adds harmonic and rhythmic elements to transform it into a compelling piece of music. A piano lead sheet consists of two elements - the song's melody, and chord symbols. The melody is notated using western music notation and is played by the right hand of a pianist. The chord symbols are notated as letters positioned above the melody and represent the type of chord that the left hand will play. Figure 1.1 shows the first 4 bars of a piano lead sheet.

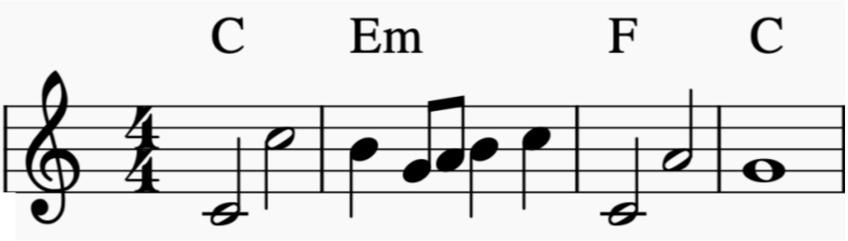


Figure 1.1 First 4 bars of the lead sheet for the song Over the Rainbow by Errol Garner. The melody is represented as western notation, and the chord types are indicated by the chord symbols above.

Lead sheet arrangement can be divided into two subtasks: harmonic arrangement and rhythmic arrangement. (1) Harmonic arrangement involves choosing a set of notes to be played for each chord symbol. The set of notes that represent a chord symbol are commonly referred to as a *chord voicing*. Musical theory defines a base voicing configuration for each chord symbol, however effective chord voicings adapt and rearrange that base configuration in a way that increases its musicality. (2) Rhythmic arrangement involves temporally arranging chord voicing notes to form a rhythmical pattern. Figure 1.2 highlights these two subtasks. The notes on the lower set of lines (bass clef) now instruct the left hand on what to play, and together with the right-hand melody on the top set of lines (treble clef), form a full arrangement.

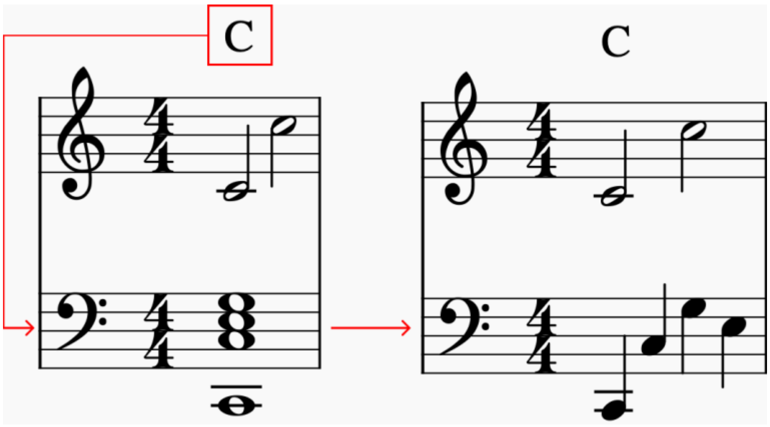


Figure 1.2 Lead sheet arrangement of the first bar from figure 1.1. From left to right, a C chord symbol (shown in red) is first arranged harmonically, and then arranged rhythmically.

In the area of jazz, a large proportion of the repertoire is notated in the lead sheet format (Martin, 2014:194). This creates a barrier to entry for pianists with limited jazz knowledge or experience, as lead sheet arrangement requires an extensive understanding of jazz music theory.

The development of a system that could take a jazz lead sheet and output a compelling arrangement would help to remove the barrier to entry that amateur and intermediate jazz pianists face. However, the development of such a system has received little commercial or academic attention (Ji et al., 2020).

There are some commercial applications, such as *Band in a Box* and *iRealPro*, that provide suggested chord voicings for the chord symbols of a given jazz song. However, they only display the chord symbols and suggested voicings, and not the melody. The systems are also reliant on a limited dictionary of predefined chord voicings.

Past academic research of lead sheet arrangement systems has typically followed a traditional programmatic approach of using decision-based algorithms pre-programmed with musical theory rules. Two programmatic systems of note are those proposed by Emura et al. (2006) and Watanabe et al. (2008), both of which can perform the task of harmonic lead sheet arrangement with reasonable success. However, as reported by the authors, there are two crucial limitations with conditional, rule-based arrangement systems. Firstly, they struggle to interpret and employ musical theory in a way that can achieve highly musical outcomes (Briot et al., 2017:6). And secondly, they struggle to produce variation, meaning that they cannot output differing arrangements of the same lead sheet (Briot et al., 2017:6).

Recently, the broader field of music generation has regained academic and public attention, largely due to the development of deep neural networks (Briot et al., 2017:3). This has in turn led to the beginnings of a deep learning based approach to lead sheet arrangement. The first and only authors of this approach, Liu & Yang (2018a, 2018b), were able to develop a generative deep learning led system that could perform harmonic arrangement on short segments of pop music lead sheets. Although the proposed system does not meet the requirements of jazz pianists, the results demonstrate that generative deep learning models are capable of producing both musical and varied arrangements.

This research project intends to build upon the current literature and develop a system that further bridges the gap between the requirements of jazz pianists and Liu & Yang's initial research. To do so, this project will first present the Jazz-Chords dataset, a novel dataset consisting of pairs of chord symbols and associated chord voicings extracted from a library of jazz piano solo arrangements. The project will also present a chord scraper program which was used to extract the Jazz-Chords dataset, and that can be used for further chord data extraction tasks. The project will present a generative deep learning model trained using the Jazz-Chords dataset to be able to generate musically pleasing and varied chord voicings for the chord symbols of a given lead sheet. And finally, the project will present a lead sheet arrangement system (LSAS) which takes as input a lead sheet, and through data manipulation and the use of the trained generative deep learning model, can output a large number of varied, musical arrangements of that lead sheet. The formal aim and objectives of the project are presented below.

1.1 Aim

The aim of this research project is to create a deep learning led system that can generate varied and musical arrangements of any given jazz lead sheet.

1.2 Objectives

The projects technical objectives are as follows:

1. To gather a library of machine readable, fully arranged pieces of jazz piano music which have annotated chord symbols.

2. To develop a program that can scrape chord symbols and their associated chord voicings from fully arranged pieces of piano music.

3. To present a novel dataset of jazz piano chord symbol-chord voicing pairs.

4. To create a generative deep learning model that can generate accurate and varied chord voicings for a given chord symbol.

5. To train the generative model using the novel dataset.

6. To develop a system that takes a lead sheet, extracts its chord symbols, generates chord voicings for them using the trained generative model, and outputs a harmonically arranged version of the lead sheet.

1.3 Structure

The structure of the report follows both the order in which the project was undertaken, as well as the order in which the system as a whole operates.

- **Section 2** presents all of the background research that was undertaken as part of this project. A comprehensive review of generative deep learning in the domain of music is presented, as this facilitated the search for a suitable model architecture for the generative model. A comprehensive review of available datasets is also presented.

- **Section 3** presents the process of gathering the fully arranged solo piano music library from which the Jazz-Chords dataset was extracted from.

- **Section 4** presents the development of the chord scraper system that was used to extract the Jazz-Chords dataset. It also presents the dataset and evaluates the effectiveness of the chord scraper tool.

- **Section 5** presents the generative deep learning model used to generate chord voicings. This includes how the Jazz-Chords dataset was pre-processed and encoded, as well as some initial deep learning experiments that helped to direct the development of the generative model.

- **Section 6** presents the development and capabilities of the Lead Sheet Arrangement System (LSAS). The section also presents a technical and participant led evaluation of the LSAS's output arrangements.

- **Section 7** presents a conclusion of the project with commentary on how the initial aims and objectives were met, how the project fits within the current literature, and some insights and recommendations for future research.

1.4 Source Code

All of the source code for the project can be found here – link.*

To install the project, please see the readme in the root directory.

The source code is split into three subdirectories, which all have their own readme instructions.

The *chord_scraper* directory houses the chord scraper system and Jazz-Chords dataset as outlined in section 4.

The *chord_generator* directory houses all of the data encoding scripts and deep learning models including the trained cGAN model outlined in section 5.

And the *leadsheet_arranger* houses the overarching Lead Sheet Arrangement System (LSAS) outlined in section 6.

*If accessing the project offline, please see submitted zip file.

2. Background Research

2.1 Motivations

The initial motivations for this project were to make a meaningful and novel contribution to the jazz piano community within the context of a computer science project. As a keen jazz pianist with a classically trained background, the task of lead sheet arrangement has always been a challenge. In order to create arrangements that sound good, a large amount of knowledge and experience of voicing chords is required. Developing a system that could generate chord voicings that were indistinguishable from the voicings of renowned pianists such as Bud Powell, Bill Evans, or Herbie Hancock would be of great use to me and others in my position.

This is the motivation that led me to pursuing a research project in the field of deep learning and lead sheet arrangement. I hope that this research can at the very least, serve as a springboard for future research to be carried out in this area.

2.2 Generative Deep Learning Models

As mentioned in the introduction, there is a limited amount of previous research on the task of lead sheet arrangement using deep learning. However, as music arrangement is a subtask of music generation, research on music generation is also highly relevant to this project. In this subsection, lead sheet arrangement and music generation literature is surveyed to assess the capabilities of current research and see how effective their models are in the task of arrangement.

In doing this research, the aim is to both find pre-existing models that can be adapted to fit the needs of this project, as well as to find a model architecture that can be used to implement a generative model that can accomplish the task of lead sheet arrangement.

In order to thoroughly survey the landscape of arrangement and generation, a full scrape of the field was conducted. The findings were filtered and reduced into a list of 35 research papers. Meta-information was manually extracted from each paper; this included the deep learning models used, datasets used, and whether the models were publicly available. This information is presented in a table and can be found at Appendix A.

The first paper of note is titled ‘Lead Sheet Generation and Arrangement by Conditional Generative Adversarial Network’ (Liu and Yang, 2018b). The authors were able to generate pop music lead sheets and then perform harmonic arrangement on them. As the title suggests, the model used was a conditional generative adversarial network or cGAN. The results of the paper demonstrated that this model is effective in the task of harmonic arrangement, as it was able to generate convincing and varying chord voicings for the chord symbols of a lead sheet.

One major limitation of this model however was that it was not capable of generating the more complex chord types used in jazz. However, this was not a fault of the model per se, but due to it being trained using a dataset of pop song arrangements. In order to gain further insight into the cGAN model architecture, the project’s source code was downloaded from GitHub. However, attempts to train and run the model failed. This was due to the list of dependencies being incomplete and some dependencies being no longer available. Contact was attempted with the authors; however, no response was received.

This paper was very useful to this project, as it highlighted a deep learning model capable of the task of lead sheet arrangement.

A further 3 papers were evaluated in which the authors used cGANs or generative adversarial networks (GANs) in the task of music generation (Dong et al., 2018; Liu et al., 2018; Szelogowski, 2021). Each model was able to generate novel instances of music that were similar to the music used to train the model. For example, Dong et al. (2018) trained a GAN using a dataset of pop songs arranged for a 4-piece band. The trained model was able to generate new, unique, and varied instances of pop song band arrangements that were similar to the arrangements in the training data.

The second paper of note was titled 'Chord Jazzification: Learning Jazz Interpretations of Chord Symbols' (Chen et al., 2020). Although this paper made no mention of the lead sheet, the model presented was able to take as input a sequence of chord symbols and output a series of chord voicings to represent those symbols, which is fundamentally the same task as harmonic lead sheet arrangement.

The main insight gained from this paper was the way in which the chord symbols and chord voicings were encoded. The authors presented an effective way of encoding chord symbols as integers and encoding chord voicings as binary vectors. This approach provided a useful framework from which this research project could encode its own data.

Chen et al. used a recurrent neural network (RNN) to generate chord voicings. The reason this type of network was used is that it excels at generating sequences of chord voicings in which each chord voicing can be assumed to be dependent on previous ones (Chen et al., 2020). RNNs can generate chord sequences that are smoothly connected and flow together, which is a musical concept referred to as *voice leading* (Chen et al., 2020).

However, one major limitation of the proposed network was that the chord voicing sequences it generated were limited to a maximum of 8 bars in length, which is far less than the average length of a jazz piano lead sheet. This limitation was assumably a result of both general constraints of RNN architecture, as well as a lack of substantial training data.

A further 5 research papers using RNNs in the task of music generation were evaluated. Each of the papers indicated that RNNs did not perform well at generating long sequences of music when trained with small datasets (Hadjeres et al., 2017; Liang et al., 2019; Teng et al., 2017; Zhao et al., 2020; Zhu et al., 2020). The meta-information of these papers can be found in the table at Appendix A.

This suggests that RNNs would not be suitable to the requirements of this project, as the sheer amount of machine-readable jazz piano data required to train them in order to output full length lead sheet arrangements is not currently available (Ji et al., 2020).

One further generative deep learning model was highlighted in a journal titled 'A Comprehensive Survey on Deep Music Generation: Multi-level Representations, Algorithms, Evaluations, and Future Directions' (Ji et al., 2020). Ji et al. cited several papers that use Variational Autoencoders (VAEs) in the task of music generation. However, the authors of these papers generally reported mixed results. The most notable of these papers was that of Borghuis et al. (2018). In the paper, the authors reported the observation of blurriness in generated samples whereby "large clusters of notes (were) being played together and in sometimes obsessive repetitions of short notes". This was concluded as being a result of the VAE "posterior collapse" problem – an issue in which VAEs struggle to reproduce instances of data that are similar to the training set (He et al., 2019:1). A further two VAE based music generation papers reported similar issues (Engel et al., 2017; Valenti et al., 2020).

After analysing the results of the filtered list of 35 research papers, it was determined that a cGAN model would be the most effective deep learning model for this project. In order to learn more about cGANs and their use in generation tasks, a wider scope of literature was searched. There were two papers of note which both used cGANs to perform tasks that were fundamentally the same as harmonic lead sheet arrangement.

The first paper was titled 'Conditional generative adversarial nets' (Mirza & Osindero, 2014). The authors proposed the cGAN model architecture for the first time and demonstrated its capabilities in a number of generation tasks. One of those tasks was the conditional generation of numeric digits. Their implementation of the cGAN model was trained using the MNIST handwritten digit dataset (Deng, 2012), and could be given as input a number between 0 and 9 and generate a corresponding image of that digit. All of the generated images appeared to be from the same data distribution as the training dataset, meaning that they looked indistinguishable from the images in the MNIST dataset.

The second paper was titled ‘Unsupervised representation learning with deep convolutional generative adversarial networks’ (Radford et al., 2015). In this paper, the authors expanded on Mirza & Osindero’s work by applying the cGAN architecture to a number of more advanced image generation tasks. Radford et al.’s cGAN model was able to successfully generate images of human faces. Importantly, the model was able to generate an infinite number of unique faces, of which all appeared to be from the same data distribution as the training images.

The results of these papers suggest that a cGAN model could be used to conditionally generate chord voicings if trained on a large dataset of chord symbol-chord voicing pairs. The results also suggest that the generated chord voicings would be highly varied, but also within the data distribution of the training data, and therefore made up of notes from the correct base voicing configuration. The two above papers also encourage the idea of representing chord voicings as images.

2.3 Generative Adversarial Networks (GANs)

This section serves as a precursor to understanding conditional generative adversarial networks (cGANs).

Generative adversarial networks are a type of neural network developed by Ian Goodfellow and his colleagues in 2014 (Goodfellow et al., 2014). For the reader to understand this paper, a high-level explanation of GANs is provided. However, for a more in depth understanding, please see Goodfellow et al., (2014).

The purpose of a GAN is to be able to generate new instances of data from a given data distribution. For example, if a GAN was trained on a dataset of images of faces, it would then be able to generate unique images of faces that would look as though they were part of the original dataset.

A GAN is made up of 2 independently functioning neural networks - a discriminator, and a generator (Goodfellow et al., 2014: 3).

The generator is a neural network that takes a random point within a specified noise distribution as input, and outputs a fake instance of data. To use the previous example, the generator would take an array of random numbers, and output a “fake” image of a face.

Initially, the generated fake images would be very poor, however through training the generator, it would eventually learn how to transform each point in the noise distribution into an image that was indistinguishable from the images in the dataset.

The discriminator is a neural network that operates in a similar way to a classification network (Goodfellow et al., 2014). It takes as input both real instances of data, and generated, fake instances of data, and outputs a number indicating whether it thought the input was real or fake.

When initially training a GAN model, the discriminator easily classifies the real instances as real, and the generated instances as fake. However, throughout training, the generator adjusts its parameters in a direction that make it better at fooling the discriminator. In theory, the generator model is fully trained when the discriminator has a 50% chance of classifying real instances as real, and fake instances as fake (Goodfellow et al., 2014).

2.4 Conditional Generative Adversarial Networks (cGANs)

Conditional generative adversarial networks (cGANs) allow the output of the generator model to be conditioned on a given input (Mirza & Osindero, 2014). An example of this would be a cGAN model that was trained on a dataset of images of faces of men, woman, boys, and girls – all of which were labelled numerically from 1 to 4. When training the cGAN model, the numerical labels would be given as an additional input to both the generator and discriminator. When the trained generator model was given 1 of the 4 labels, it would generate a face of that labels training data distribution. For example, inputting a label of '1' to the trained generator would produce a unique image of a man's face that would look as though it was part of the training dataset.

2.5 Available Datasets

Deep learning generation models require a large amount of data for training (Goodfellow et al., 2014). cGANs require data that is labelled. For this research project, the data must consist of a large number of chord voicings which are labelled with their associated chord symbol.

In other deep learning fields, such as image generation, extended research over the last two decades has yielded a great number of refined datasets for which researchers can use to train their models (Mogadala et al., 2019). However, as the field of deep learning in music is still relatively small, the availability of datasets is not comparable (Ji et al., 2020).

In order to determine if there was a pre-existing dataset that could train the cGAN model in this project, an extensive survey of available music datasets was conducted. The survey found 15 jazz-related datasets. However, after conducting a survey, a much more comprehensive list of datasets was found (Ji et al., 2020:56-58). Therefore, only the datasets that are not present in Ji et al.'s paper are presented below:

Format	***Type***	***Contains***
Text	Jazz Audio-Aligned Harmony (JAAH) Dataset	Meter, structure, and chords of 113 Jazz tracks
	Weimar Jazz Database (WJazzD)	Transcriptions of 135 jazz solos
	JazzCorpus	Annotated chord progressions for 77 jazz pieces
	Chord-Jazzification dataset	Annotated chord voicings from 50 pop piano solos
MIDI	LMD-matched	45,129 songs matched to the Million Song Dataset
	Big_Data_Set: "The Largest MIDI Collection on the Internet"	130,000 songs including jazz solos
MusicXML	Charlie Parker's Omnibook data	Transcriptions of 50 Charlie Parker improvisations

Table 2.1 Jazz-related datasets that are available and open to use. This is an extension of Ji et al.'s (2020) dataset survey.

Following a survey of all the available datasets, only one was identified as being of partial interest - The Chord-Jazzification dataset. This was the only existing dataset that consisted of chords symbol-chord voicing pairs. However, it was deemed unsuitable as (despite the paper title) the chord pairs were predominantly extracted from pop songs, and therefore did not contain the more complex chords used in jazz.

Following this survey, it was determined that a novel dataset of chord symbol-chord voicing pairs would need to be gathered in order to train the cGAN. Gathering the dataset would involve scraping fully arranged jazz piano solos which had chord symbol annotations.

2.6 Existing Chord Scraping Tools

The deep learning model in this project requires a large dataset of chord symbol-chord voicing pairs. As no such dataset exists, it must be gathered from existing fully arranged jazz piano solos with chord symbol annotations. A search of the literature provided no instances of any previous research that had developed a scraper that was capable of such a task. Further searches of existing software and GitHub repositories presented 3 tools which were capable of extracting chord symbols from sheet music in varying formats, however the tools did not extract the notes of the chord voicings. It was quickly concluded that a bespoke chord scraper tool would need to be developed as part of this project in order to extract the required dataset.

This chord scraper tool is presented in section 4.

2.7 Technologies

The Python programming language was identified as a highly suitable technology to develop all of the systems within this research project. The reason for selecting Python was both its effectiveness as a scripting language and its array of inbuilt data manipulation packages. It was also chosen as it supports a number of libraries that are used in this project.

The first of these libraries is Keras, which is an interface for the TensorFlow library. TensorFlow contains implementations of neural-network building blocks and is the most commonly used library in deep learning research. It is used in the deep learning tasks of this project.

The NumPy library is also used in this project. It provides support for working with large multidimensional arrays and matrices and is used to manage and manipulate the project's datasets.

The other Python library of note that is used is Matplotlib. This provides extensive data plotting functionality and is used to analyse and present data throughout the project.

All scores and audio representations of sheet music are created using MuseScore, which is a piece of widely used score writing software.

The project is developed in a virtual environment through the use of the Python library venv. Version control is managed using Gitlab.

A Microsoft Azure NC-series virtual machine with a NVIDIA Tesla K100 GPU is used for neural network training.

3. Gathering a Library of Jazz Piano Solo Arrangements

3.1 Initial Research

Training cGANs requires a large amount of labelled data. In the context of this project, the data is chord voicings, and the labels are their associated chord symbols. Figure 3.1 shows how these data pairs would be represented inside a fully arranged jazz piano solo score.

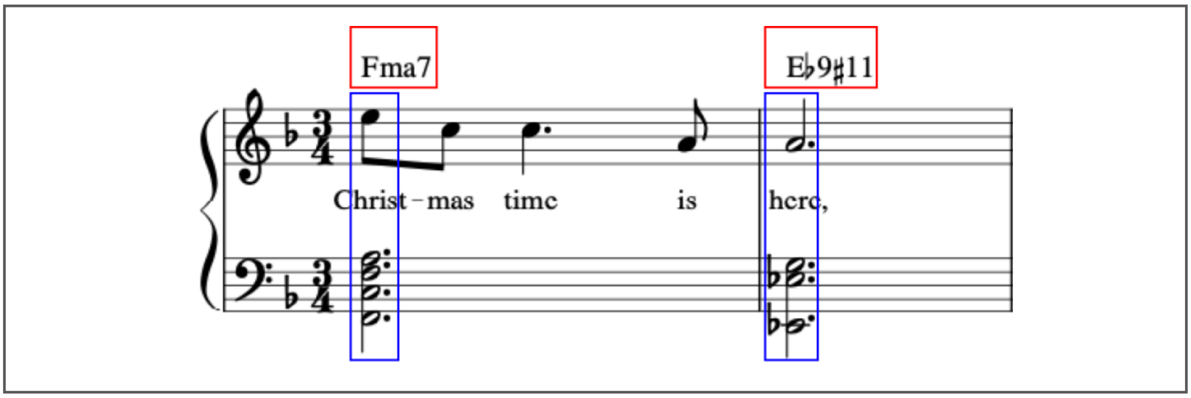


Figure 3.1 Musical score showing chord symbols (red) and chord voicings (blue) as they would appear in a fully arranged jazz piano solo.

A review of available datasets in section 2.5 found one corpus which contained chord symbols and chord voicings; however the chords were taken from pop songs and thus didn't meet the requirements for this project.

In order to gather a novel dataset of labelled chord voicings, a large library of fully arranged jazz piano solos with annotated chord symbols would need to be scraped to extract chord symbols-chord voicings pairs. The solos would need to be of a certain degree of quality in that they would need to contain chord symbols that accurately described their associated chord voicings, as well as chord voicings that were highly musical.

Machine readable music is most commonly represented in either MIDI or MusicXML format (Ji et al., 2020). As MIDI files cannot contain chord symbols, a library of jazz piano solos in MusicXML format was required.

An extensive survey of available MusicXML libraries was found on the website musicxml.com. The survey showed that a website called *MuseScore.com* had over 1 million scores in XML format, significantly more than any of the other libraries.

The search function on MuseScore showed a total of 4,837 available jazz piano solos. However only 300 of the solos were fully arranged and contained annotated chord symbols. The others were either lead sheets, or full arrangements without chord symbols. Of the 300 suitable solos, 171 were manually selected. Further details of this are found on the next page.

3.2 Fully Arranged Jazz Piano Solo Library

171 fully arranged jazz piano solos with annotated chord symbols were manually downloaded from *musescore.com*. In order for a solo to be selected, it must have met all of the following criteria:

1. Fully arranged, i.e., both the treble (right hand) and bass clef (left hand) present
2. Have annotated chord symbols indicating chord type
3. Composed by a recognised jazz musician
4. Arranged by a reputable user with a peer reviewed score of 4.7 stars out of 5 or above

The solos are in MusicXML format, which is an XML-based file format. More information on the MusicXML format is presented in the next subsection.

The full library can be found here - link*.

*If accessing the project offline the library can be found in the chord_scraper subdirectory.

3.3 MusicXML

In order to contextualise section 4, a brief explanation of MusicXML is provided. For a more detailed understanding, see Good (2001).

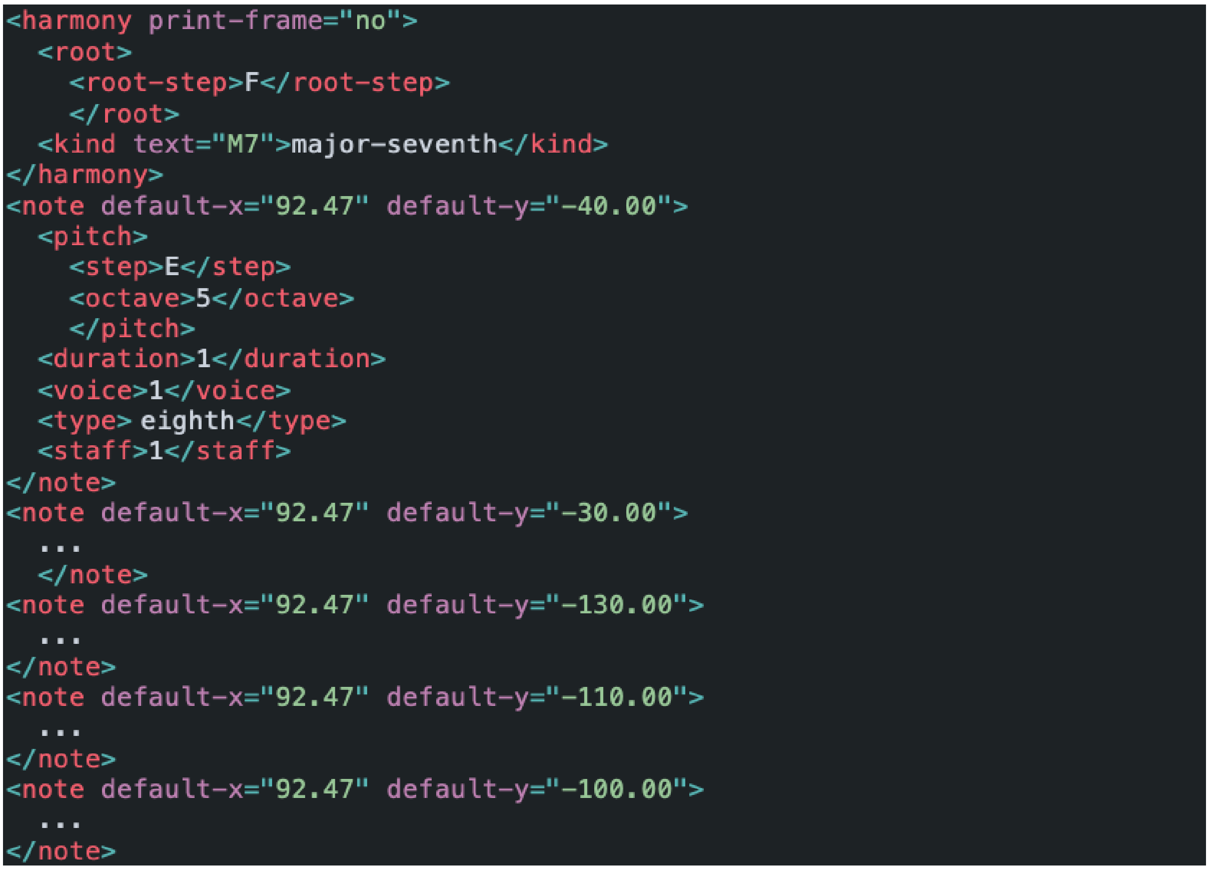
MusicXML is an XML-based markup language for representing musical scores. Like MIDI, MusicXML is standardised, and can be interpreted by all major music notation software and displayed as a musical score.

Figure 3.2 on the next page shows how the first chord symbol-chord voicing pair in figure 3.1 is represented in MusicXML format.

Looking at figure 3.2, it can be seen that the chord symbol is represented by the use of a harmony element. The inner root element indicates the bottom note of the chord voicing, and the kind element indicates the chord type. The harmony element can contain additional elements for more complex chords.

The notes that make up the chord voicing are represented as note elements. It can be seen that there are a total of 5 note elements for the 5 notes in the chord voicing.

The default-x attribute represents the x-axis position of each note and is helpful for determining which notes make up a chord voicing as the notes that make up a voicing are often vertically aligned. The default-x attribute was key to the chord scrapers functionality, as explored in the following section.



```
<harmony print-frame="no">
  <root>
    <root-step>F</root-step>
    </root>
  <kind text="M7">major-seventh</kind>
</harmony>
<note default-x="92.47" default-y="-40.00">
  <pitch>
    <step>E</step>
    <octave>5</octave>
    </pitch>
  <duration>1</duration>
  <voice>1</voice>
  <type> eighth</type>
  <staff>1</staff>
</note>
<note default-x="92.47" default-y="-30.00">
  ...
  </note>
<note default-x="92.47" default-y="-130.00">
  ...
</note>
<note default-x="92.47" default-y="-110.00">
  ...
</note>
<note default-x="92.47" default-y="-100.00">
  ...
</note>
```

Figure 3.2 First chord symbol-chord voicing pair from figure 3.1 represented in MusicXML format.

4. Chord Scraper

This section will first detail the Chord Scraper system developed and used in this project to extract the Jazz-Chords dataset. A high-level overview of the system is provided as well as a detailed report of how the system was developed. Following this, the extracted Jazz-Chords dataset is presented and evaluated. Finally, there is an evaluation of the effectiveness of the Chord Scraper system. Detail on how the system requirements were met is presented in section 4.4.2.

4.1 System Requirements

1. To extract chord symbol-chord voicing pairs from MusicXML solo piano scores.
2. To output raw data as well as data that has been partially cleaned and refined (see section 4.2).
3. To work on all piano solos in MusicXML format including previously unseen solos.
4. To be able to scrape a large number of scores (more than 100) in a reasonable amount of time (less than 5 minutes).
5. To extract chord voicing notes with a high degree of accuracy (more than 97%).
6. To have a highly usable command line interface that enables other researchers to use the tool.

4.2 System Overview

As presented in figure 4.1, the chord scraper takes as input a number of piano scores in MusicXML format, and outputs chord data in 2 different formats. The first output format represents the chords exactly as they were engraved in the input files, whereas the second transforms the chords into a structure that is more suited to machine learning model inputs. Examples of these two output types are explored below and displayed in figures 4.4 and 4.5.

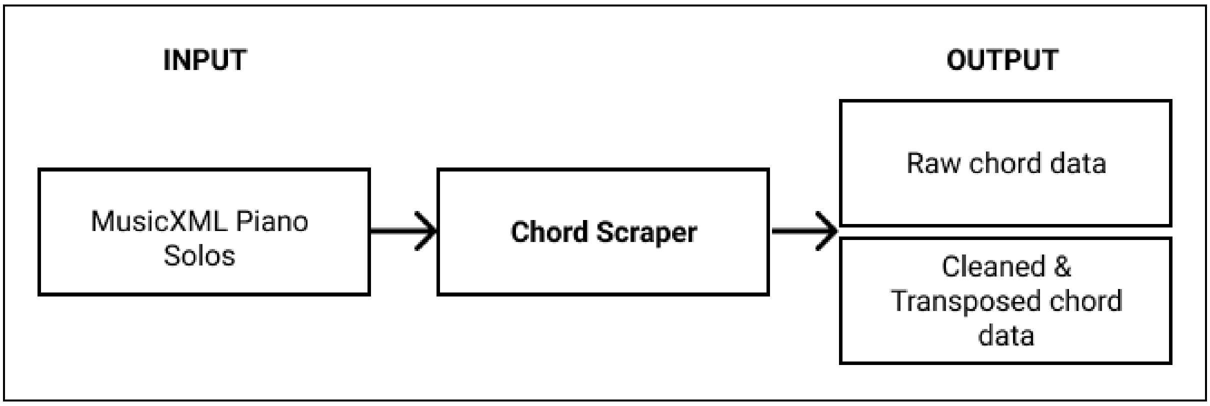


Figure 4.1 System diagram showing the chord scraper system.

In order to highlight the workings of the system, a small scraping example is presented below.

Figures 4.2 and 4.3 show a 1-bar extract from a fully arranged jazz piano solo as a score view and in its MusicXML encoding. The numbered labels seen in figure 4.2 map to the numbers at the start of each line in figure 4.3 and show how the chord symbols and notes map to their associated harmony and note MusicXML elements. The numbered labels are also colour coded to indicate chord symbol-chord voicing pairs. For example, label 1 points to a G minor-seventh chord symbol and associated MusicXML harmony element, and labels 2 & 12-15 point to the notes and associated note elements that make up the G minor-seventh chord voicing.

Figures 4.4 and 4.5 show the resulting outputs of passing the MusicXML from figure 4.3 into the chord scraper.

The raw chord data format seen in figure 4.4 stores chord symbols in the root, type, & extensions columns. For example, the first line of data represents the first chord (yellow labelled) in figure 4.2 - the root is G, and the type is minor-seventh. The raw chord data format stores chord voicings in the note_labels & note_numbers columns. The first line of data shows 5 note labels that represent the five notes that make up the G minor-seventh chord. The note numbers represent the note labels using a note number system in which the numbers 1 – 88 denote the notes of the piano in ascending (left to right) order.

The cleaned and transposed chord data seen in figure 4.5 represents the raw chord data after undergoing two data processing steps. First, all of the substandard chord pairs are removed or altered. Then the remaining chords are transposed so that they all have the same root note. This essentially equalises all of the chord voicings whilst still keeping their individual structures. As a result, root and note_label information is no longer needed. Details of these two data processing steps are soon outlined in section 4.3.2.

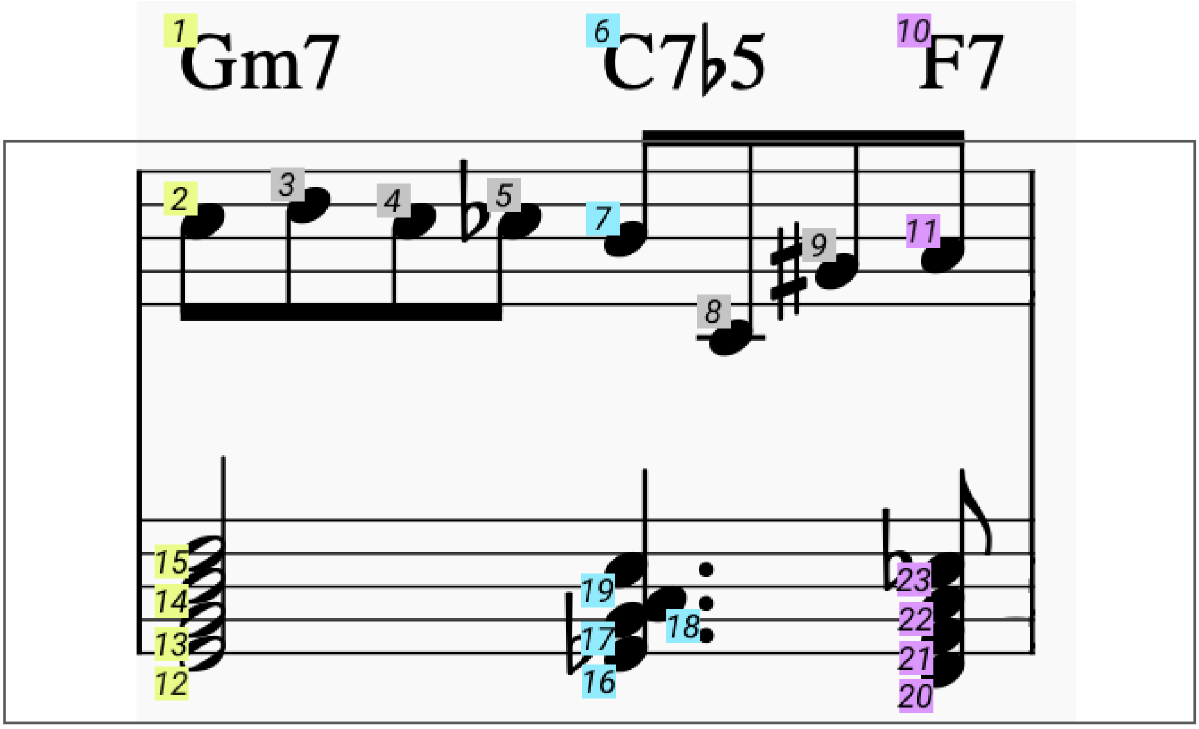
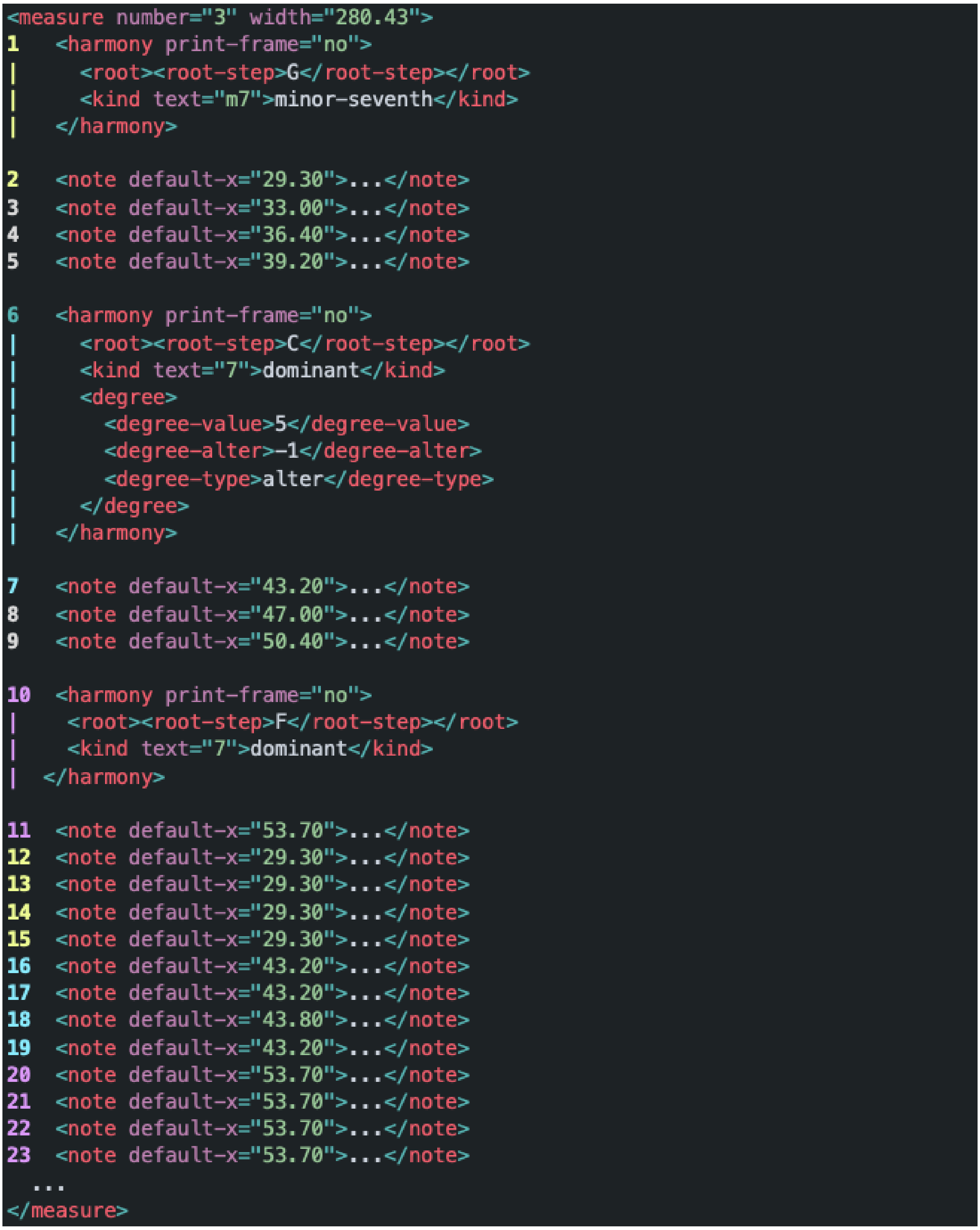
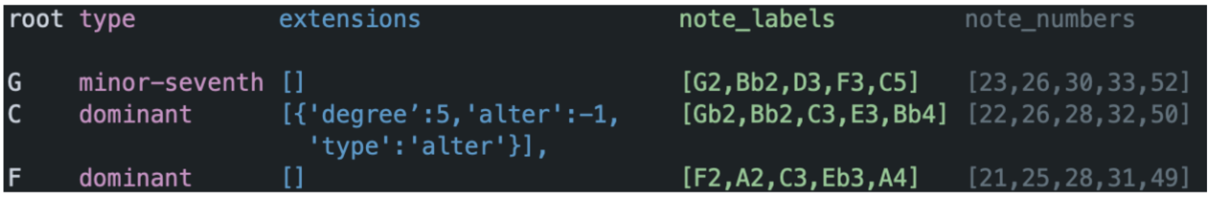


Figure 4.2 1-bar score representation of a MusicXML jazz piano solo score showing how elements are ordered. The top five lines represent the treble clef and are played by the right hand. The bottom 5 lines represent the bass clef and are played by the left hand.



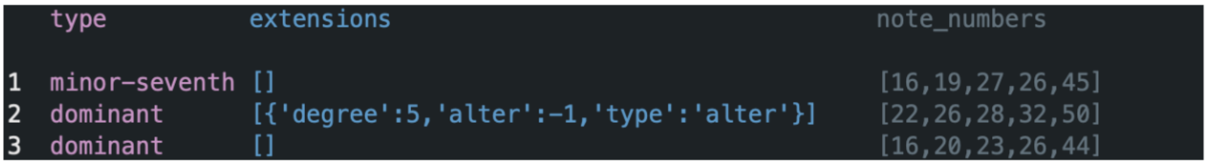
```
<measure number="3" width="280.43">
1    <harmony print-frame="no">
|      <root><root-step>G</root-step></root>
|      <kind text="m7">minor-seventh</kind>
|    </harmony>

2    <note default-x="29.30">...</note>
3    <note default-x="33.00">...</note>
4    <note default-x="36.40">...</note>
5    <note default-x="39.20">...</note>

6    <harmony print-frame="no">
|      <root><root-step>C</root-step></root>
|      <kind text="7">dominant</kind>
|      <degree>
|        <degree-value>5</degree-value>
|        <degree-alter>-1</degree-alter>
|        <degree-type>alter</degree-type>
|      </degree>
|    </harmony>

7    <note default-x="43.20">...</note>
8    <note default-x="47.00">...</note>
9    <note default-x="50.40">...</note>

10   <harmony print-frame="no">
|     <root><root-step>F</root-step></root>
|     <kind text="7">dominant</kind>
|   </harmony>

11   <note default-x="53.70">...</note>
12   <note default-x="29.30">...</note>
13   <note default-x="29.30">...</note>
14   <note default-x="29.30">...</note>
15   <note default-x="29.30">...</note>
16   <note default-x="43.20">...</note>
17   <note default-x="43.20">...</note>
18   <note default-x="43.80">...</note>
19   <note default-x="43.20">...</note>
20   <note default-x="53.70">...</note>
21   <note default-x="53.70">...</note>
22   <note default-x="53.70">...</note>
23   <note default-x="53.70">...</note>
  ...
</measure>
```

Figure 4.3 1-bar excerpt of a MusicXML jazz piano solo showing how elements are ordered.



```
root type          extensions                  note_labels          note_numbers

G    minor-seventh []                          [G2,Bb2,D3,F3,C5]    [23,26,30,33,52]
C    dominant      [{'degree':5,'alter':-1,    [Gb2,Bb2,C3,E3,Bb4]  [22,26,28,32,50]
                     'type':'alter'}],
F    dominant      []                          [F2,A2,C3,Eb3,A4]    [21,25,28,31,49]
```

Figure 4.4 Raw chord data representing chords in figure 4.2/4.3 (CSV format).



```
  type          extensions                                   note_numbers

1 minor-seventh []                                           [16,19,27,26,45]
2 dominant      [{'degree':5,'alter':-1,'type':'alter'}]     [22,26,28,32,50]
3 dominant      []                                           [16,20,23,26,44]
```

Figure 4.5 Cleaned and transposed chord data representing chords in figure 4.2/4.3 (CSV format).

4.3 Development

The initial aim when developing the system was to write a script that could effectively and consistently extract chord information from MusicXML files. Following this, an additional script was written in order to transform the extracted raw chords data into a format that would be more suitable to the machine learning model used for chord generation, as well as for other potential users. As mentioned above, this involved removing or altering invalid chords, and transposing each chord so that it had a root of C.

For the purposes of readability, these two scripts are presented as separate entities, however it must be noted that in reality they are to some extent, interconnected, and therefore exist inside the same Python file. The chord scraper can be found inside the *chord_scraper* directory at the root of the project.

4.3.1 Chord Extraction

The first functioning version of the script worked by iterating through each line in each MusicXML file and using regular expressions (Regex) to extract the required data. The script worked by first finding a harmony element and extracting the chord symbol information. However, there was an initial issue in locating all of the note elements that represented each associated chord voicing.

The way in which MusicXML is formatted means that the first note element after each harmony element is the top note of that chords voicing. However, because MusicXML lists all of the treble clef notes (top lines) before the bass clef (bottom lines) notes, finding the remaining notes of a chord voicing using the order of elements was not possible. This issue is highlighted in figures 4.2 and 4.3. For example, the first chord symbol and the top note of its associated chord voicing are in position 1 and 2. However, the remaining notes in the chord voicing are in positions 12-15. Finding these remaining notes by relying on the order of the elements was not possible, as the position of bass clef notes is arbitrary.

After further analysing the MusicXML markup syntax, it was discovered that each note element had a default-x attribute, which denoted the notes *x*-axis coordinate. By using the default-x attribute of the note directly after each harmony element, the other notes that made up that chord voicing could be found. For example, looking back at figure 4.2, the default-x attribute of note 2 could be used to locate and extract notes 12-15, as they are all vertically aligned. Their associated note MusicXML elements in figure 4.3 confirm this, as they all have a default-x value of 29.30.

The issue with this approach is that there are cases in which some of the notes in a chord voicing are not vertically aligned.

The first cause of this is when there are 2 notes that are next to each other, and vertically aligning them would cause the notes to overlap and thus be unreadable. Figure 4.6 highlights this issue, in which one note from each of the chord voicings is slightly offset and therefore has a different *x*-axis coordinate to the top note in the voicing. Looking at the figure, the vertical red line shows that using the default-x value of the top note misses a note from each voicing.

The second cause of this issue is when chord voicings are temporally arranged to form a rhythmic pattern (rhythmically arranged). This means that the notes that make up the chord voicing are not vertically aligned, but adjacent to one another. This is highlighted in figure 4.7, in which only 1 note has the same "default-x" attribute as the top note.

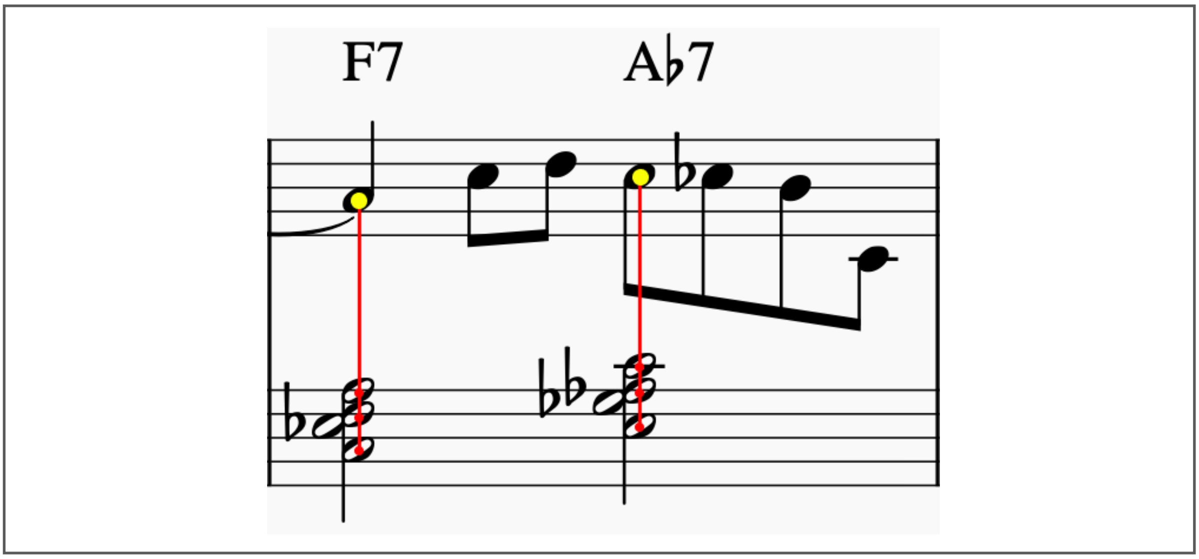


Figure 4.6 Musical score highlighting how "x-location" values can miss vertically offset notes.

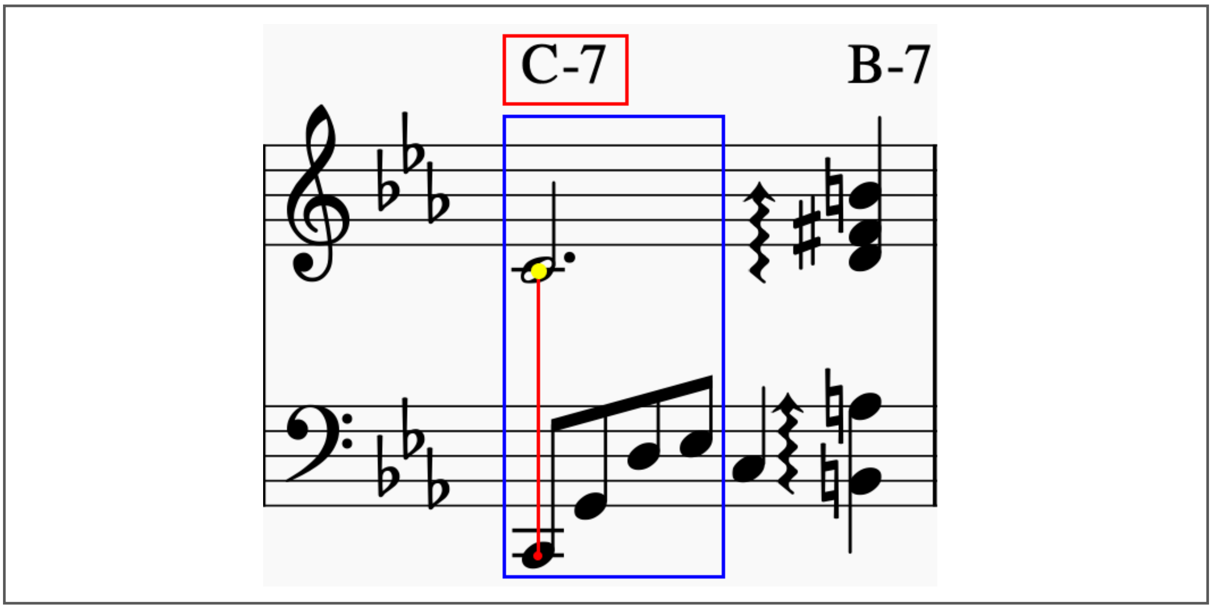


Figure 4.7 Chord voicing in which the notes are arranged in a rhythmic pattern.

In order to solve the first issue, the concept of *x-axis tolerance* was introduced to the chord extraction script. This allowed for notes that were either side of the *x*-axis coordinate to be identified as part of the chord voicing. For example, if the first note element after a harmony element had an x-location value of 100, and the x-axis tolerance was set to 10, then any note with an x-location of between 95 and 105 would be extracted as part of that chord voicing.

After extensive experimentation, it was found that an x-axis tolerance value of 20 yielded the best results. For the first issue of closely positioned notes, the x-axis tolerance ensured that all chord voicing notes were extracted whilst preventing notes either side of the chord voicing from being included. This is highlighted in Figure 4.8.

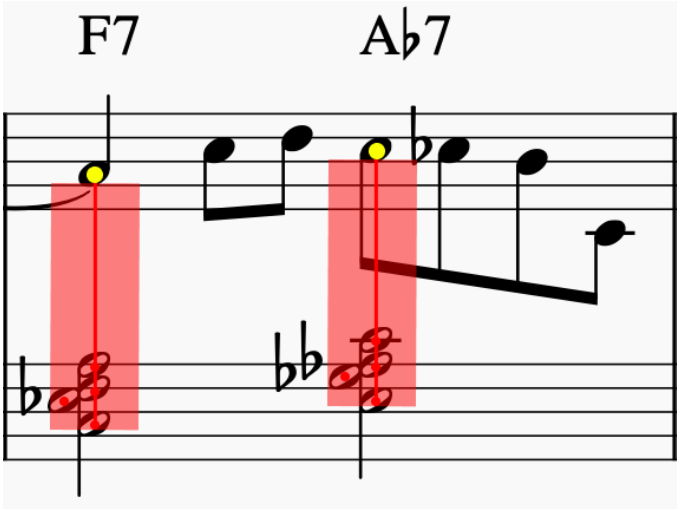


Figure 4.8 The use of *x-axis tolerance* to extracted all of the notes in a chord voicing.

It was not possible to use x-axis tolerance to successfully extract chord voicings arranged in a rhythmic pattern. For example, an x-axis tolerance value large enough to extract rhythmically arranged chord voicings would result in many unwanted notes being included when extracting vertically aligned voicings.

The first version of the script functioned as required, however as MusicXML is a structured markup language, iterating through each file line by line and using Regex to extract information seemed like bad practice. After some further research, Python’s built-in XML *ElementTree* was identified as a suitable tool, as it was able to transform XML into an ordered tree data structure. This tree structure could then be traversed in order to find and extract chord information from elements. Traversing the tree structure would also allow for search operations to be performed in worst case logarithmic time - O(log n) and best case constant time - O(1), as opposed to linear time - O(n) when iterating through *n* number of lines per file.

The second and final version of the script used ElementTree to transform each MusicXML file into an ordered tree data structure. Rather than going through each file line by line, the elements of a file were found by traversing through each measure (bar) element. Each bars children were then searched to extract each harmony element and their associated note elements to form pairs of symbols and voicings. The MusicXML tree structure can be seen in figure 4.9. The method of finding the note immediately after a harmony element and using its default-x attribute to find all of the other notes was again used in this version.

Implementing the chord extraction script using ElementTree allowed requirement 4 to be met.

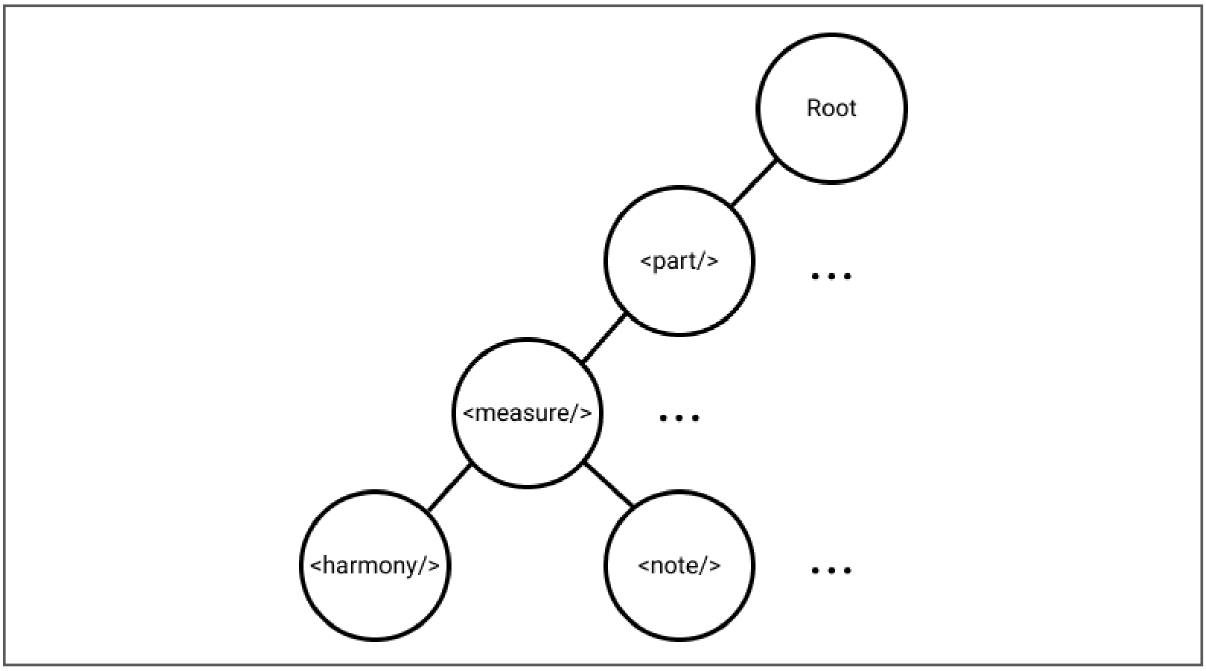


Figure 4.9 MusicXML as an Ordered Tree Data Structure

4.3.2 Chord Data Manipulation

The raw chord data output of the Chord Scraper system represents the chord symbol-chord voicing pairs exactly as they were extracted. This means that it contains errors such as incorrect chord voicings or null entries. The reason for outputting this raw data was so that other users of the system could perform their own data manipulation. However, it was also decided that the system should output a more improved and refined dataset that would be in a more "ready-to-use" state for machine learning tasks. The details of this are presented below.

After developing a functioning chord extraction script, some additional data manipulation functions were created in order to improve the output of the chord scraper. For example, as chords by definition must have 3 or more notes, the gathered chord voicings that had less than 3 notes were removed.

The chord voicings were also subjected to a small degree of pitch normalisation, in which any very high or very low notes were moved further towards the middle of the piano, for example, C8 would become C5, and C1 would become C3.

Finally, all of the chords were transposed so that their root note was C. For example, every note in a D major-seventh chord would be moved down by 2 steps (semitones). This is because the note D is 2 steps (semitones) above the note C. This is highlighted below in figure 4.10.

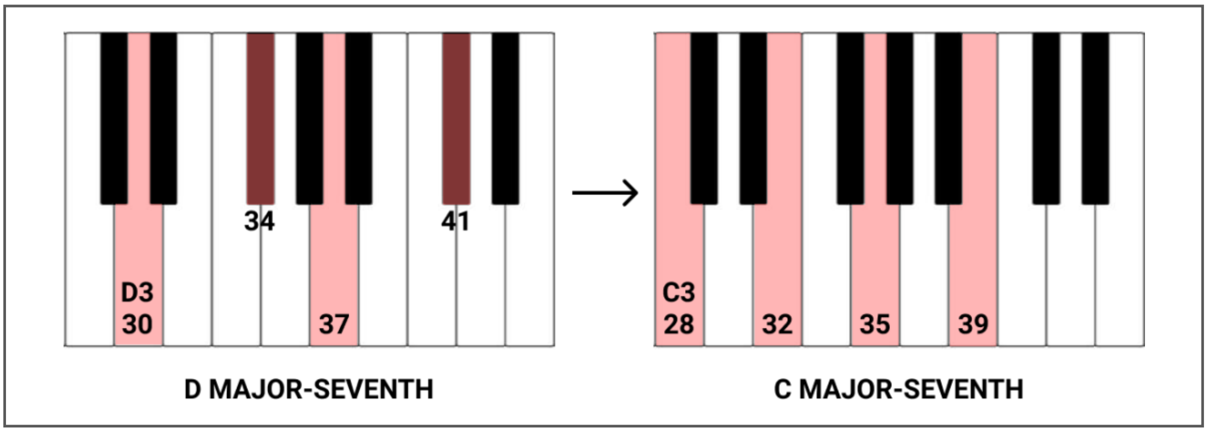
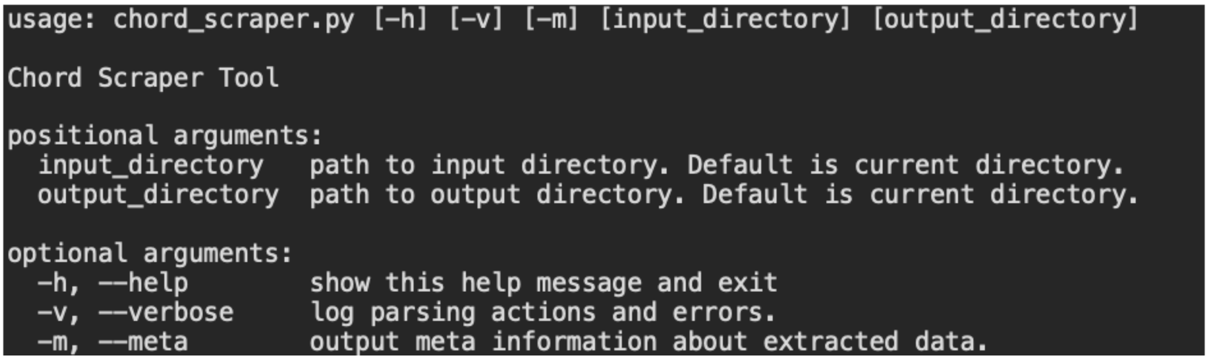


Figure 4.10 Piano keyboards showing process of transposition.

Transposing each chord symbol-chord voicing pair meant that there was no distinction between root notes, and only between chord types. For example, rather than having 200 C major-sevenths, 150 E major-sevenths, 400 F major-sevenths, and 250 Ab major-sevenths, the data would consist of 1000 C major-sevenths. Therefore, when using the dataset to train a machine learning model, there would be many more examples of each chord type. There is also no downside to this approach, as any generated chords can be transposed back to their original root note whilst keeping their voicing structure.

Initially, the two datasets were outputted as Python data pickles - serialised data objects. This was due to the fact that pickles make extracting and reimporting Python lists and dictionaries simple. As the pickle format is Python specific and other programming languages do not offer in-built support, the datasets were additionally outputted in CSV format. It must be noted however, that storing Python data structures in this format is less than ideal.

The Chord Scraper system features a simple command line interface (CLI), which was implemented using the Python library *argparse*. Users can specify input and output directory paths, as well as enable error logging and the printing of chord meta information. The CLI can be seen below in figure 4.11.



```
usage: chord_scraper.py [-h] [-v] [-m] [input_directory] [output_directory]

Chord Scraper Tool

positional arguments:
  input_directory   path to input directory. Default is current directory.
  output_directory  path to output directory. Default is current directory.

optional arguments:
  -h, --help        show this help message and exit
  -v, --verbose     log parsing actions and errors.
  -m, --meta        output meta information about extracted data.
```

Figure 4.11 Chord Scraper CLI help menu.

4.4. Results and Evaluation

In this sub-section, the Jazz-Chords dataset gathered by the Chord Scraper is presented and evaluated. The performance and effectiveness of the Chord Scraper is also evaluated.

When evaluating the Chord Scraper, it is important to make a clear distinction between the correctness of the input data versus the effectiveness of the Chord Scraper to correctly extract chords. For example, if the output dataset contains errors, it must be determined whether or not this is a cause of an incorrect input MusicXML score, or inaccurate chord extraction by the Chord Scraper.

4.4.1 Jazz-Chords Dataset

To download the dataset, please go to this link. If viewing offline see ". /chord_scraper/dataset".

In order to gather the Jazz-Chords dataset, a Chord Scraper tool was developed and given as input a library of 171 fully arranged jazz piano solos. Details of the library can be found in section 3. The Jazz-Chords dataset consists of 7404 chord symbol-chord voicing pairs. All of the chord voicings have a root of C. There are a total of 23 different chord types present in the dataset. The dataset is presented as a data pickle and in CSV format. The structure of the dataset can be seen below in table 4.1. Looking at the table it can be seen that each chord pair in the dataset has 3 keys. The *type* and *extensions* keys represent chord symbol information, and the *note_numbers* key represents the chord voicing information.

As mentioned, there are 23 different chord types present in the Jazz-Chords dataset; figure 4.12 and table 4.2 show these chord type distributions.

Key	***Data Type***	***Description***
type	String	The type of chord, i.e., “major-seventh”
extensions	List of Dictionaries	A list containing any chord symbol extensions
note_numbers	List of Integers	A list containing chord voicings as note numbers

Table 4.1 Jazz-Chords dataset structure.

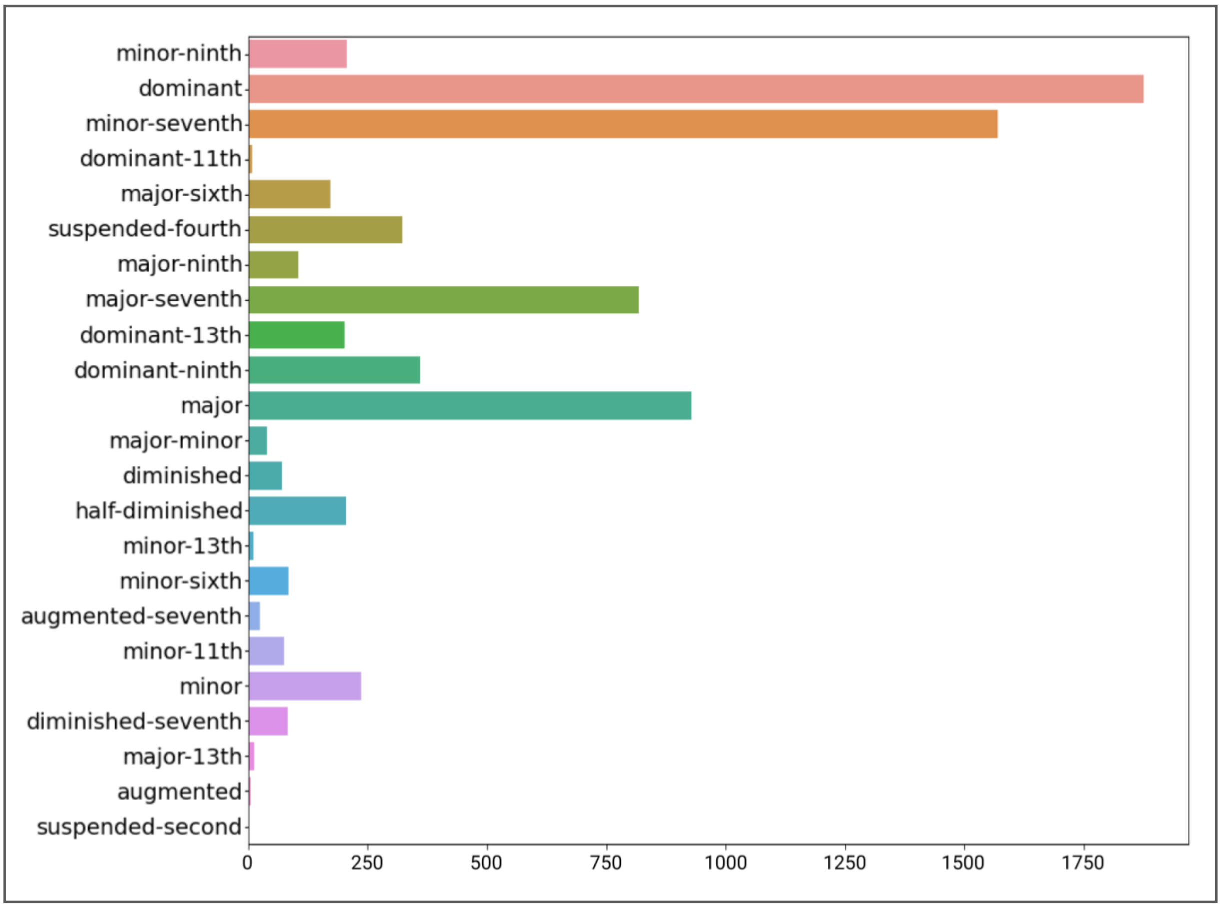


Figure 4.12 Chord type distribution of the Jazz-Chords dataset.

Chord Type	*Count*
dominant	1875
minor-seventh	1569
major	928
major-seventh	818
dominant-ninth	359
suspended-fourth	322
minor	236
minor-ninth	206
half-diminished	204
dominant-13th	202
major-sixth	171
major-ninth	104
minor-sixth	83
diminished-seventh	82
minor-11th	75
diminished	70
major-minor	39
augmented-seventh	24
major-13th	12
minor-13th	11
dominant-11th	8
augmented	4
suspended-second	2

Table 4.2 Chord type distribution of the jazz-chords dataset.

The chord distribution is typical of jazz piano solos (Eremenko et al., 2018:487). However, the issue with the datasets chord distribution is that many of the chord types do not have enough associated chord voicings to be used in machine learning tasks. This means that in practice, the dataset can arguably only facilitate the generation of 4 different chord types – dominant, minor-seventh, major, and major-seventh.

As mentioned in section 1, jazz musical theory defines a standardised set of notes that can be included in a chord voicing. For example, a C major-seventh base voicing configuration contains the following notes – C, E, G, B (Mulholland & Hojnacki, 2013:ix). This base voicing can be rearranged and spread across the piano to make it sound more musically pleasing. Chords voicings can also have additional colour-tones, which are notes that are not part of the standardised voicing but can be added to give the voicing additional colour (Mulholland & Hojnacki, 2013:1). Adding these colour tones does not change the type of the chord. Each chord type also has notes that are considered incorrect, meaning that they're presence would change the type of the chord (Mulholland & Hojnacki, 2013:1).

In order for the presented dataset to be of use, its contained chord voicings need to have a high degree of accuracy, and therefore must contain little to no incorrect notes. The 4 most common chord types were tested for the presence of unwanted notes. Table 4.3 shows the incorrect notes for each of the 4 chord types. Figures 4.13 - 4.16 show the note distribution of all of the chord voicings for each chord type. It must be noted that dominant chords have no incorrect notes, as all additional notes outside of the base voicing configuration are considered valid colour notes (Mulholland & Hojnacki, 2013:1).

Chord Type	*Incorrect Notes*
Dominant	-
Minor-seventh	Db, E, F#, Ab, B
Major	Db, Eb, F, F#, Ab, Bb
Major-seventh	Db, Eb, F, Ab, Bb

Table 4.3 Incorrect notes for listed chord types.

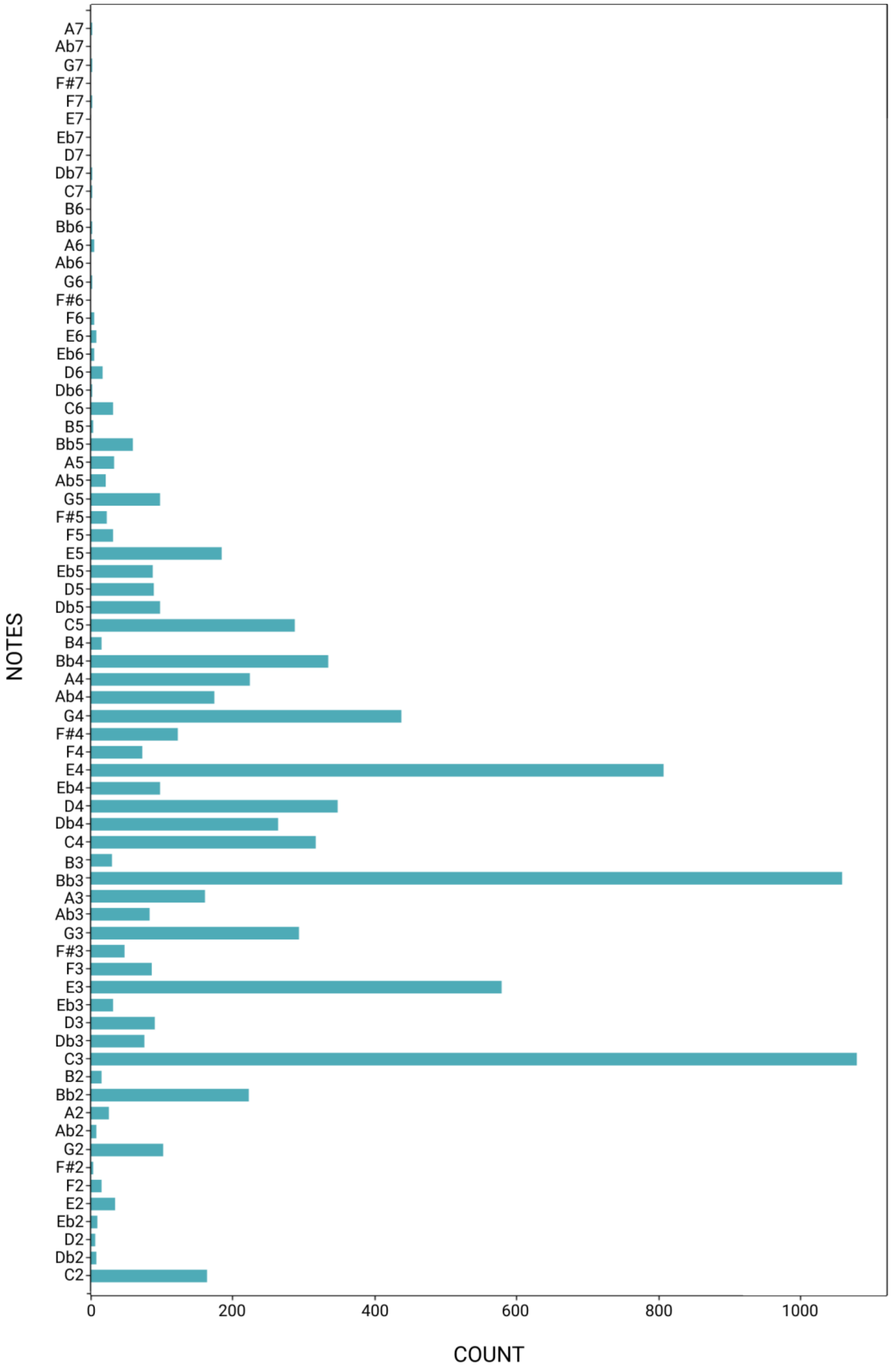


Figure 4.13 Note distribution for all dominant chord voicings in Jazz-Chords dataset.

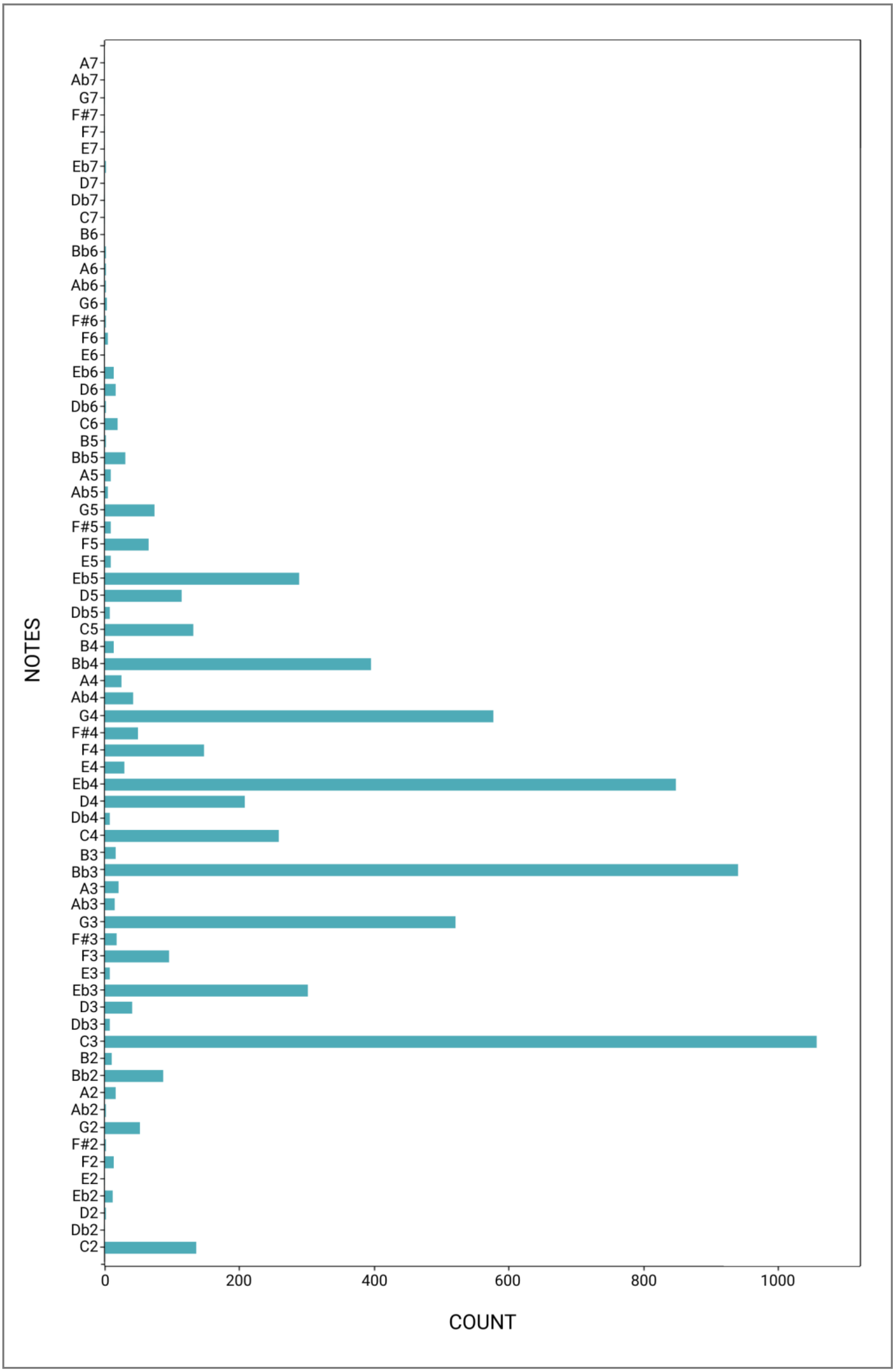


Figure 4.14 Note distribution for all minor-seventh chord voicings in Jazz-Chords dataset.

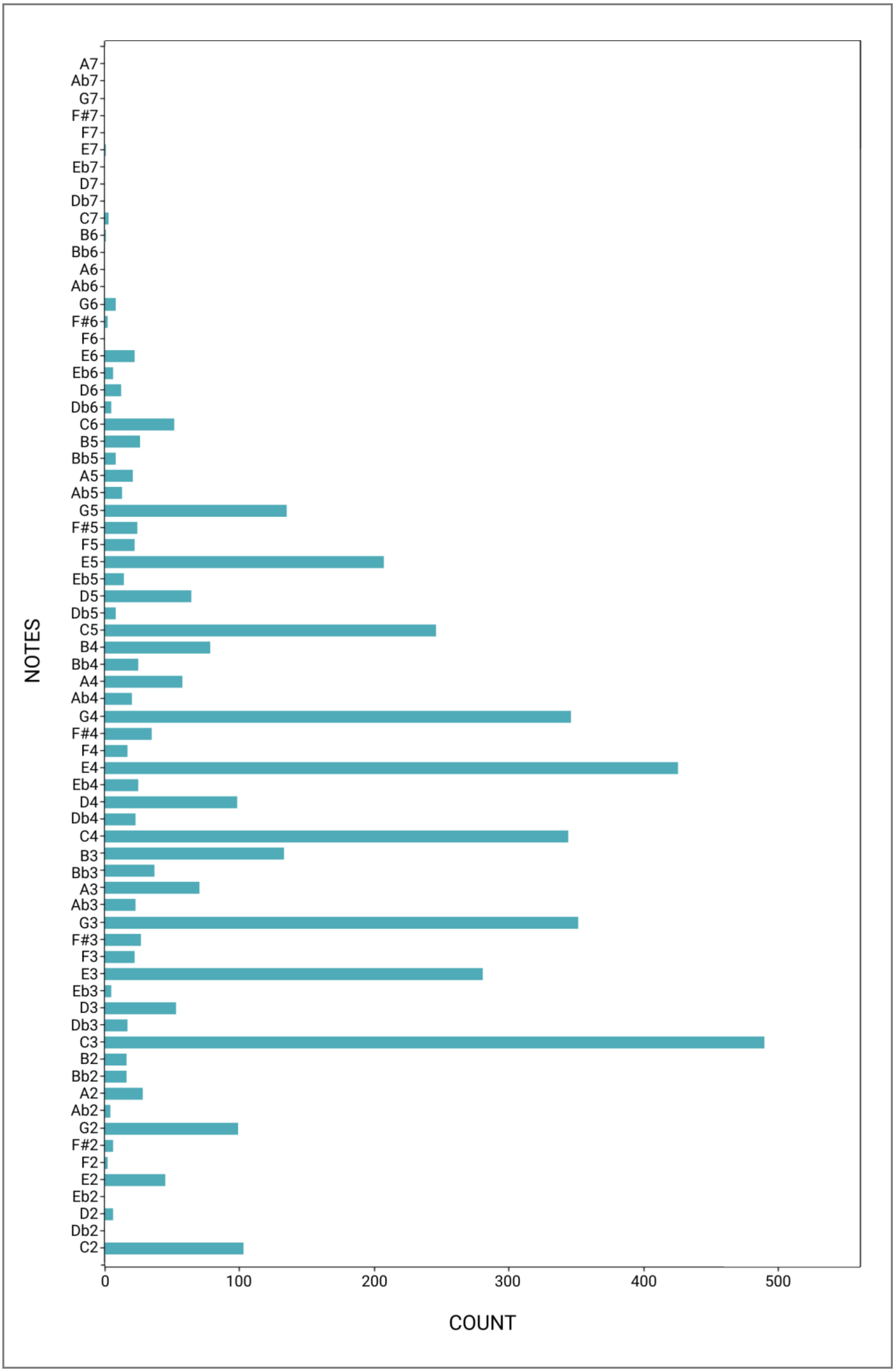


Figure 4.15 Note distribution for all major chord voicings in Jazz-Chords dataset.

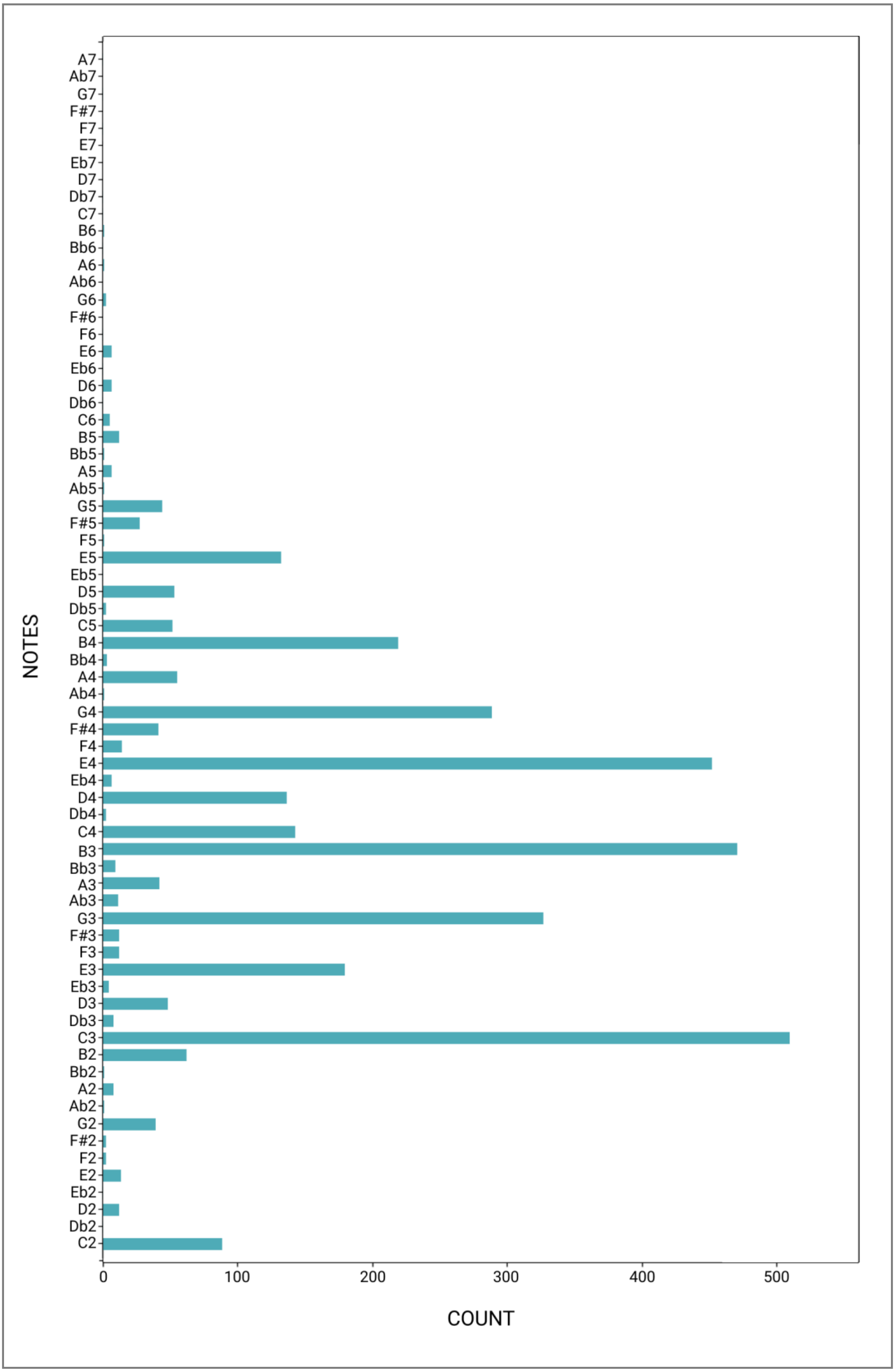


Figure 4.16 Note distribution for all major-seventh chord voicings in Jazz-Chords dataset.

By cross referencing figures 4.14 – 4.16 with the incorrect notes in table 4.3, it can be seen that minor-seventh, major, and major-seventh chord voicings all have the presence of unwanted notes. Table 4.4 presents further chord voicing accuracy information.

Chord Type	***Total Notes***	***Total Incorrect Notes***	***Accuracy***
Dominant	8419	0	1
Minor-seventh	6999	258	0.963
Major-seventh	3587	79	0.978
Major	4259	406	0.904

Table 4.4 Chord voicing accuracy of Jazz-Chords dataset.

As seen in table 4.4, aside from the major voicings, the datasets chord voicings have a high degree of accuracy. In the following section, it is determined whether the presence of unwanted notes is a result of the Chord Scrapers input being inaccurate, or a flaw in the Chord Scrapers functionality.

4.4.2 Evaluation of Chord Scraper's Performance

At the beginning of this section, the requirements of the Chord Scraper system were set out. Looking at the resulting dataset presented in the previous section, it can be concluded that, for the most part, the Chord Scraper met all of its requirements.

The Jazz-Chords dataset shows that the Chord Scraper was able to meet requirement 1, as it was able to successfully extract a large number of chord symbol-chord voicing pairs from MusicXML piano solos.

Through the successful implementation of the chord extraction and data manipulation scripts, the Chord Scraper was able to meet requirement 2, as it is able to output raw and refined chord datasets.

The system also met requirement 3, as it able to successfully scrape 166 unseen MusicXML files having been implemented using 5 test files. This suggests it will be able to successfully scrape further unseen MusicXML files.

Converting each arrangement into an ordered tree structure allowed the system to meet requirement 4, as it was able to scrape all 171 MusicXML files in around 40 seconds.

Using the argparse library to develop a simple CLI allowed the Chord Scraper to meet requirement 6.

However, the Chord Scraper failed to fully meet requirement 5, as the "major" chord type voicings failed to meet the requirement of 97% accuracy. In order to find the cause of this inaccuracy, an investigation was conducted in which extracted chord voicings were manually compared with the MusicXML file from which they were obtained.

A total of 94 major chord pairs with incorrect notes were randomly selected from the Jazz-Chords dataset for manual comparison. The results of this comparison showed that 95% of the inaccuracy was present in the MusicXML library, and thus existed before being presented to the chord scraper as input. This investigation suggests that the inaccuracy found within major chord voicings was largely a result of the inputted data, and not a result of the Chord Scraper's functionality. The high level of accuracy present in the other chord type groups also supports this suggestion. However, as the investigation was only based on a small sample of the total number of inaccurate major chords, this suggestion cannot be fully asserted. If given more time, a larger sample could be examined.

Another issue with the Chord Scraper system was that a total of 1872 chord symbol-chord voicing data pairs were removed from the raw chord data. This was due to the chord voicings having less than 3 notes. As explored in section 4.3.1, the Chord Scraper could not be designed in a way in which it could extract chord voicings which were arranged in a rhythmic pattern. It is suspected that this is the reason for so many chord voicings with less than 3 notes existing in the raw dataset. Figure 4.7 on page 17 highlights this issue, as it shows that the Chord Scraper can generally only extract 2 notes from rhythmically arranged chord voicings.

In order to confirm this assumption, a manual comparison was carried out. A total of 40 chord voicings with less than 3 notes were randomly selected from the raw dataset. These chord voicings were then manually compared to the MusicXML file from which they were obtained. The results of this comparison showed that all 40 voicings were arranged in a rhythmic pattern, and thus all notes could not be extracted by the Chord Scraper.

This investigation confirms a limitation of the designed Chord Scraper system which resulted in it not being able to successfully extract 20% of the chords present in the inputted MusicXML library. This issue has the potential to be resolved if given more time. One solution could be to find a way in which the Chord Scraper could distinguish between rhythmically and non-rhythmically arranged chord voicings. This would then allow for the *x-axis tolerance* to be adjusted accordingly.

5. Chord Generator

This section will detail the development of a conditional generative adversarial network (cGAN) with the aim of generating accurate and varied jazz chord voicings for a given chord symbol. This section will first look at how the Jazz-Chords dataset was pre-processed and encoded for use as training data for the cGAN model. The section will then present some initial deep learning experiments that served as a precursor to developing the cGAN model. Following this, the development and implementation of the cGAN will be presented. Finally, the results and capabilities of the cGAN model are presented and evaluated.

Sections 2.3 & 2.4 are recommended precursors to this section. All models were trained using a Nvidia Tesla v100 GPU.

Note that in this section, *chord symbols* are generally referred to as *chord labels*.

5.1 Objectives

1. To process and encode the Jazz-Chords dataset for use as training data for the cGAN model.

2. To experiment with different data encodings and different network architectures through carrying out some classification tasks.

3. To implement a cGAN capable of generating accurate (more than 90%) and varied (more than 20%) chord voicings for a given chord symbol.

5.2 Preparing the Training Data

Deep learning models require a large amount of data in order to be effectively trained (Sun, 2017:843). A cGAN model capable of generating jazz chord voicings needs a large dataset of chord label-chord voicing pairs. For example, for a cGAN model to generate major chord voicings, it needs to be given a large enough amount of major chord label-chord voicing pairs for a mathematical relationship between label and voicing to be learnt. As mentioned in section 2.4, cGANs are capable of generating multiple different types of output, meaning that they can be trained to generate multiple different types of chord voicings.

This subsection first presents how the Jazz-Chords dataset was pre-processed in preparation to train the cGAN model. It then details how the chord labels and voicings were encoded into data representations better suited to deep learning models.

5.2.1 Pre-processing

Looking at the chord distribution of the Jazz-Chords dataset in table 4.2 on page 22, there are only 4 chord types that have enough chord voicings to be included in the training data. These types are dominant, minor-seventh, major, and major-seventh. All other chord types were removed from the training data.

The four chord types have differing numbers of voicings, meaning that the distribution of the overall training data is imbalanced. Training the cGAN model with an imbalanced dataset could cause the model to perform inconsistently, as it would be less effective at generating chord types with lower representation (Wang et al., 2016:4368). In order to rectify this, 818 dominant, minor-seventh, and major chord pairs were randomly selected, and combined with the 818 major-seventh chord pairs to form a balanced dataset.

As presented in section 4.4.1, the dataset voicings contained a small amount of inaccuracy. These inaccuracies were removed from the dataset before it was used to train the cGAN model. In order to do so, all of the incorrect notes seen in table 4.3 on page 22 were programmatically removed from all of their associated chord type's voicings. It should be noted that dominant chords have no unwanted notes, and thus had no notes removed.

Furthermore, all of the chord voicing's note distributions were capped at the note F#4. This was so that the generator did not learn to generate chord voicings with high pitch notes that would clash with a lead sheets melody notes, which can commonly be found as low as G4. The processed training dataset is presented in table 5.1 and figures 5.1 and 5.2.

Chord Label	***Count***
dominant	818
minor-seventh	818
major	818
major-seventh	818

Table 5.1 Chord label distribution of the training dataset.

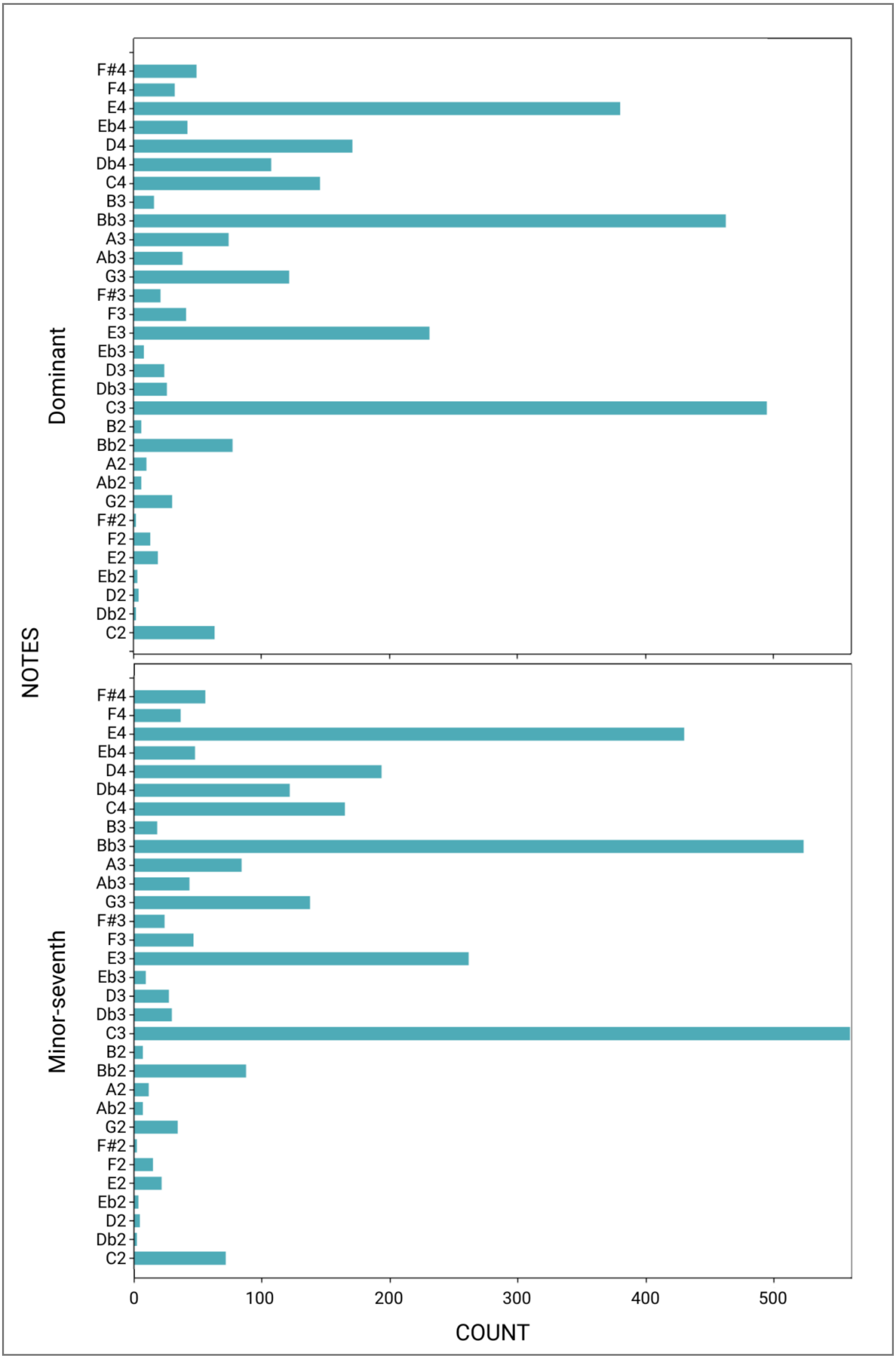


Figure 5.1 Dominant and Minor-seventh note distribution (training dataset).

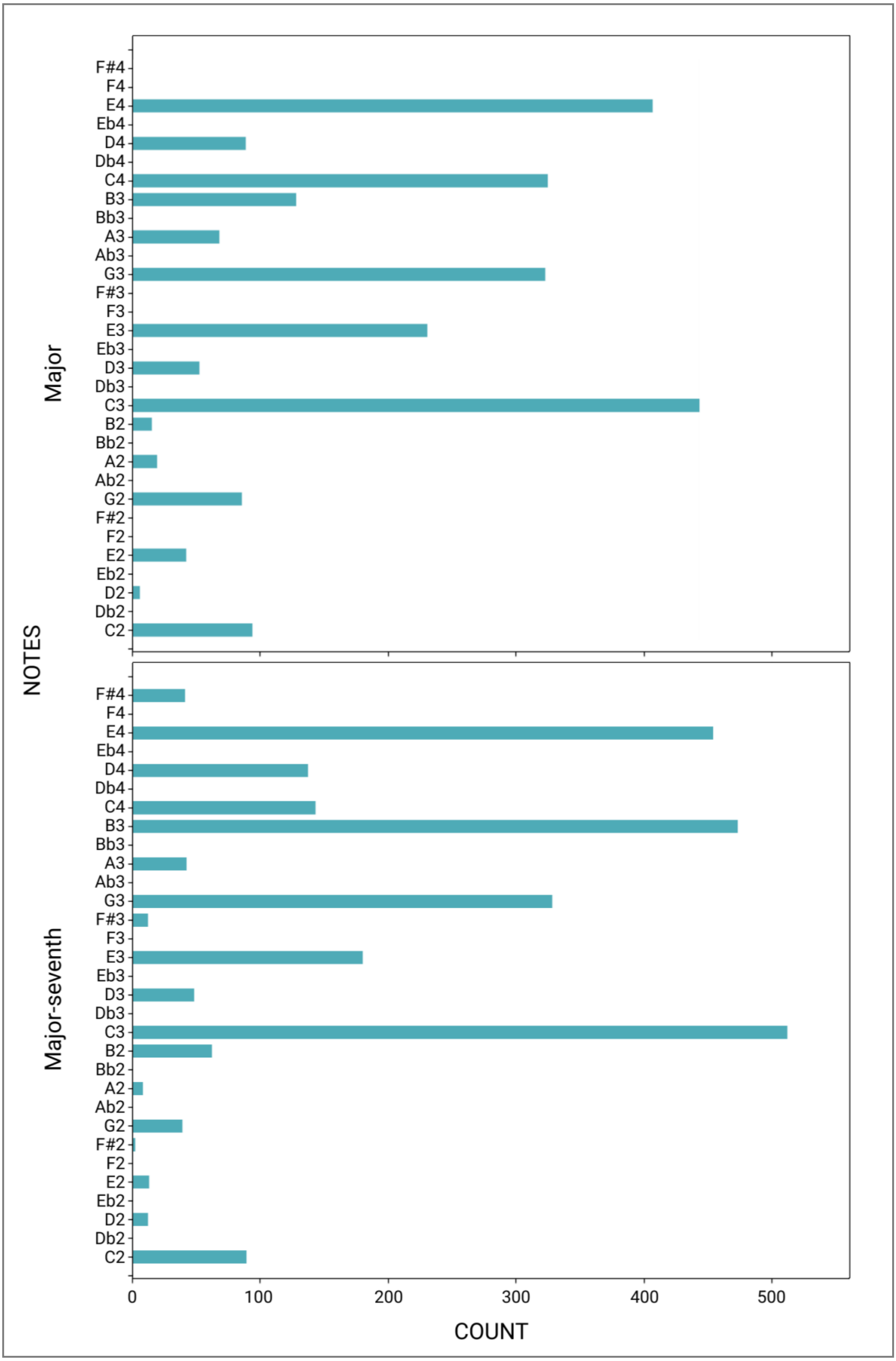


Figure 5.2 Major and major-seventh note distribution (training dataset).

By cross-referencing the training data note distributions with table 4.3 (page 22), it can be seen that there are no incorrect notes in each of the chord types note distributions. It can also be seen that there are no notes at F#4 or above.

5.2.2 Encoding

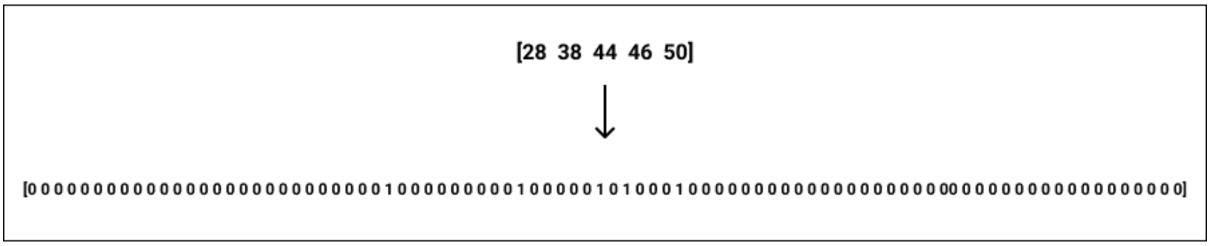
The final step in preparing the training data was to represent the chord labels and chord voicings in a format that the cGAN could better comprehend. An integer encoding was applied to the chord labels; this can be seen in table 5.2.

Chord Label	***Integer Encoding***
dominant	0
minor-seventh	1
major	2
major-seventh	3

Table 5.2 Integer encoding for chord labels.

In the Jazz-Chords dataset, the chord voicings are represented as lists of note numbers. The note numbers are between 1 and 88 and represent the notes on the piano in ascending order (left to right). In order to maximise the performance of the dataset and model, two different chord voicing encodings were tested.

The first followed the approach set out by Chen et al. (2020) in their chord-jazzification system. Each chord voicing was represented as a 1D array of 88 binary numbers, denoted as $\in \mathbb{R}^{88}$. Each binary number represents a note on the piano, and the presence of a note is known by a value of 1 as opposed to a value of 0. Figure 5.3 below shows this encoding.



[28 38 44 46 50]

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

Figure 5.3 Chord voicing encoded as an 88 binary number vector.

The second encoding approach had the intention of representing chord voicings as binary images. On a piano, there is a repeating pattern of 12 notes which is called an octave. Excluding the bottom 3 notes and top note of the piano leaves 7 full octaves. As none of the voicings within the training data contained these notes, they were discarded. Thus, the chord voicings were encoded in 7x12 matrices. Figure 5.4 on the next page shows this encoding.

$$
\begin{array}{c}
[28\ \ 38\ \ 44\ \ 46\ \ 50] \\
\downarrow \\
\begin{bmatrix}
0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&1&0&1&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0
\end{bmatrix}
\end{array}
$$

Figure 5.4 Chord voicing encoded as 12 x 7 binary number matrix.

In order to determine which of these encodings better represents the chord voicings, a classification task was conducted. This is presented in the next section.

Applying the above data-processing and encoding steps in line with machine learning best practices meant that the first objective of this section was achieved.

5.3 Classification Task

As explained in section 2.3 and 2.4, cGANs consist of 2 independent neural networks. The architecture of these networks depends on the type of data that is being generated. If the cGAN is tasked with generating 1D chord vectors, then Deep Feedforward Networks (DFN) can be effective for both the generator and discriminator. DFNs are “vanilla” neural networks which consist of a high number of interconnected nodes (neurons) that work to collectively learn from their input in order to optimise the final output (O'Shea & Nash, 2015:1). Each of these neurons has a weight and a bias, and through training, the values of these parameters self-adjust in order to achieve the desired outputs for given inputs.

If a cGAN is tasked with generating 2D or 3D matrices such as images, then convolutional neural networks (CNNs) are generally more effective architectures for both the generator and discriminator. CNNs are analogous to traditional DFNS, however they have additional inner layers that apply convolutions in order to extract image specific patterns such as edges (O'Shea & Nash, 2015:3). These patterns can be used to learn unobservable mathematical relationships.

As presented in the above section, the chord voicings were encoded as both 1D vectors and as 2D binary images. It is unknown which of these encodings better represents the ground truth of the chord voicings. In order to determine this, both data encodings were used in a classification task experiment. Classification network architectures are much simpler than cGANs, and offer a simple but effective way to test the performance of the differing chord voicing encodings.

To test the 1D vector encoding, a DFN classification network was constructed. To test the binary image encoding, a CNN classification network was constructed.

All attempts will be made to optimise both networks equally.

5.3.1 Deep Feedforward Network (DFN)

The initial DFN network configuration was taken from Brownlee’s (2016) machine learning textbook. The input and output layers were then altered to fit the data.

The adapted model took as input a 1D array of length 88 and outputted a 1D array of length 4. The model had one densely connected hidden layer between the input and output layers. The value of each of the output layer neurons indicated a chord label prediction. For example, an output of [1, 0, 0, 0] would mean that the model predicted the input voicing as having a dominant chord label. This is highlighted in figure 5.5 below.

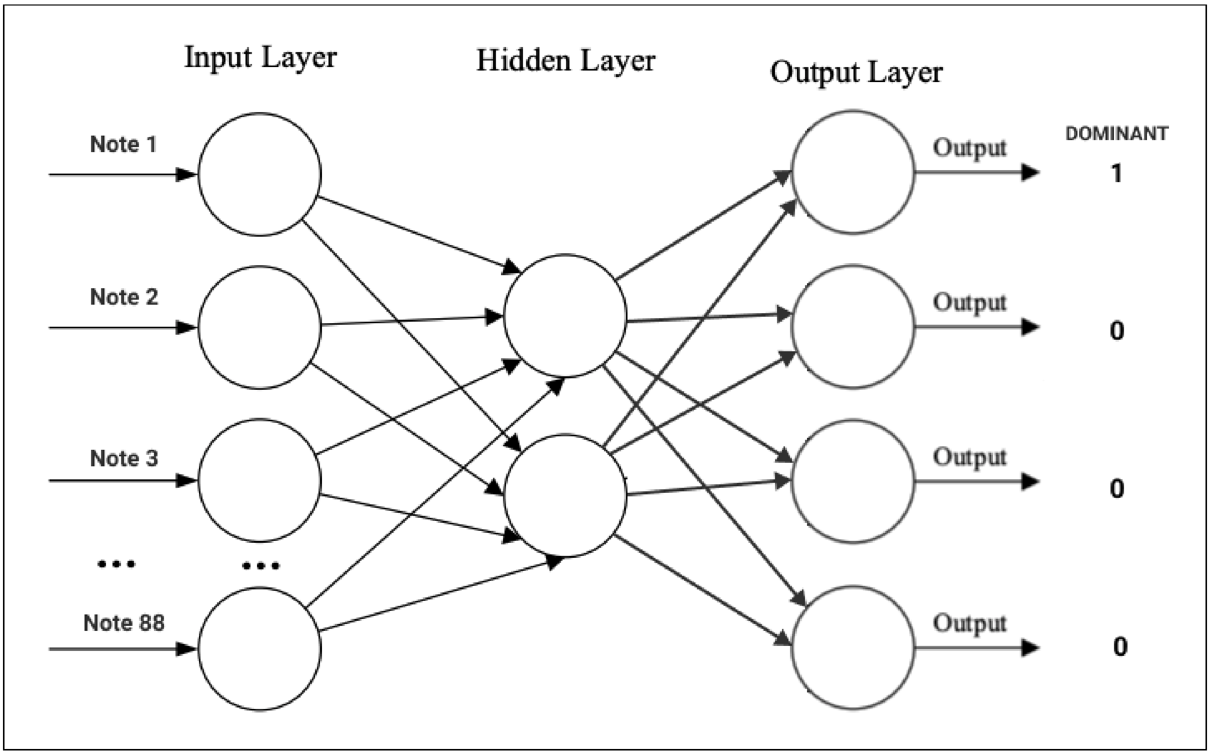


Figure 5.5 DFN architecture showing 88 input neurons, a hidden dense layer, and 4 output neurons.

Before attempting to optimise the model, an initial training pass was executed. To train the model, two thirds of the dataset were given to the model over a number of training iterations or epochs. During training, the network moves each neuron's weight and bias in a direction that reduces its loss when making chord label predictions. The loss can be defined as the measurable discrepancy between an incorrect chord label prediction and the correct label. Throughout training the model reduces this discrepancy in order to become better at making predictions.

The remaining one third of the dataset was then used to test the model's accuracy in making predictions. As this data has never been seen by the model, it gives a true indication on how well the model is performing.

The initial model had a training set accuracy of 65% and a test set accuracy of 60%, meaning that it was incorrectly labelling 4 out of every 10 input chord voicings. This accuracy seemed unusually low. Further research suggested that this issue was more than likely caused by an issue within the training data (Brownlee, 2016). Common issues such as uncleaned or unbalanced data could be ruled out. However, a chapter in Goodfellow's (2014) book suggested that low accuracy in classification networks could be due to two or more chord voicing types having overlapping distributions, meaning that the network would struggle to tell them apart.

After looking at the note distributions of the 4 chord types in the training data (figures 5.1 & 5.2, page 31 & 32), it could be seen that the major-seventh and major chord voicings were very similar.

Figure 5.6 below shows the note distribution similarity between major and major-seventh chords in the data set.

In order to resolve this issue, major chords were removed from the training dataset. This decision was justified by the fact that major-seventh chords are much more frequent in jazz piano lead sheets. The updated chord labels can be seen in table 5.3.

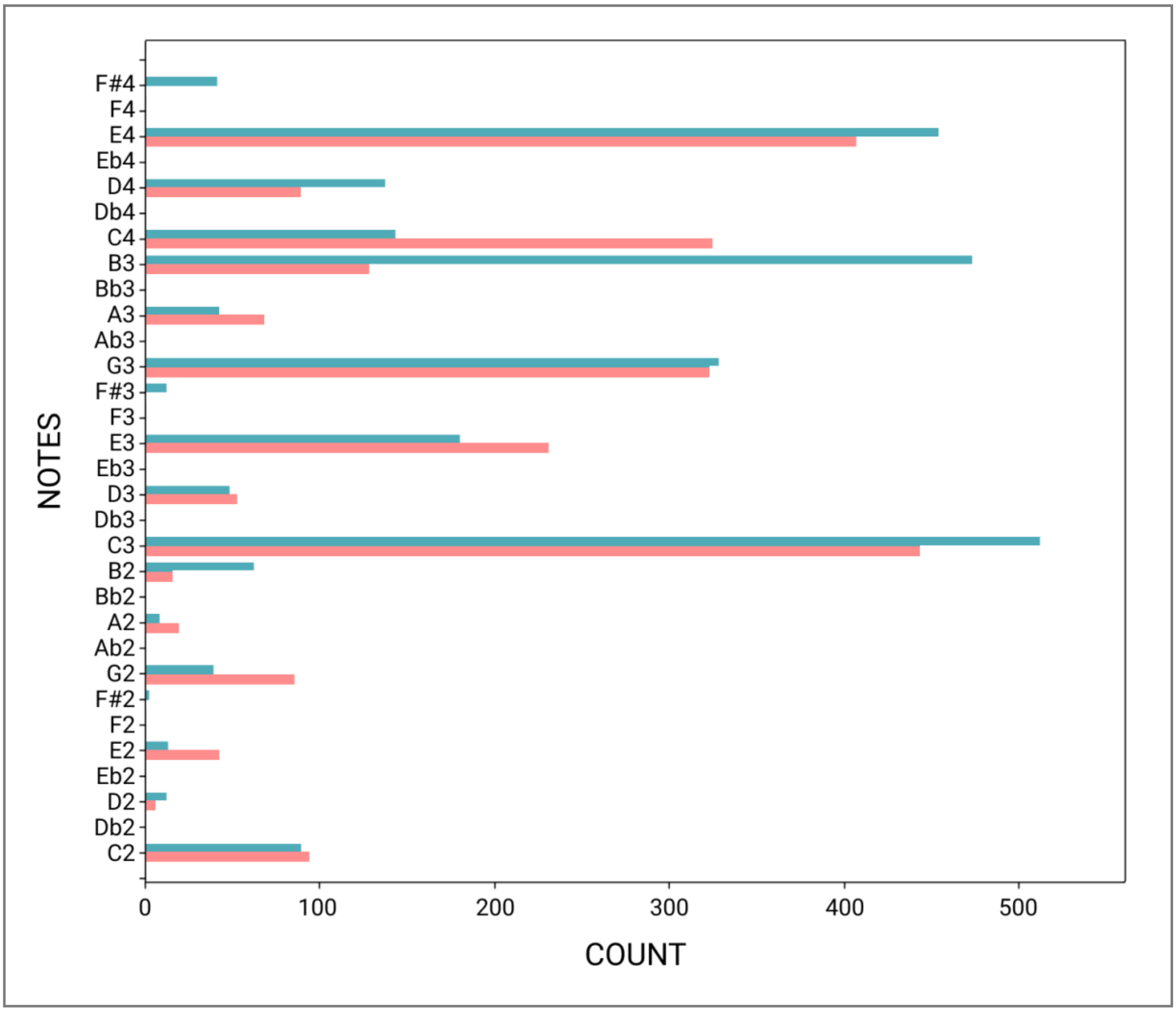


Figure 5.6 Major (red) vs. major-seventh (blue) note distribution.

Chord Label	***Integer Encoding***
dominant	0
minor-seventh	1
major-seventh	2

Table 5.3 Updated table showing integer encoding for chord labels.

The removal of major chord voicings improved the models training and test set accuracy to 81% and 70% respectively. However, a significantly poorer test set accuracy suggested that the model was overfitting, meaning that its parameters were too specific to the training set and not generalising the relationship between label and chord voicing. To combat the issue of overfitting, a series of dropout layers were added to the model. Dropout is a regularization technique that neutralises different parts of the network throughout training

(Goodfellow et al., 2016:258). This prevents an overreliance on specific areas of the network and leads to a more balanced network that has a higher likelihood of generalising.

Following this, further optimisation was carried out in order to increase the accuracy of test set predictions. To find the ideal network architecture, Goodfellows (2016) principle of optimising through experimentation whilst monitoring the test set accuracy was followed.

To do this, additional dense layers were incrementally added to the network whilst monitoring the test set accuracy. As outlined soon, the addition of more trainable neurons resulted in the performance of the network increasing. The resulting optimised DFN's architecture is presented in figure 5.7. Note that each box represents a layer in the network.

Looking at the figure, it can be seen that the DFN has an input layer that takes a 1D 88-note vector (denoted by None, 88), and an output layer with 3 neurons (denoted by None,3) which indicate one of the three chord label predictions. The inner layers are a combination of dense layers and dropout layers. It is the weight and biases of these inner dense layers that learn the mathematical relationship between chord voicing and label.

Figures 5.8 and 5.9 show the accuracy and loss curves of training the network for 300 epochs. As mentioned previously, the network loss is the sum of the errors made on each chord voicing prediction.

A regularisation principle called "early-stopping" suggests that training should be limited to the point at which the accuracy stops improving and/or the test data loss starts increasing, as these two points indicate the beginning of the model overfitting (Goodfellow et al., 2016:246). Therefore, the final iteration of the model was trained for a total of 45 epochs. The results can be seen in table 5.4.

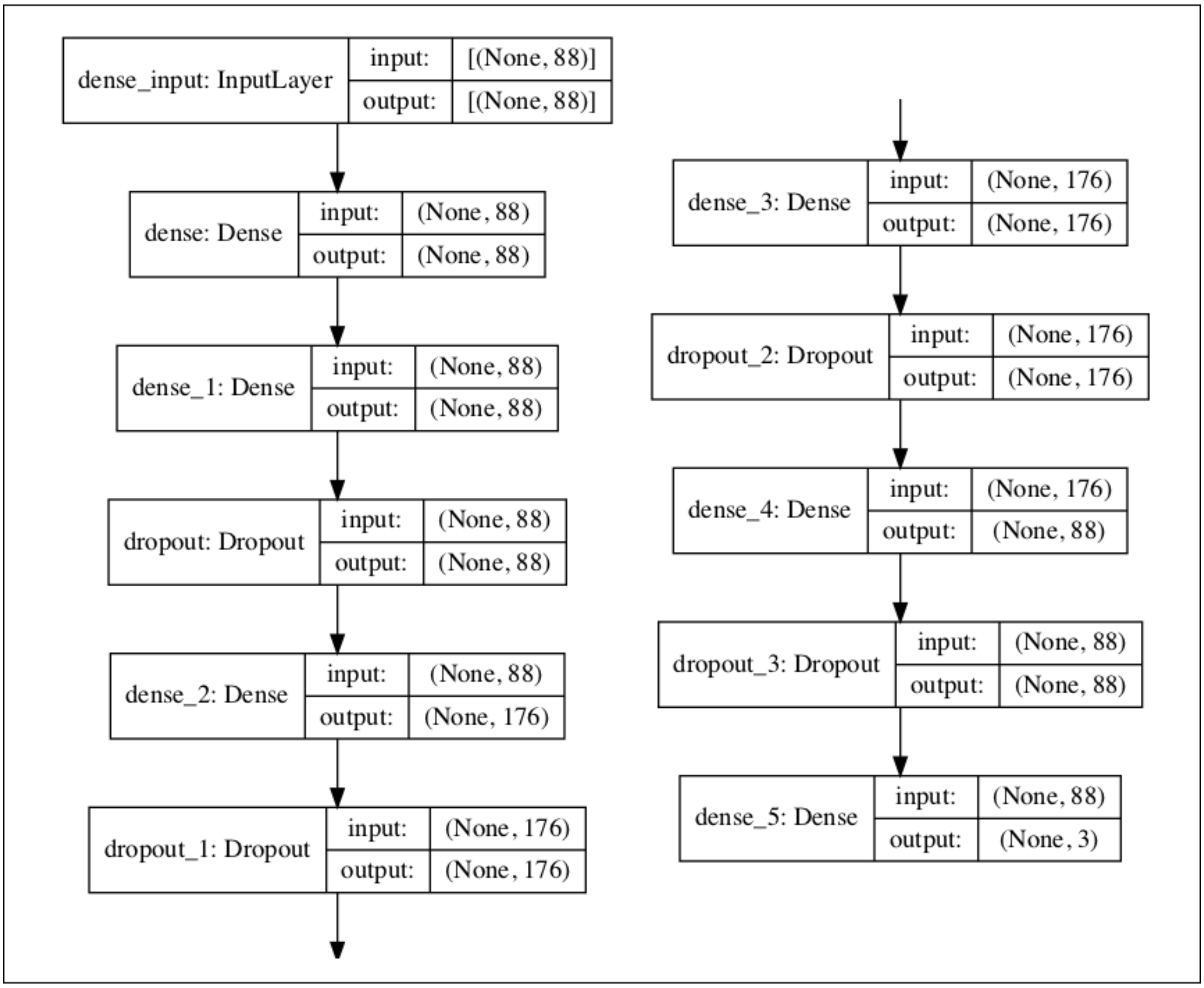


Figure 5.7 Optimised DFN architecture.

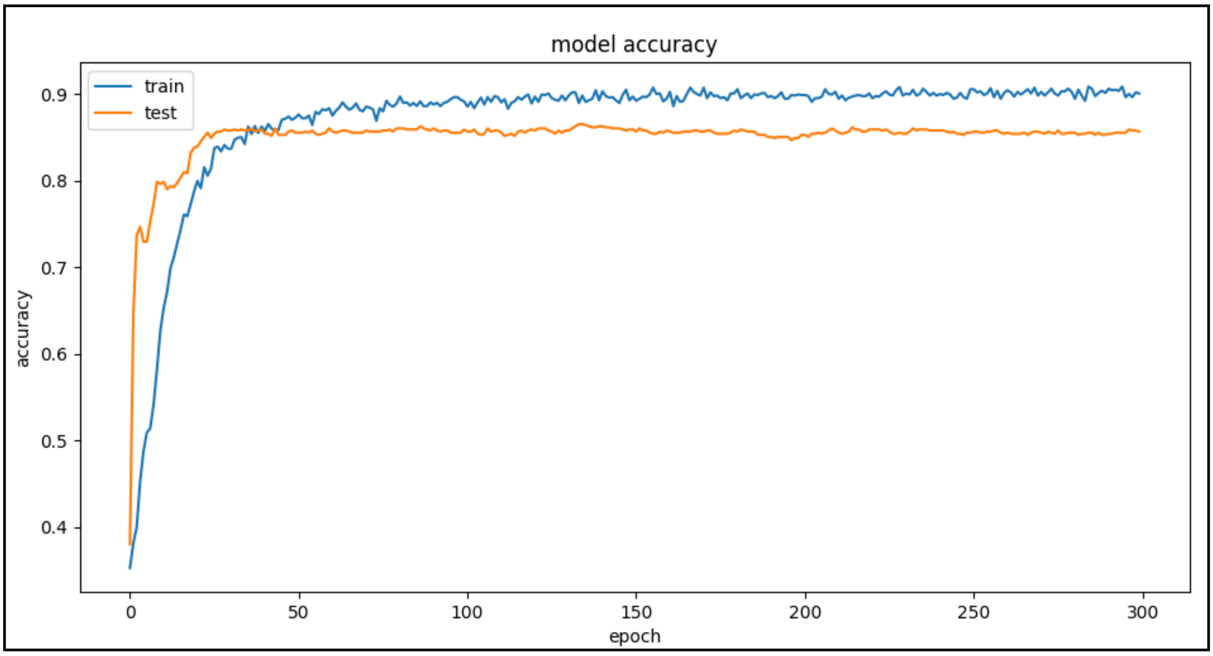


Figure 5.8 DFN accuracy after training for 300 epochs.

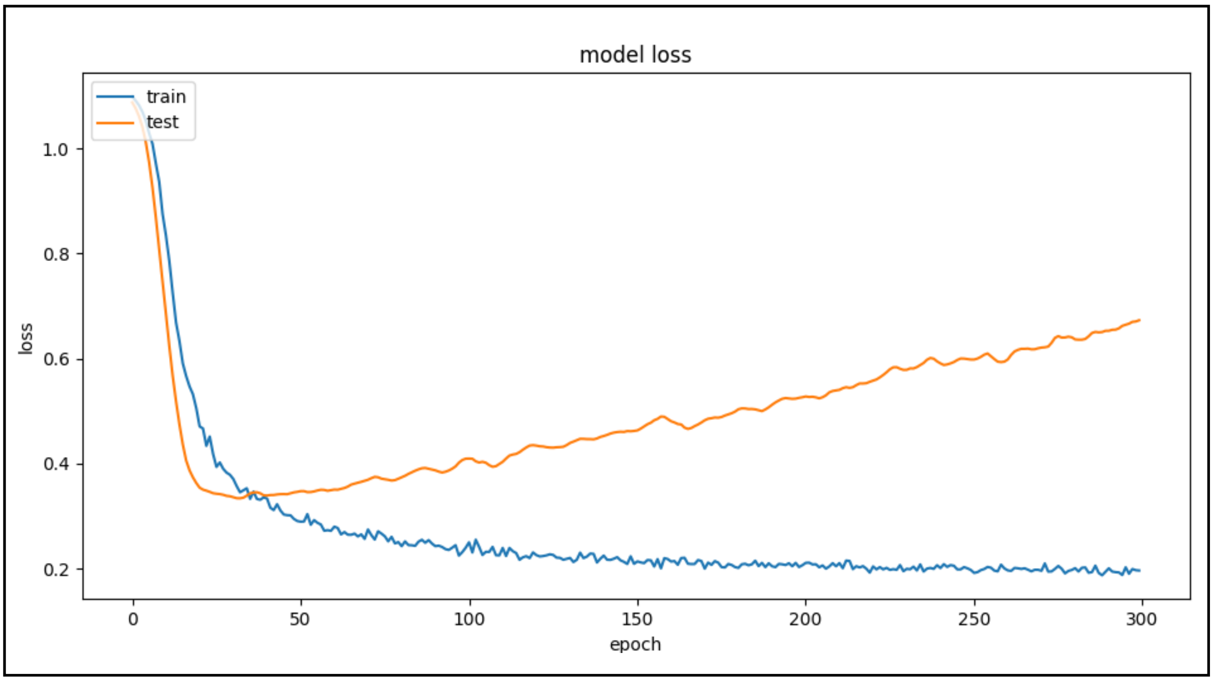


Figure 5.9 DFN loss after training for 300 epochs

Test Accuracy	0.8617
Training Accuracy	0.8631
Test Loss	0.3435
Training Loss	0.3280

Table 5.4 DFN performance after 45 training epochs.

This experiment provided insight into the performance of representing chord voicings as 88-note binary number vectors. The experiment also helped to develop a DFN architecture that can be used for the discriminator of the cGAN if needed.

5.3.2 Convolutional Neural Network (CNN)

As mentioned in the previous section, CNNs are used for machine learning tasks that involve images.

The majority of previous research in the field of music and deep generation represents chord voicings as 1D arrays. However, the presence of a repeating pattern (octave) on the piano makes the encoding of chord voicings as 2D arrays seem intuitive. The result of this encoding is essentially a binary image; of which some examples can be seen in figure 5.10 below. Please refer back to figure 5.4 (page 34) for reference on how these images are encoded. By creating and optimising a CNN and performing a classification task, it will be determined whether or not this 2D encoding more meaningfully represents chord voicings than its 1D alternative.

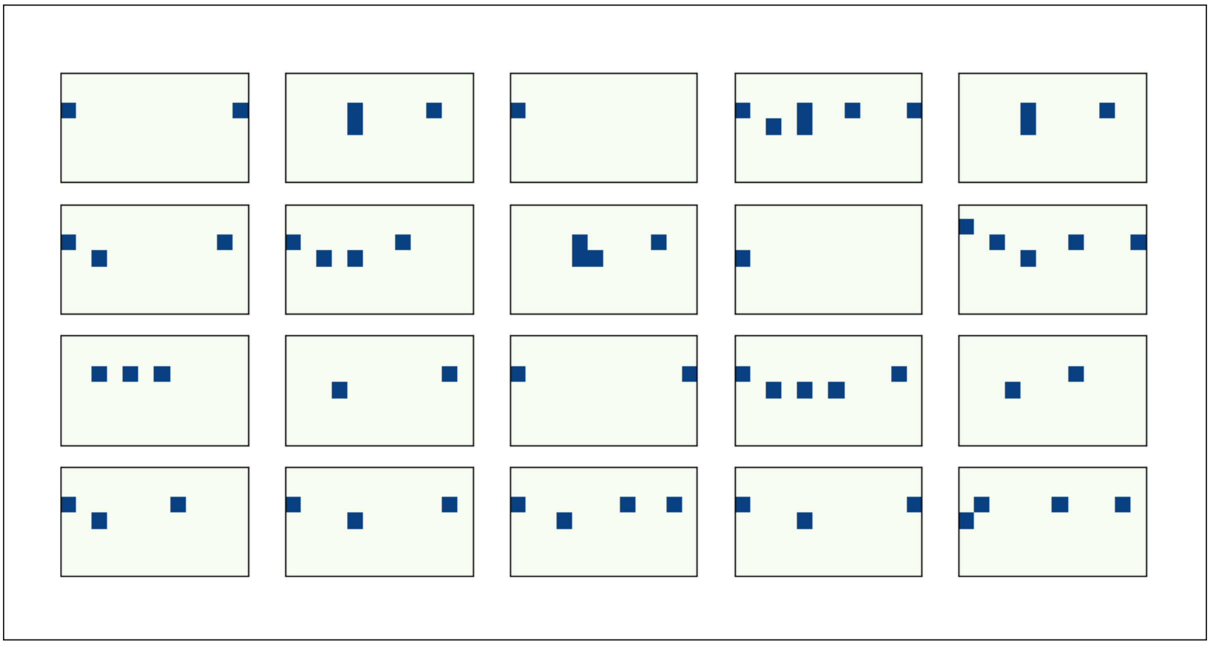


Figure 5.10 Chord voicings represented as binary images.

The initial CNN architecture was based on AlexNet, a model proposed by Krizhevsky et al. (2012). The model is highly effective at image classification tasks. It is largely made up of pairs of convolutional and max pooling layers. The convolutional layers aim to expose patterns, and the max pooling layers aim to accentuate those patterns by maximising the highest pixel values within those patterns (Krizhevsky et al., 2012:4).

The first adaptation of the AlexNet CNN model was to adjust the input and output layers to fit the shape of the data. This is shown in figure 5.11, in which the first layer has a 2D input shape of 7x12 (denoted by None, 7, 12), and the last layer has a 1D output shape of 3 (denoted by None, 3).

As with the DFN described above, the adapted CNN model was optimised through experimentation whilst observing the test set accuracy (Goodfellow et al, 2016).

This involved adding a dropout layer after each max pooling layer in order to combat a degree of overfitting that was found.

The optimised CNN model is presented in figure 5.11, and figures 5.12 & 5.13 show the performance of the CNN after training for 300 epochs.

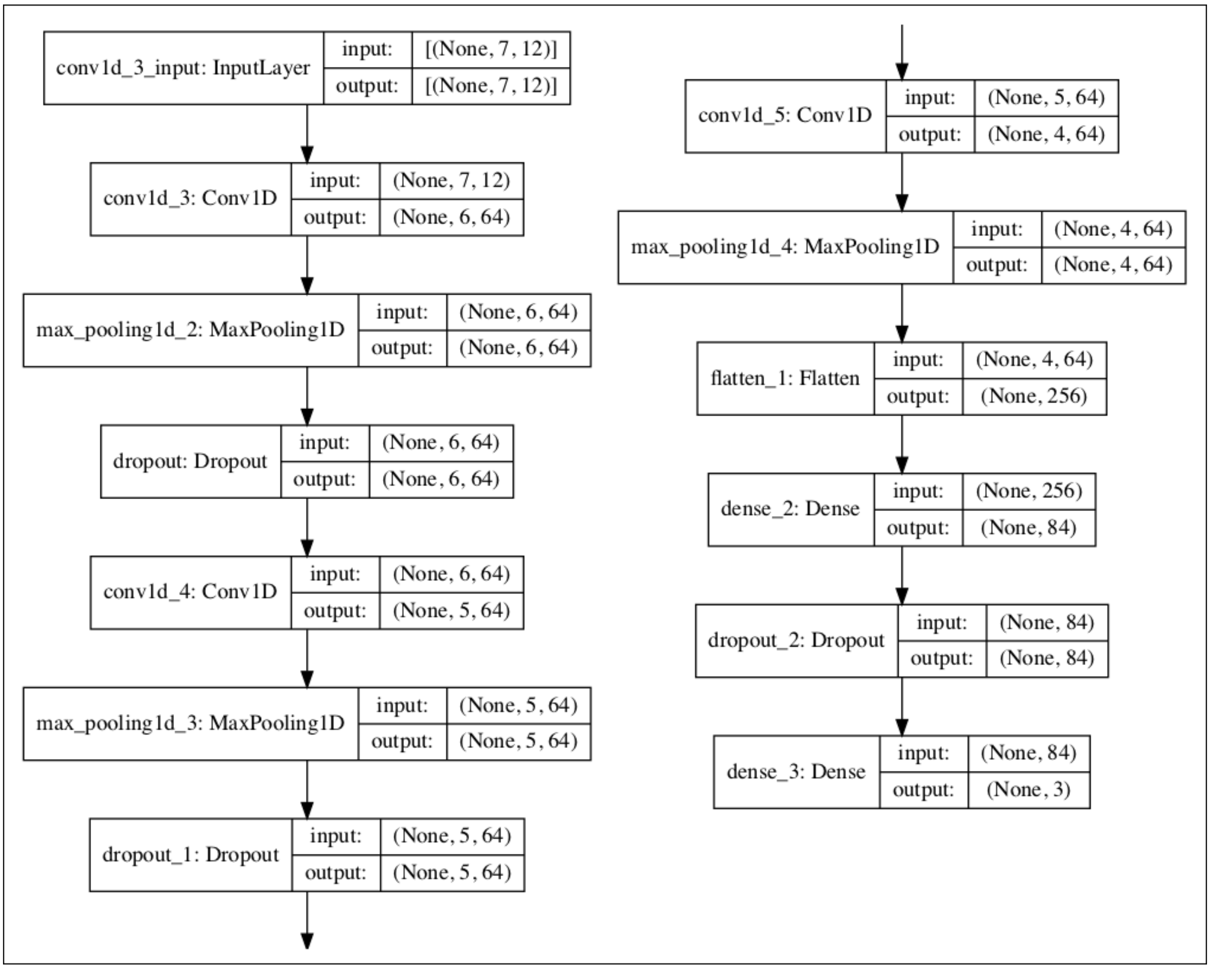


Figure 5.11 Optimised CNN architecture.

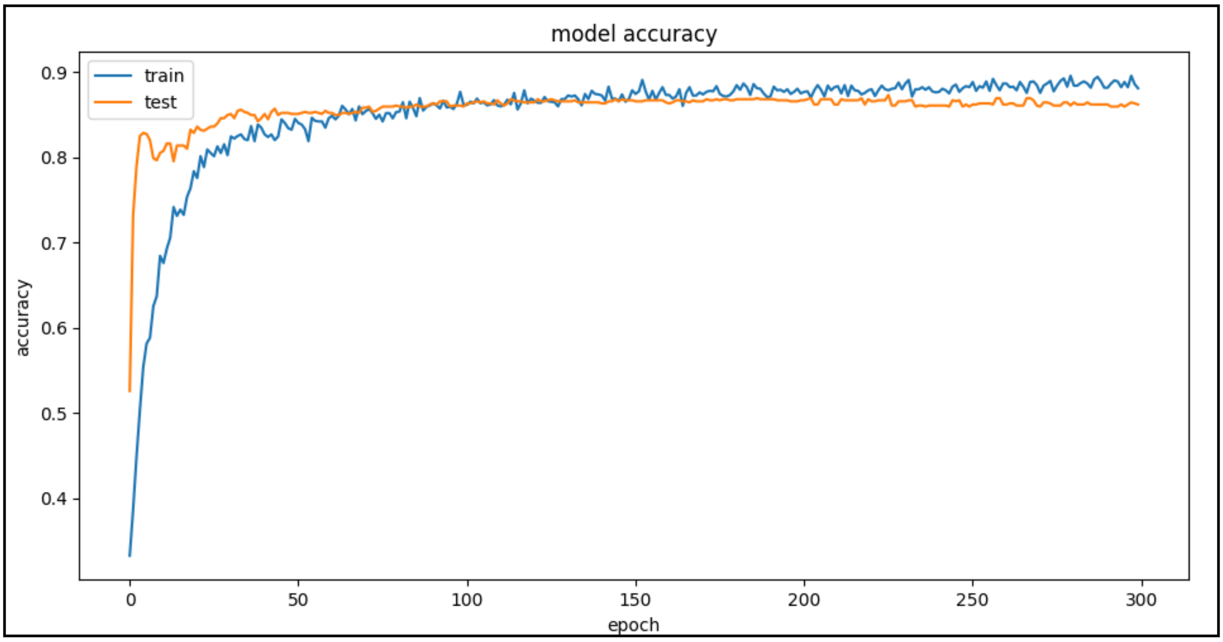


Figure 5.12 CNN accuracy after training for 300 epochs.

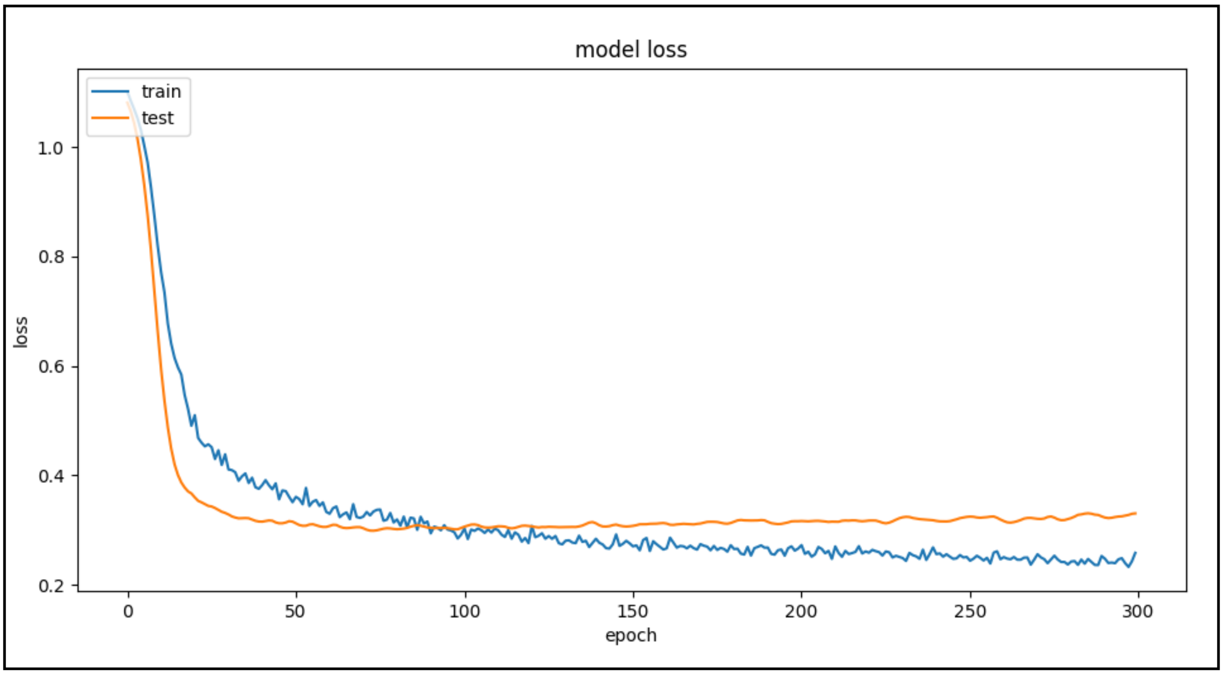


Figure 5.13 CNN loss after training for 300 epochs.

Looking at the performance metrics, the network is optimally trained at around 200 epochs, after which the test set accuracy shows no improvement. The total losses also show no sign of reconverging, indicating that the model will continue overfitting to the training data. The performance of the CNN model after 199 epochs is presented in table 5.5.

Test Accuracy	0.8679
Training Accuracy	0.8771
Test Loss	0.3128
Training Loss	0.2653

Table 5.5 Performance of the CNN model after training for 199 epochs.

5.3.3 Results

The purpose of running the above classification task experiments was to determine the optimal chord voicing embedding and network architecture for the cGAN model. Looking at table 5.6 it can be seen that the binary image encoding and CNN architecture performed best. The results suggest that representing chord voicings as binary images embeds a meaning closer to the ground truth, therefore that approach to encoding was used in the cGAN model along with a CNN architecture.

Performing the above classification tasks using different chord voicing encodings allowed for the second objective of this section to be met.

Metric	***1D Vectors + DFN***		***Binary Images + CNN***	
	Result	***Vs. CNN***	***Result***	***Vs. DFN***
Test Accuracy	0.8617	-0.0062	0.8679	+ 0.0062
Test Loss	0.3435	-0.0307	0.3128	+0.0307

Table 5.6 DFN vs CNN classification performance.

5.4 Chord Voicing Generation

As described in section 2.4, cGANs can be conditioned to generate outputs on a given input label. In the case of chord voicing generation, when given a chord label, a chord voicing that represents that label will be generated. In order for the generated chord voicings to be varied, a noise distribution is also passed into the generator model alongside the chord labels. This means that points within that noise distribution will be conditioned on a particular chord label. By keeping the chord label fixed, and varying the point in the noise distribution, varying chord voicings representing that label will be generated.

In this subsection, the process of implementing and optimizing the cGAN model will be described. Then, the results and capability of the developed cGAN will be presented and evaluated.

5.4.1 Initial cGAN Model

The model used to generate chord voicings was based on a class of CNNs called deep convolutional generative adversarial networks (DCGANs), first proposed by Radford et al. (2015). DCGANs use CNNs for both the generator and discriminator networks. Brownlee (2019:360) presents a working conditional DCGAN (cDCGAN) implementation built using TensorFlow; the source code presented was copied and used as a basis for this projects model.

Brownlee’s cDCGAN model was built to conditionally generate images of 10 different items of clothing. The model was trained using the MNIST fashion dataset (Xiao et al., 2017), which is made up of 28 x 28 grayscale images. The 10 different types of clothing had integer labels from 0-9. Figure 5.14 shows some examples of items from the MNIST dataset, as well as some conditionally generated items.

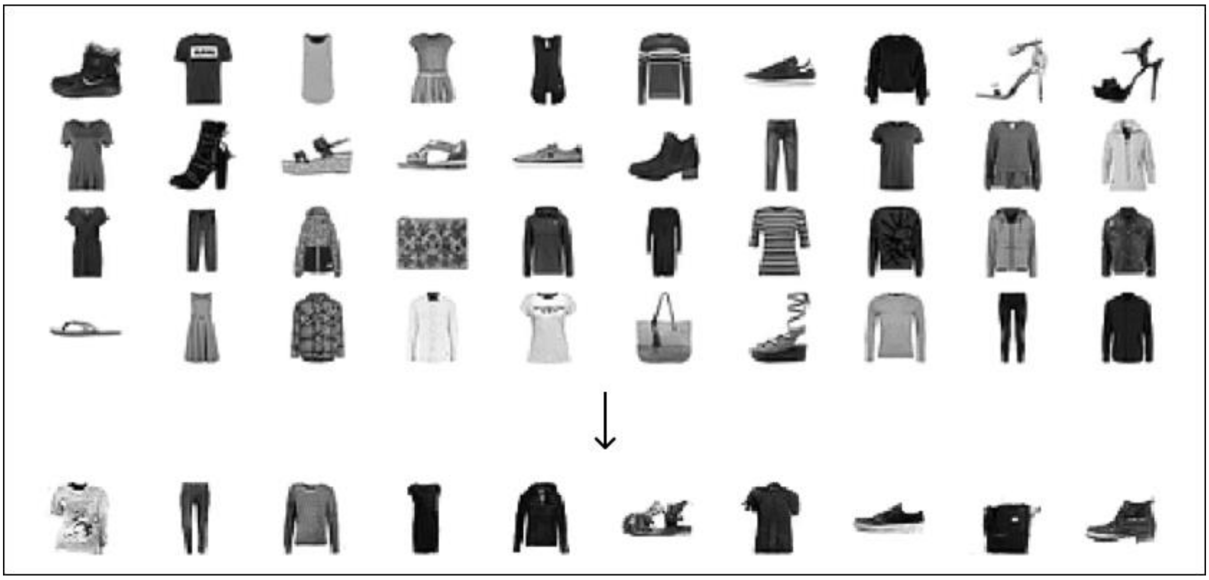


Figure 5.14 Real (top) and generated (bottom) images from the MNIST cDCGAN (Brownlee, 2019) (adapted image).

5.4.2 Adapting the cDCGAN Model

The initial phase in adapting Brownlee’s cDCGAN model was to adjust the input and output layers to fit the shape of the data. The number of different label inputs to the generator was changed from 10 to 3, in line with the dataset which has 3 different chord type labels. The input of the discriminator was also changed to take 2D images which were 7x12 in shape.

The second adaption was to adjust the inner layers of the generator so that it outputted binary images that were 7x12 matrices, as opposed to 28x28 matrices. This was done by reducing the amount of up-sampling that was applied in the inner convolutional layers of the generator. This change allowed the generator to produce chord voicing images of the same size as the training data

5.4.3 Optimising the cDCGAN Model

Using the adapted model, some initial training runs were performed using the following hyper parameters:

Optimizer: Adam algorithm with learning rate = 0.0002
Loss function: Binary cross-entropy
Number of epochs: 250
Batch size: 20

After running several training runs, two issues arose. Firstly, the loss of the discriminator with respect to generated images quickly converged to 0 and stayed there. This indicated that the generator was producing very poor chord voicing images that were not fooling the discriminator. It also indicated that the generator was not improving over time.

Following guidance from Goodfellow’s original GAN paper, the generator network was further optimised in an attempt to improve its performance (Goodfellow et al., 2014). This was done by changing the configuration of the inner convolutional layers of the generator whilst monitoring the discriminators loss with respect to generated voicing images. The discriminator network was also made less performant in an attempt to improve the performance of the generator. However, despite these changes, the generators performance remained poor.

Further research suggested that the issue could be a result of the training data (Goodfellow et al, 2016). More specifically, the chord voicing images could contain additional noise that would confuse the discriminator and prevent the generator from improving.

After looking at the training data, one issue that was identified was the fact that all of the notes of the chord voicings were distributed within only 3 of the 7 rows of the 7 x 12 matrices. This can be observed in figure 5.10 on page 39, in which only 3 of the 7-pixel rows have notes present. It was assumed that these empty matrix rows were the cause of additional noise within the training data.

To test this assumption, each label's 888 chord voicings were run through a function that removed the top row and last 3 rows, resulting in binary chord images that were of shape 3x12. This process is visualised in figure 5.15.

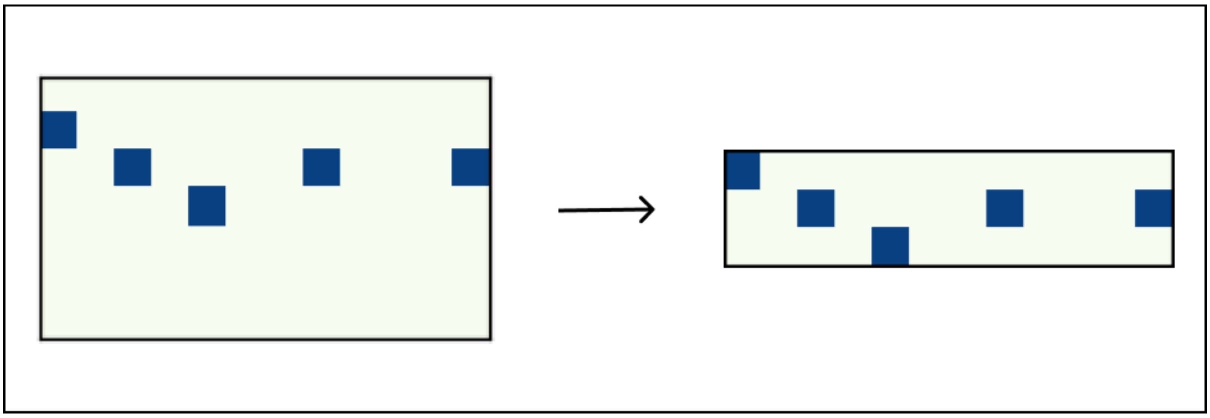


Figure 5.15 Transforming chord binary images from 7x12 matrices to 3x12 matrices.

After changing the shape of the chord voicings, the generator was updated to generate images of size 3x12. The input of the discriminator was also adjusted accordingly.

This reduction in image shape greatly improved the performance of the adapted cDCGAN model, with the discriminator loss in respect to generated images stabilising at around 0.2 after 200 epochs. This meant that the trained generator model was now able to fool the discriminator into thinking that every 2 out of 10 chord voicings it generated were from the real dataset.

However, when inspecting the generated chord voicings of the trained model, it was seen that more than 50% of the voicings looked as though they were from the training dataset. This suggested that the loss of the discriminator with respect to generated images did not properly indicate the real performance of the generator. The ineffectiveness of using discriminator losses to measure the performance of GANs is well documented within the current literature (Lucas et al., 2019). Many recent authors have proposed and shown the benefit of using "perceptual metrics" in order to better analyse the performance of GANs (Lucas et al., 2019: 1). These perceptual metrics can then be used to aid network optimisation.

In order to implement some perceptual metrics, a function was created that checked the accuracy and uniqueness of the chord voicings generated by the generator after each training epoch. The accuracy was a measure of how many generated chord voicings didn't have the presence of incorrect notes (as presented in table 4.3, page 22). The uniqueness was implemented as the number of chord voicings that were unique.

By examining these perceivable metrics after each training epoch, the true performance of the generator network could be seen. This allowed for Goodfellow’s principle to be more easily followed in order to optimise the network architecture.

The final adapted cDCGAN generator architecture is presented below in figure 5.16. Looking at figure 5.16, it can be seen that the generator takes 2 inputs, a 1D random noise vector of length 100, and a single chord label integer. The last layer outputs a 3x12 matrix (denoted by None, 3, 12, 1), which represents a generated chord voicing binary image. The inner layers concatenate the noise vector and chord label into a 3-dimensional array and use convolutional and down-sampling layers to map the array into a 3x12 chord voicing matrix. It is the parameters of these inner layers that learn how to transform chord labels into convincing chord voicings.

The discriminator model architecture can be seen at Appendix B.

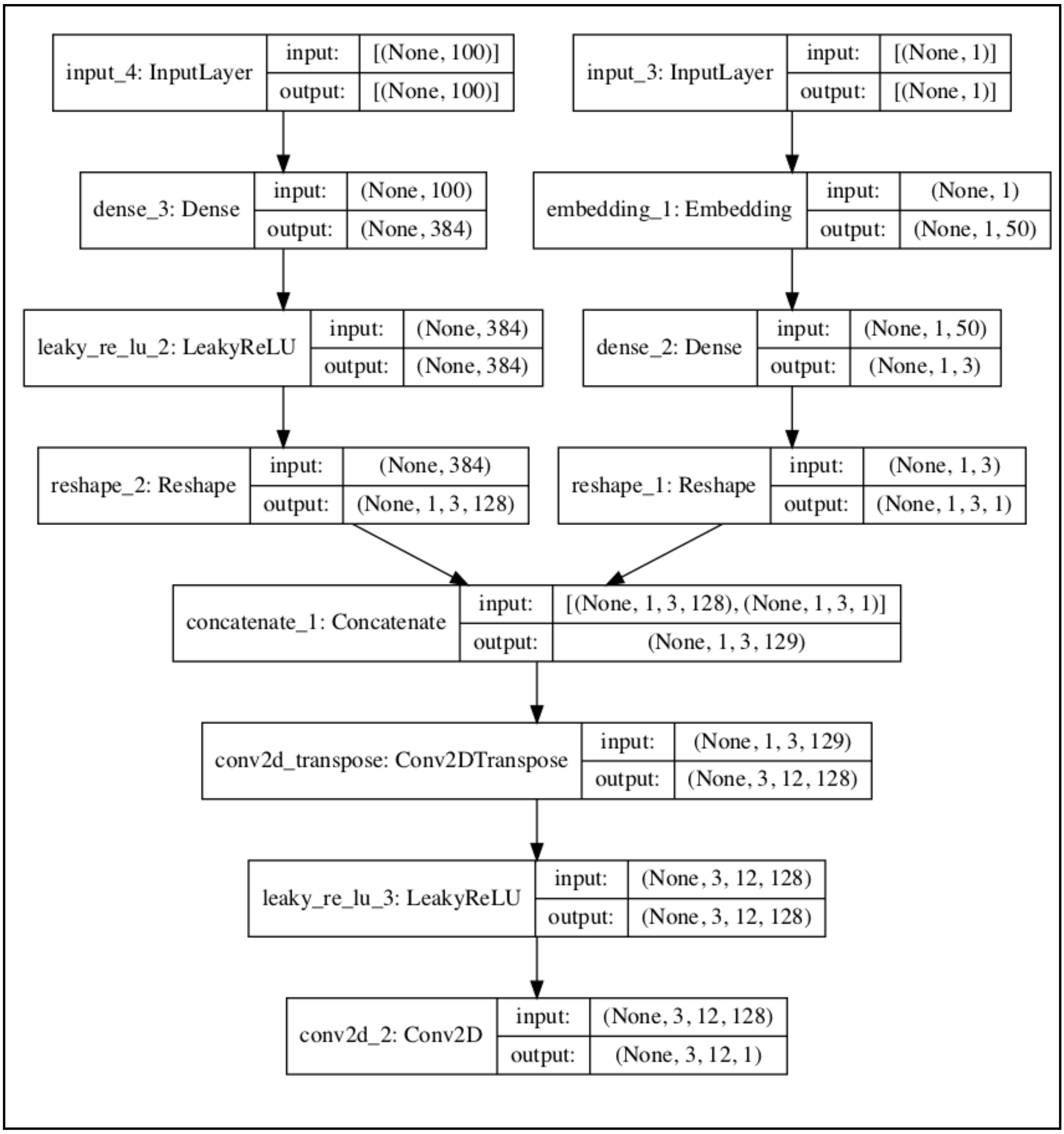


Figure 5.16 Adapted cDCGAN generator network architecture.

5.4.4 Results

After adapting Brownlee's cDCGAN model to fit the binary image training data and implementing a system to measure perceivable metrics, Goodfellow's principle was followed in order to maximise the accuracy and uniqueness of the generators output chord voicings.

The cDCGAN was trained for a total of 250 epochs. Figure 5.17 shows the accuracy and uniqueness curves of the generator. The performance of the trained model after 250 epochs is presented in table 5.7.

As can be seen in table 5.7, the adapted cDCGAN model is able to conditionally generate chord voicings of a given chord label with high accuracy and satisfactory uniqueness. These outcomes meet the desired results that were set out in objective 3 at the start of the section.

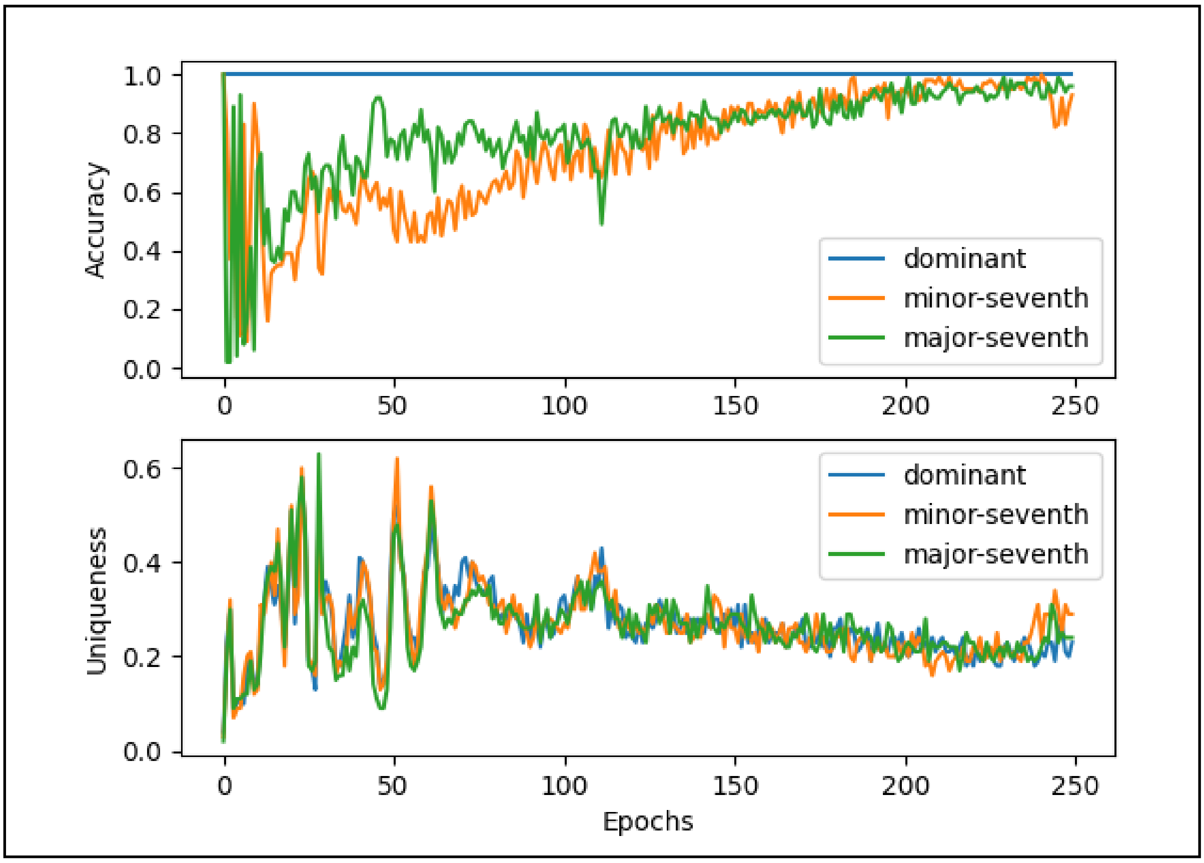


Figure 5.17 Generated chord accuracy and uniqueness after 250 training epochs.

	Dominant	***Minor-seventh***	***Major-seventh***
Chord Accuracy	1.00	0.93	0.96
Chord Uniqueness	0.23	0.29	0.24

Table 5.7 The performance of the trained adapted cDCGAN model.

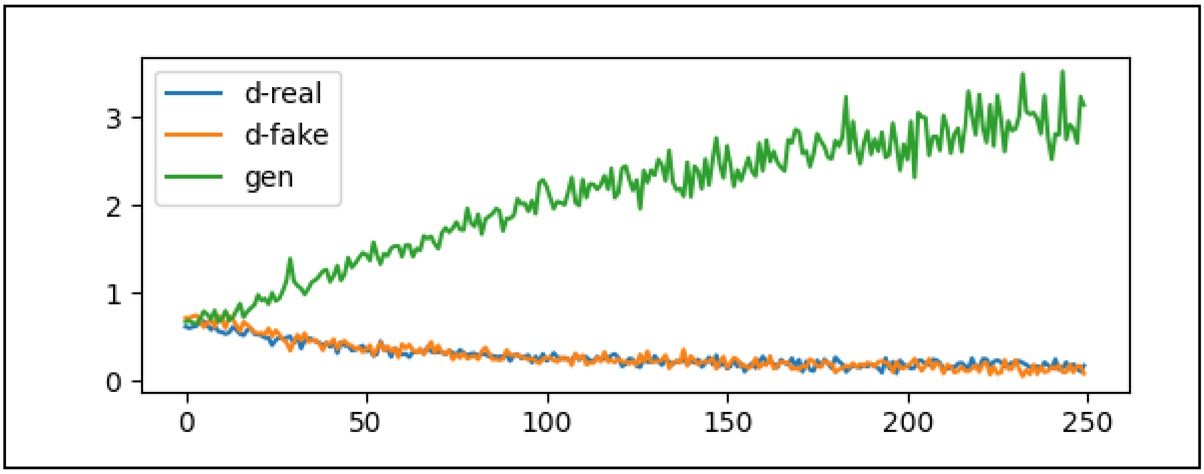
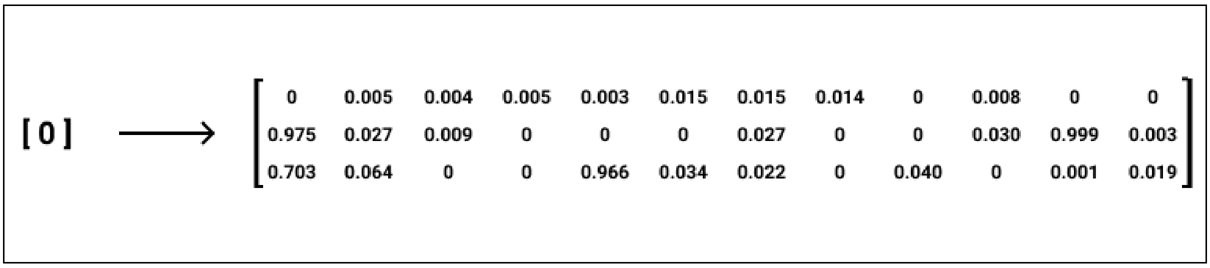


Figure 5.18 Discriminator and Generator loss after 250 training epochs.

Looking at figure 5.18, the discriminator loss with respect to generated images stabilised at around 0.25 after 250 epochs. This suggests that the trained generator is performing poorly, as it is only able to trick the generator with ¼ of its generated chord voicings. However as mentioned previously, research has shown that discriminator loss is not always a reliable indicator of generator performance (Lucas et al., 2019). This is reflected in the following section, in which it will be found that the generated chord voicings have a high degree of musicality.

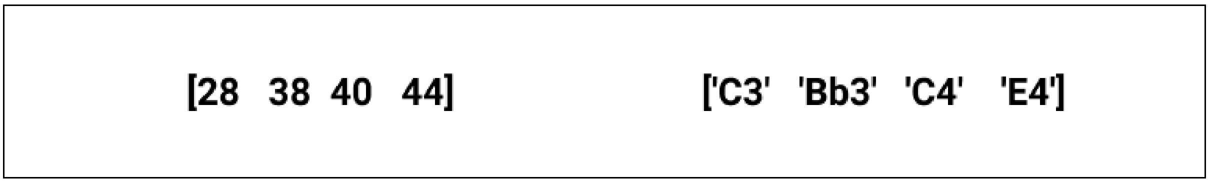
Figure 5.19 below shows an example of an input and output of the trained generator model. The integer label of 0 represents a dominant chord, and the output matrix is a 3 x 12 chord voicing image. Each number represents a probability of that note being in the chord. Figure 5.20 shows the generated chord matrix as note numbers and as note labels. Note that the first value in the output matrix is 16, or C2.

The adapted cDCGAN model can now be used by the LSAS to perform harmonic lead sheet arrangement as it can take successfully take chord symbols and output chord voicings to represent them. More details of this are presented in the following section.



$$[0] \longrightarrow \begin{bmatrix} 0 & 0.005 & 0.004 & 0.005 & 0.003 & 0.015 & 0.015 & 0.014 & 0 & 0.008 & 0 & 0 \\ 0.975 & 0.027 & 0.009 & 0 & 0 & 0 & 0.027 & 0 & 0 & 0.030 & 0.999 & 0.003 \\ 0.703 & 0.064 & 0 & 0 & 0.966 & 0.034 & 0.022 & 0 & 0.040 & 0 & 0.001 & 0.019 \end{bmatrix}$$

Figure 5.19 An example of the trained generators input and output. The input is an integer label representing a dominant chord, and the output is a 3x12 matrix.



[28 38 40 44] ['C3' 'Bb3' 'C4' 'E4']

Figure 5.20 The generated dominant chord represented as note numbers and note labels.

6. Leadsheet Arrangement System (LSAS)

The Leadsheet Arrangement System (LSAS) is the core system of this project. It takes as input a lead sheet in MusicXML format, and outputs an arrangement of that lead sheet. The LSAS consists of three subsystems – the *data_extractor, chord_generator, and leadsheet_arranger*. Each of these systems function as individual components and serve to complete three operations:

1. The *data_extractor* extracts the chord symbols from the lead sheet as well as some meta-information.

2. The *chord_generator* passes the extracted chord symbols into the trained cDCGAN generator model.

3. The *leadsheet_arranger* takes the generated chords and uses the extracted meta information to format and insert them into the lead sheet, transforming it into a full arrangement.

Figure 6.1 highlights these operations.

In this section, the development of the LSAS is presented. Following this, some input and output examples are presented, and those outputs along with the system itself are evaluated.

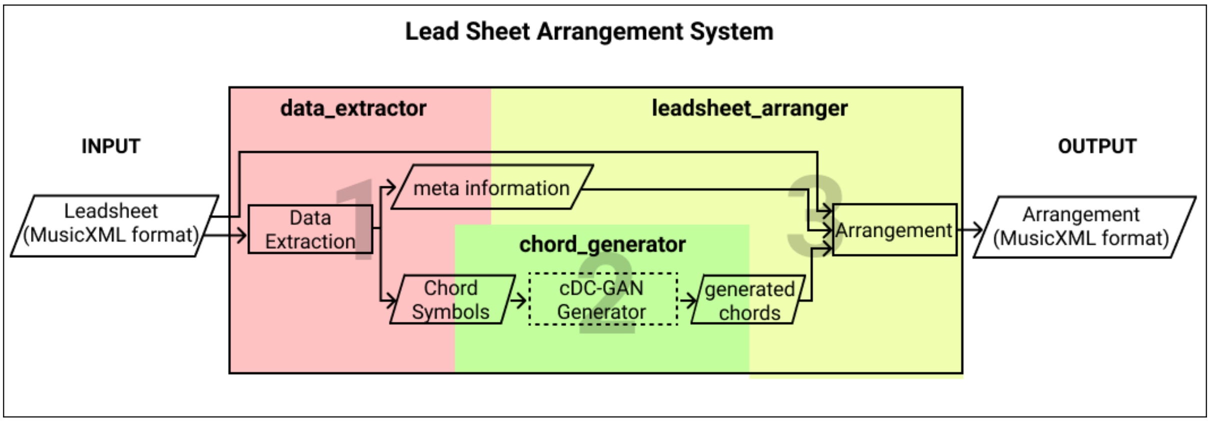


Figure 6.1 LSAS system diagram.

6.1 System Requirements

1. **1.** To take MusicXML piano lead sheets as input.

2. **2.** To extract chord symbols and meta-information from inputted lead sheets with a high degree of accuracy (more than 99%).

3. **3.** To use the trained cDCGAN model to generate chord voicings for a lead sheet's chord symbols.

4. To insert the generated chords into the lead sheet MusicXML file and make any modifications so that the system outputs a correctly formatted MusicXML arrangement.

5. To output highly accurate, varied, and musical arrangements for a given lead sheet.

6. To work on all piano lead sheets in MusicXML format including previously unseen lead sheets.

7. To have a highly usable command line interface that enables musicians to use the tool.

6.2 Development

This section will give detail on each subsystem and outline how it was developed.

6.2.1 data_extractor

The data_extractor subsystem takes as input a MusicXML lead sheet, performs some data parsing and extraction functions, and outputs the lead sheet's chord symbols and meta-information. The purpose of the data_extractor system is two-fold. Firstly, it extracts and presents the chord-symbols of the lead sheet to the chord_generator subsystem in a format that the trained cDCGAN generator can understand. This allows the generator to use the chord symbols to generate chord voicings. And secondly, it extracts meta-information such as the time signature & key signature and passes it to the leadsheet_arranger. This meta-information helps the leadsheet_arranger to insert the generated chords into the MusicXML lead sheet and format it into a full arrangement.

The development of the data_extractor system followed a very similar design pattern to that of the Chord Scraper, outlined in section 4.3.1. The system uses Python's Element Tree library to transform the lead sheet into an ordered tree data structure. By doing this, the chord symbols are easily extracted by traversing through the tree and finding each MusicXML harmony element. The extracted chord symbols are transposed so that their root is C and represented as chord labels. For example, the chord symbol Gm7 would first be transposed to Cm7. The root information would then be removed leaving only the chord type, which is "minor-seventh". The chord type labels are then passed to the chord_generator subsystem.

The data_extractor also uses tree traversal to extract the lead sheets key signature, time signature, and divisions. It does so by first traversing the tree to find the first measure (bar) element. The children of the first measure element contain all of the meta information needed. Figure 6.2 on the next page shows the tree traversal required to extract the meta-information.

This meta information is stored in a Python dictionary and outputted to the leadsheet_arranger subsystem. The key signature helps the leadsheet_arranger subsystem to transpose the chord voicings back to their original root. The time signature and divisions help the leadsheet_arranger subsystem to decide how long to make the notes of the generated chord voicings.

As mentioned above, there was a sizable functional overlap between the data_extractor subsystem and Chord Scraper. Common functionality could have been modularised in Python; however this would have been at the expensive of creating interdependencies between the two systems. The decision was made to reimplement any overlapping functionality so that the systems could be developed and used individually in future research projects.

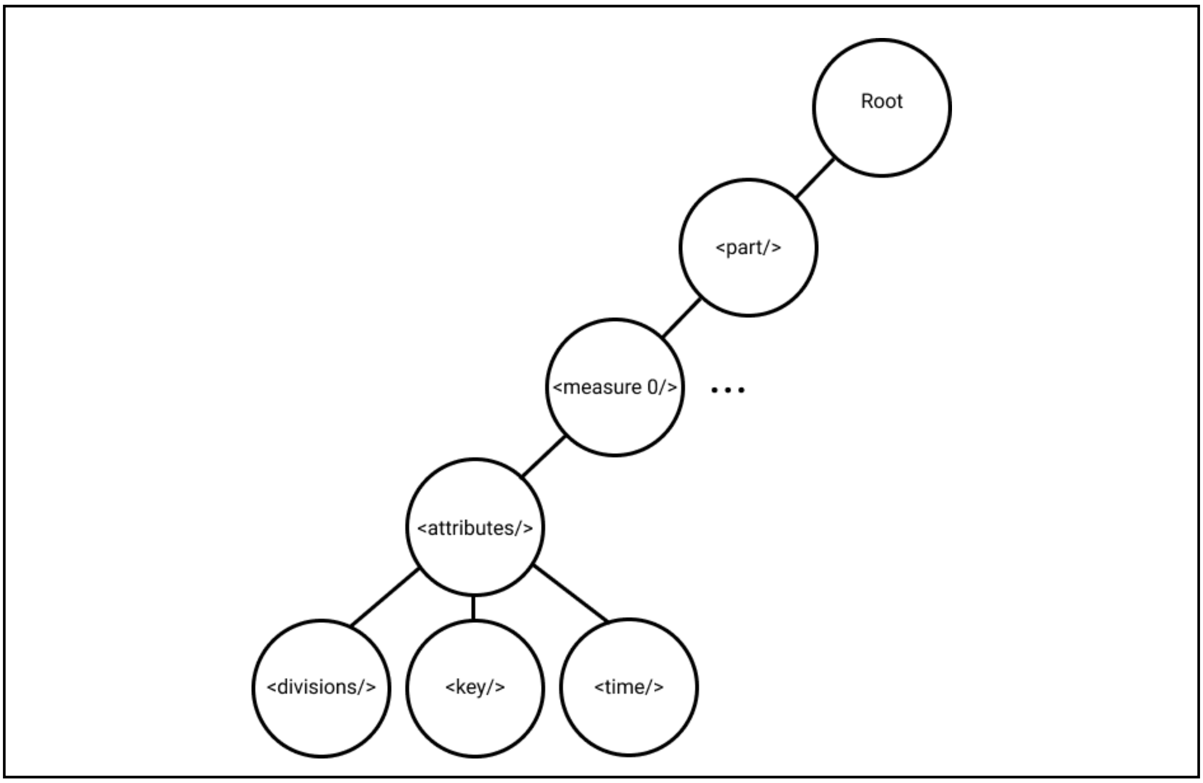


Figure 6.2 MusicXML Ordered Tree Data Structure showing meta-information nodes.

6.2.2 chord_generator

The chord_generator system takes as input a list of chord-type labels, and outputs a list of chord voicings that represent the inputted chord-type labels. The chord voicings are outputted as a list of note numbers, which are then passed to the leadsheet_arranger subsystem. Representing the chord voicings as numbers allows the leadsheet_arranger to easily transpose them back to their original root. An input-output example of the chord_generator subsystem is shown below:

If the chord_generator was given the following list of chord-type labels:

```
["dominant", "major-seventh", "dominant"]
```

The corresponding output would be something similar to:

```
[ [28, 38, 40, 44], [28, 32, 35, 39], [16, 27] ]
```

As can be seen, three chord voicings are outputted by chord_generator. Each number in the voicings represents a note on the piano.

The development of the chord_generator system was straightforward. The system imports the trained cDCGAN generator model and passes it the list of chord type labels, which in turn outputs generated chord voicings. As shown in figure 5.19 on page 47, the generator outputs chord voicing predictions as 3x12 matrices with floating point values between 0 and 1. The chord_generator subsystem converts the 3x12 matrices to the note number lists as shown above.

This is done by first flattening each 3x12 matrix to a 1D array of length 36. The indices in the array represent note numbers from 16 (C2) to 51 (B4). To identify the notes of a chord voicing, the 1D array is iterated over to find all of the index positions that hold a value that is greater than 0.5. For example, if the first number in the array was 0.9, then the note number 16 (C2) would be added to the note number list. This process is highlighted below in figure 6.3.

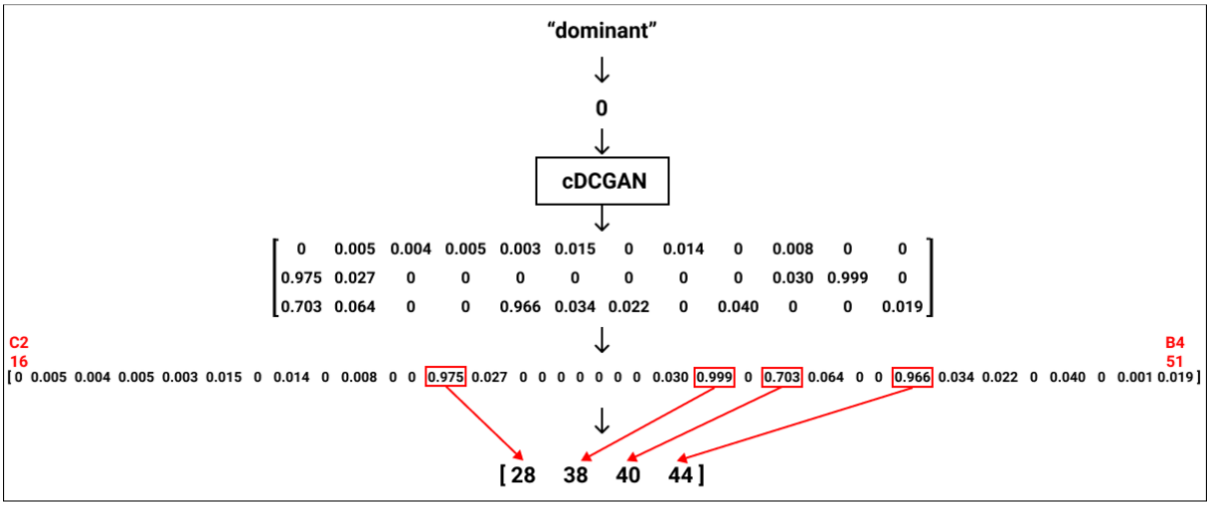
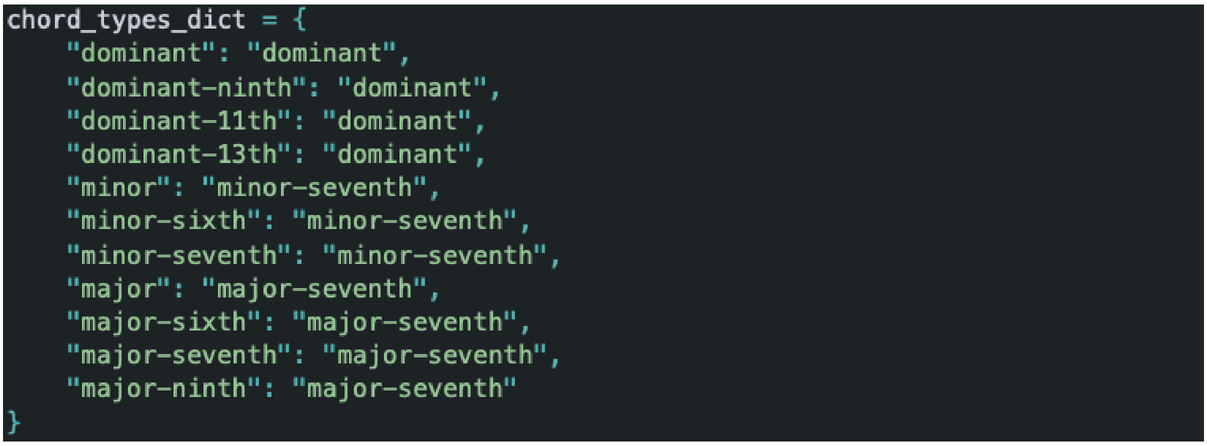


Figure 6.3 Flow chart showing chord_generator operations.

As seen in figure 6.3, a dominant chord label is given to the chord_generator, which is then encoded as its associated integer label (as outlined in table 5.3, page 36). This 0 is then given to the cDCGAN model, which returns a 3x12 matrix of probabilities indicating whether each note should be included in the chord voicing. The matrix is then flattened, and the chord voicing is extracted and presented as a note number list.

The chord_generator also generates a random noise distribution and passes it into the generator alongside each of the chord type labels. As explored in the earlier cDCGAN development section, this enables the generator to output varied chord voicings for the same chord type label.

As outlined in section 5, the trained cDCGAN model can only generate chords for 3 chord type labels – dominant, minor-seventh, and major-seventh. In order to increase the capability of the chord_generator, each of the three compatible chord-types were paired with other non-supported chord-types that were closely related. For example, a non-supported major-sixth chord type was paired with the supported major-seventh chord type. Both of these chords have 3 of the same notes in their base voicing configuration, with only 1 note slightly differing. This means that they sound very similar. Therefore, generating major-seventh chord voicings for major-sixth chord label inputs allowed for additional capability whilst still maintaining a good level of correctness. This "similarity-mapping" approach increased the chord_generator's input capacity from 3 to 11; all of the compatible input chord-type labels can be seen in figure 6.4 by looking the *chord_types_dict*'s keys.



```python
chord_types_dict = {
    "dominant": "dominant",
    "dominant-ninth": "dominant",
    "dominant-11th": "dominant",
    "dominant-13th": "dominant",
    "minor": "minor-seventh",
    "minor-sixth": "minor-seventh",
    "minor-seventh": "minor-seventh",
    "major": "major-seventh",
    "major-sixth": "major-seventh",
    "major-seventh": "major-seventh",
    "major-ninth": "major-seventh"
}
```

Figure 6.4 Python dictionary showing how similar chord labels were assigned to 1 of the 3 cDCGAN's supported chord labels.

It must be noted that in doing this, some of the chords generated by the chord-generator would, by strict musical theory standard, be slightly incorrect. For example, although C major-sixth and C major-seventh chords share three of the same notes, a C major-sixth base voicing configuration has a 4th note of A, as opposed to a C major-seventh, which has a 4th note of B. These differences would be present in chord voicings generated by the cDCGAN model. The implications of this similarity dictionary are explored further in the results section.

Aside from the 8 additional chord labels, there were also many existing chord labels that the generator could not support due to them being too dissimilar to the 3 supported chord labels. Instead of giving these incompatible chord labels no voicing, they were instead given the notes C2 (16) and C3 (28). These notes would later be transposed back to their original root, and act as bass notes. This adaptation allowed the LSAS to avoid any unwanted gaps in the arrangements. An example of an unsupported chord label voicing is shown below in figure 6.5 on the next page. Looking at the figure, the D augmented-seventh chord label has the notes D2 and D3 (transposed from C2 & C3), which as mentioned, act as simple bass notes.

The implementation details of the similarity-mapping and substitution systems were as follows. A function was created to validate each chord label before it was given to the cDCGAN model. The function looked up each chord label in the *chord_types_dict* shown above in figure 6.4. If the label was present as a key in the dictionary, then the function simply returned the corresponding chord label value. For example, looking back at figure 6.4 it can be seen that a lookup of "major-sixth" would return a cDCGAN compatible label of "major-seventh", and a lookup of "dominant" would simply return "dominant".

If the chord label wasn't present as a key, then a Key Error was raised. The function caught this error and printed an error message indicating that the chord could not be properly voiced. The simple 2-note voicing was then added to the list of generated chords. An example of this would be looking up the chord-type – "augmented-seventh", as seen in figure 6.5.

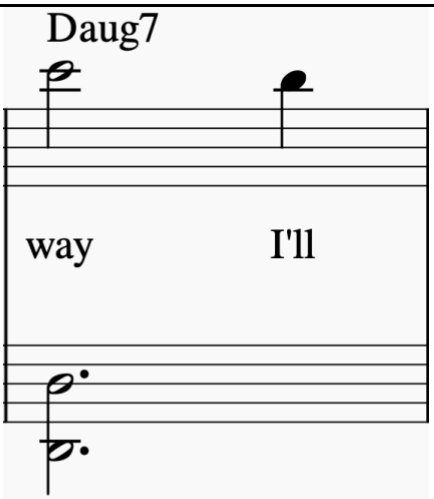


Figure 6.5 1-bar extract of a full arrangement produced by the LSAS showing bass notes given to an unsupported chord label.

6.2.3 leadsheet_arranger

The leadsheet_arranger subsystem takes lead sheet meta-information and generated chord voicings and uses them to transform the inputted MusicXML lead sheet into a full arrangement of the same format. The leadsheet_arranger contains the main method of the LSAS, and is where all of the other subsystems are executed. The main methods of the other subsystems are imported using Python's module import system.

Firstly, the leadsheet_arranger executes the main methods of the data_extractor and chord_generator subsystems. Doing this gives it the meta-information and generated chords.

It then inserts a bass clef into the input lead sheet in preparation to insert the notes of the generated chord voicings. Finally, it uses the meta-information to transform the generated chord voicings into MusicXML note elements and inserts them into the input lead sheet. The resulting output is an arrangement of the input lead sheet in MusicXML format.

The initial development of the leadsheet_arranger focused on inserting a bass clef into the input lead sheet. As mentioned previously, the bass clef is the bottom set of 5 lines that contain the chord voicing notes that the left hand plays. By inspecting fully arranged jazz solos from the initial MusicXML library, it was identified that the bass clef was represented as a clef element and was positioned as a child of the attributes element - shown in figure 6.2 on page 50. To insert a bass clef into the lead sheet, a function was written which created a clef MusicXML element with the standardised bass clef configuration. The function then traversed the lead sheet tree structure to locate the attributes element and insert the clef element as its child.

Further development of the leadsheet_arranger subsystem focused on converting the generated chords voicings into MusicXML note elements that could be inserted into the lead sheet. As the generated chords were rooted in C, they needed to be transposed back to their original root. In order to do this, a function was created that simultaneously iterated through the lead sheets original chord symbols and the generated chord voicings. For each chord, the function checked the original root note, and calculated its distance from C. Each note number in the chord voicing was then transposed by the calculated distance, thus bringing the chord voicing back to its original root. An example of this is presented on the next page.

Take the following chord symbol information and generated chord voicing:

Original chord root **= G**

Generated chord voicing (rooted in C) **= [28, 38, 43]**

The root note 'G' is looked up in a dictionary to see how many notes away it is from C (shortest path either up or down). In this case C is 5 notes (semi-tones) down from G, meaning all of the notes in the chord voicing will be reduced by 5:

Generated chord voicing (now rooted in G) **= [23, 33, 38]**

The chord voicing is now rooted in its original note; the note number 23 is equal to G2.

Creating *note* MusicXML elements required the chord voicings to be represented as note labels rather than note numbers. This process required the key signature of the lead sheet, as note numbers can point to different note labels depending on the key signature that they are in. For example, some key signatures refer to the black notes as flats (b), whereas some of the key signatures refer to the black notes as sharps (#). An example of this would be the black note between the white notes C and D, which half of the key signatures would call C-sharp (C#), and the other D-flat (Db). Inserting misnamed notes into the arrangement would result in its being incorrect and hard to read.

To implement the note integer to note label conversion, two conversation dictionaries were created with note integers as keys and corresponding note labels as values. One dictionary was used when the key signature referenced black notes as flats (b), and the other when the key signature referenced black notes as sharps (#).

The advantage of creating look up dictionaries is they allow for each conversion to be performed in constant time - O(1). An example of note integer to note label conversion is presented below:

Take the following lead sheet key signature and example chord voicing:

Key signature **= G minor (flat key)**

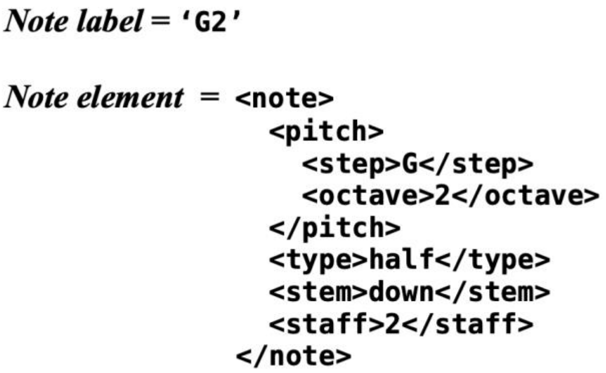
Chord voicing (note numbers) **= [23, 33, 38]**

The key signature is G minor, which labels the black notes as flats (b). Therefore, each of the note integers is looked up in the flat key signature dictionary to get the corresponding note label. The resulting note-label chord voicing representation is presented below:

Chord voicing (note labels) **= ['G2', 'F3', 'Bb3']**

Notice that there is an Bb3 note label present; this could have been wrongly named A#3 if the above system was not in place.

To convert the note labelled voicings into MusicXML elements, a parser function was created that looped through each note label in each voicing and converted it into its matching MusicXML note element. The Element Tree built in functions were used to create the *note* elements. An example of a note label – note element conversion is presented below:



Note label = **'G2'**

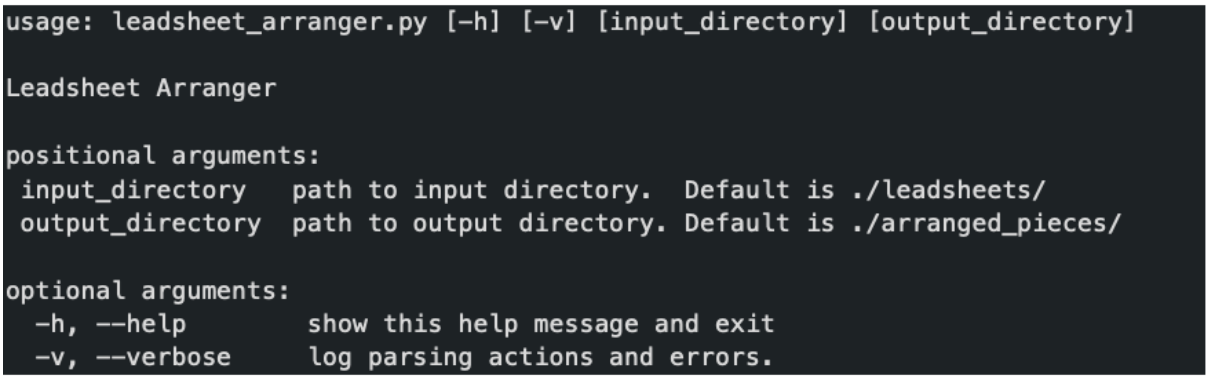
Note element =
```
<note>
  <pitch>
    <step>G</step>
    <octave>2</octave>
  </pitch>
  <type>half</type>
  <stem>down</stem>
  <staff>2</staff>
</note>
```

Looking at the note element, it can be seen that the pitch sub-element holds the note label information. The type element indicates how long the note is, and is determined by the time signature and division meta-information.

Once each chord voicing was represented as a series of MusicXML note elements, they could be inserted into the correct position in the lead sheet tree structure. Each chord voicings position was determined by locating the bar of its associated chord symbol. The note elements were then inserted as children of that measure (bar) element. The note elements were inserted after the end child of the measure element, resulting in them appearing in the bass clef.

Some error handling was also implemented within the leadsheet_arranger subsystem. For example, the system checks that the number of generated chords are equal to the number of chord symbols in the lead sheet. Checks are also performed on the lead sheet meta-information to ensure that it is accurate and conforms to MusicXML standards. Implementing this error handling ensured that the LSAS met objective 2.

As the leadsheet_arranger subsystem executes the main method of the LSAS, a CLI was developed to allow users to interact with the system and arrange lead sheets without having to open the source code. The CLI allows the user to specify an input directory and output directory. It also allows the user to enable a verbose mode, which results in errors being logged whilst the program is running. The CLI also has a help menu, which can be seen below in figure 6.6. Implementing the CLI allowed the LSAS to meet objective 7.



```
usage: leadsheet_arranger.py [-h] [-v] [input_directory] [output_directory]

Leadsheet Arranger

positional arguments:
 input_directory   path to input directory.  Default is ./leadsheets/
 output_directory  path to output directory. Default is ./arranged_pieces/

optional arguments:
  -h, --help        show this help message and exit
  -v, --verbose     log parsing actions and errors.
```

Figure 6.6 LSAS CLI help menu.

When the LSAS is executed, it iterates through each file in the input directory, and executes on any files ending in ".MusicXML" or ".XML". For each successful arrangement, a confirmation message is shown on the terminal along with the output path.

6.3 Results and Evaluation

The Lead Sheet Arrangement System (LSAS) was tested using the Wikifonia dataset, which is a collection of 7000 jazz lead sheets. Figures 6.7 and 6.8 show two examples of the LSAS performing lead sheet arrangement. In order to thoroughly test the system, it was executed with 200 randomly selected MusicXML lead sheets in the input directory. No major errors were reported, and 200 MusicXML arrangements were outputted. Of these 200 arrangements, 5 have been randomly selected for both a technical and participant led evaluation. Before presenting these evaluations, the capabilities and limitations of the LSAS are presented.

It is encouraged that the reader listens to some of the arrangements – they can be found in the *leadsheet_arranger* subdirectory or via the links in table 6.1 on page 59.

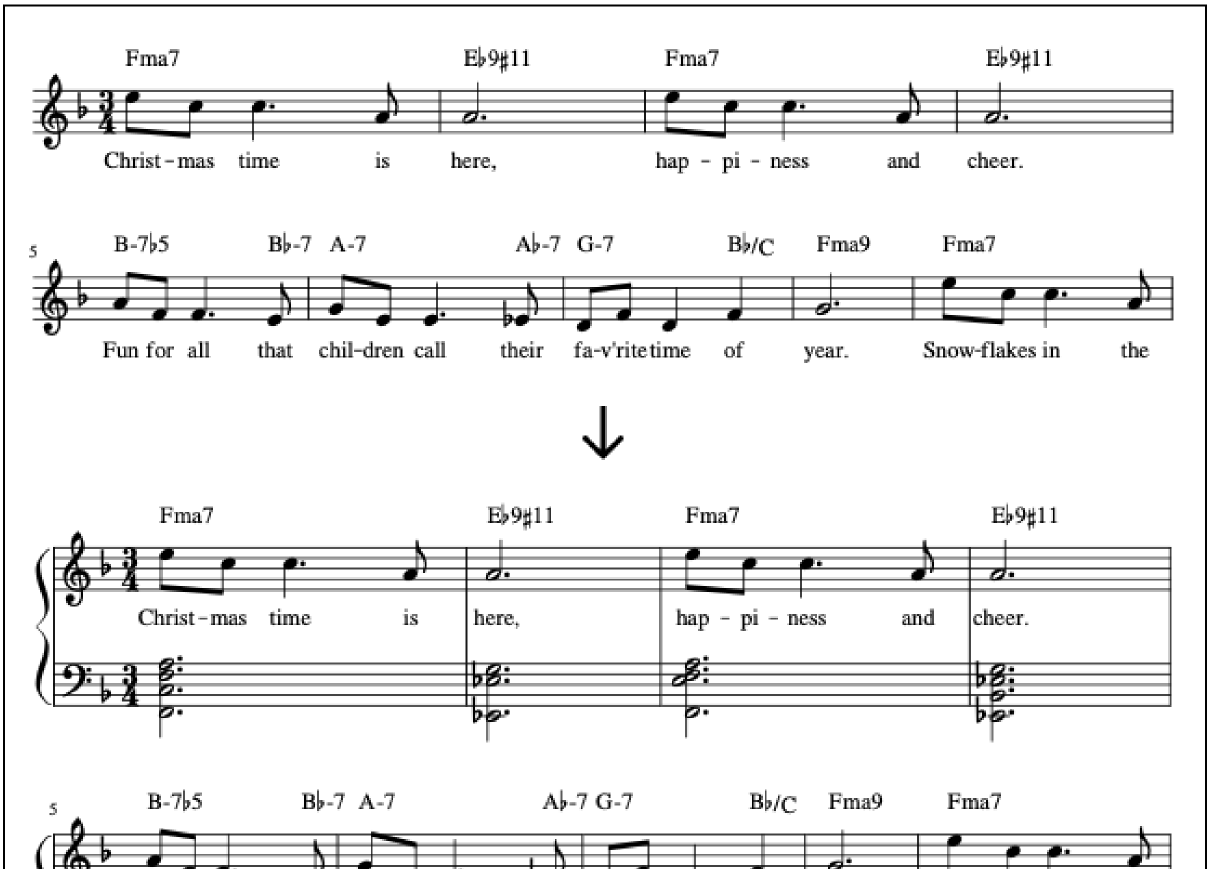

Fma7
E♭9♯11
Fma7
E♭9♯11
Christ - mas time is here, hap - pi - ness and cheer.
5
B-7♭5
B♭-7 A-7
A♭-7 G-7
B♭/C
Fma9
Fma7
Fun for all that chil-dren call their fa-v'rite time of year. Snow-flakes in the
Fma7
E♭9♯11
Fma7
E♭9♯11
Christ - mas time is here, hap - pi - ness and cheer.
5
B-7♭5
B♭-7 A-7
A♭-7 G-7
B♭/C
Fma9
Fma7
Fun for all that chil-dren call their fa-v'rite time of year. Snow-flakes in the

Figure 6.7 First 8 bars of the song Christmas Time is Here by Lee Mendelson and Vince Guaraldi. The top 8 bars show the song as a lead sheet, and the bottom 8 bars show the LSAS's arrangement.

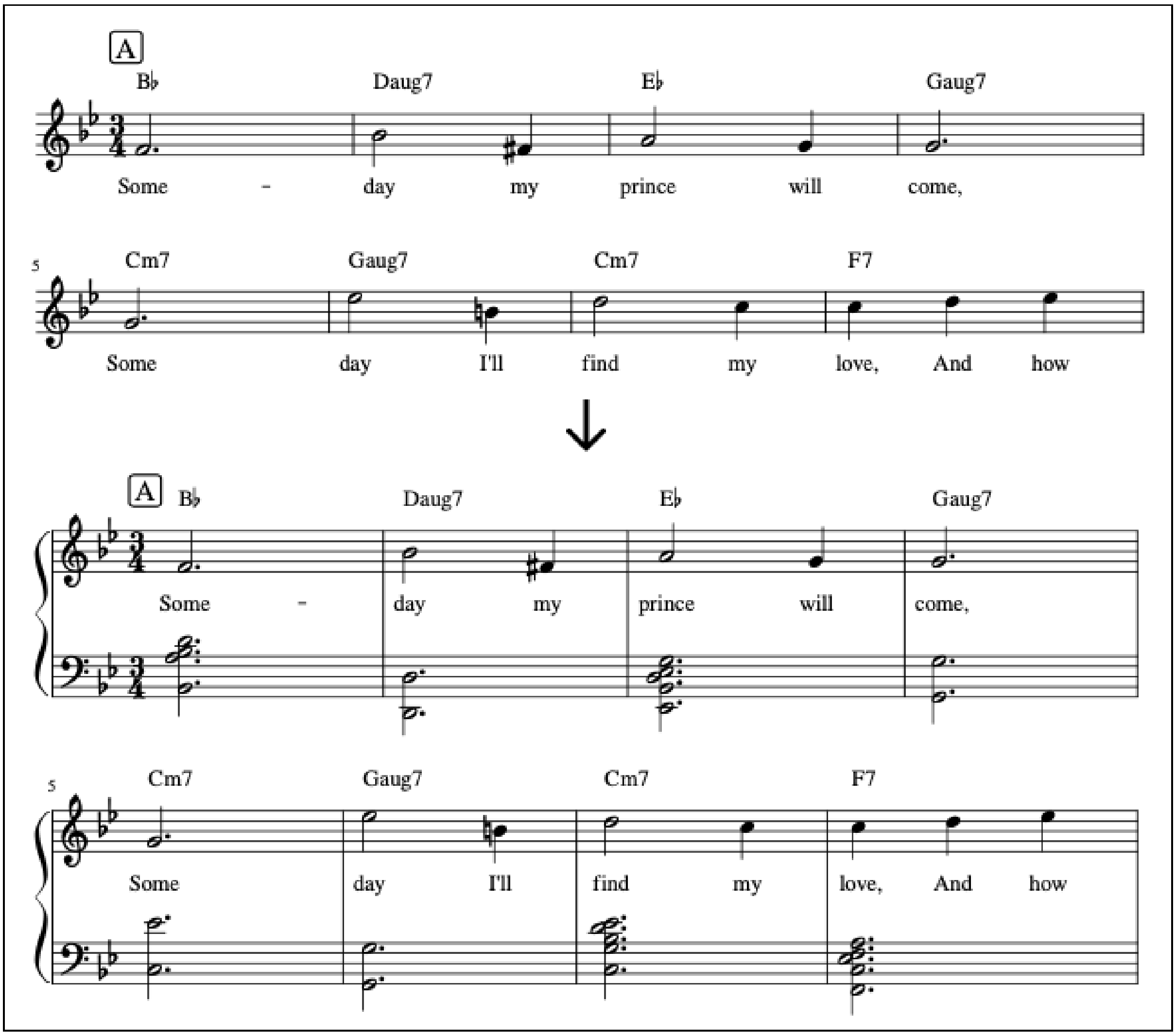

A
B♭
Daug7
E♭
Gaug7
Some - day my prince will come,
5
Cm7
Gaug7
Cm7
F7
Some day I'll find my love, And how
A
B♭
Daug7
E♭
Gaug7
Some - day my prince will come,
5
Cm7
Gaug7
Cm7
F7
Some day I'll find my love, And how

Figure 6.8 First 8 bars of the song Some Day My Prince Will Come by Frank Churchill. The top 8 bars show the song as a lead sheet, and the bottom 8 bars show the LSAS's arrangement.

6.3.1 Capabilities

- The system can successfully perform harmonic arrangement on a given lead sheet.
- The harmonies (chord voicings) are highly accurate – more than 93% of the voicings contain no incorrect notes.
- Due to the nature of the cDCGAN generator, the number of varied arrangements possible for a lead sheet is immeasurably large.
- The system can arrange any unseen lead sheet as long as it is in MusicXML format.
- The system can arrange any genre of piano lead sheet and is not limited to jazz.
- The system can arrange a standard-length lead sheet in under 1 second and can arrange an infinite amount of lead sheets in one execution.

- The system is flexible in that the generator model can be easily substituted or retrained on a different dataset.

6.3.2 Limitations

- The system cannot perform the task of rhythmic arrangement (this was formally outlined as a known limitation).

- The chord voicings generated for 8 of the 11 supported chord types could be slightly inaccurate (see section 6.2.2).

- There are a number of chord symbols that the system cannot generate proper chord voicings for and must provide only simple bass notes as a substitute.

As can be seen in the above capabilities and limitations, most of the system requirements were met. The LSAS used the data_extractor to take MusicXML lead sheets and perform extraction, thus meeting objectives 1 and 2. The chord_generator subsystem was able to predominantly meet objective 3 by generating chord voicings for a lead sheet's chord symbols. However as explored above, there were some chord types which only bass notes could be provided. The implications of this are explored in the following evaluations. Inspection of the 200 test arrangements showed that the leadsheet_arranger subsystem met objectives 4 and 6, as it was able to successfully arrange previously unseen lead sheets and output correctly formatted arrangements. The implementation of a simple CLI allowed the LSAS to meet objective 7.

In order to evaluate how the LSAS met objective 5, both a technical and participant led evaluation will be performed on arrangements produced by the system. The technical evaluation will consider 3 factors:

1. **Correctness** – how many unwanted notes were present in the chord voicings. The accuracy of the cDCGAN has already been analysed and presented in section 5.4.4 (table 5.7), however following the additional of the similarity-mapping system, it will be revisited.

2. **Playability** – how easy or hard are the arrangements for a pianist to play.

3. **Voicing Quality** – how well are the chords voiced. Are the notes of the voicing in the best sounding part of the piano, is there any evidence of advanced voicing techniques such as spread voicings or voice leading; both of which are defined later.

For the participant led evaluation, 20 beginner, intermediate, or advanced jazz pianists will be asked to examine and listen to 5 arrangements created by the LSAS. They will be asked a series of qualitative questions relating to the musicality of the arrangements.

Both of these evaluations are presented below.

6.3.3 Selected Arrangements for Evaluation

Below are the 5 randomly selected arrangements that will be used in both the technical and participant led evaluations:

Title	***Artist***	***Lead sheet***	***Arrangement***	
		PDF	***PDF***	***MP3***
Christmas Time is Here	Lee Mendelson and Vince Guaraldi	link	link	link
Baltimore Oriole	George Harrison	link	link	link
Nardis	Miles Davis	link	link	link
Autumn Leaves	Joseph Kosma	link	link	link
Someday My Prince Will Come	Frank Churchill	link	link	link

Table 6.1 A table showing 5 randomly selected arrangements chosen for evaluation.

(If viewing offline - the files can be found at ./leadsheet_arranger/ inside leadsheets/ & arranged_pieces/)

6.3.4 Technical Evaluation

As proposed above, the technical evaluation was conducted based on the three criteria listed above. Each of the 5 selected arrangements were manually examined and given a rating for each of the criteria. The technical evaluation was performed by the author of this paper, who is an intermediate jazz pianist. Respected musical theory literature was used to support the evaluation where possible.

A manual investigation of the 5 arrangements found that out of a total of 160 chord voicings, there were 11 incorrect chord voicings present – a total of 6.9%. The chord voicings were deemed incorrect by comparison with chord type definitions taken from the *Berklee Book of Jazz Harmony* (Mulholland & Hojnacki, 2013). The investigation also found that 17 of the 160 chord voicings were unplayable – a total of 10.6%. This judgement was based on the assumption that a pianist can reach a maximum span of 10 notes, or more specifically, a major 10th interval. It can be therefore noted that the number of unplayable chord voicings could be higher depending on the player.

In general, the results of the first two criteria can be viewed as favourable. The number of incorrect chord voicings present in the arrangements was around 4% higher than when the cDCGAN generator model was independently tested. However, this is explainable by the fact that an additional 8 similar chord types were incorporated into the arrangement system. As these additional chord types are slightly different from the generator-compatible chord labels of which they were associated with, the number of inaccuracies increased. For example, figure 6.9 below shows a 1-bar extract of an LSAS arrangement of the song “Over the Rainbow” by Errol Garner. The chord type is “major”, which is supported by the LSAS through its mapping to the “major-seventh” chord-type. As a result of this, the generated major chord has an incorrect note – “B”, which is highlighted in red. The second bar shows the correct version of the chord voicing.

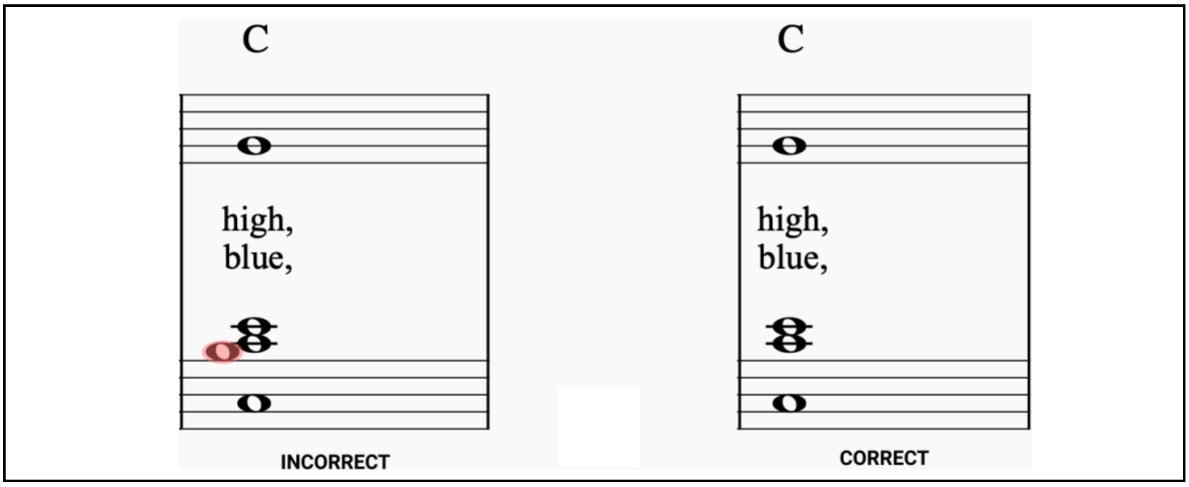


Figure 6.9 Example of LSAS having incorrect voicings as a result of the chord-type similarity work-around.

The number of unplayable chords is more than likely a result of both the dataset used to train the cDCGAN model, and the nature of the model itself. It can be assumed that some of the arrangements in the source MusicXML library had chord voicings that were unplayable. This could have been because the arranger had a larger than average hand span, or that the arrangement was of poor quality. The model may also have generated some unplayable voicings.

Some post-processing of the generated chord voicings could be implemented in order to ensure that every voicing was playable. The initial training dataset could also be cleaned to ensure that all of the training data voicings were playable.

The investigation into the quality of the chord voicings provided mixed findings. In general, the chord voicings were positioning in their most optimal position. The majority of chords were spread across the range of notes between C2 and C4, which is the best sounding range of the piano for chords (Mulholland & Hojnacki, 2013). There were however 15 voicings (9%) which had a number of closely positioned notes in the lower range of the piano. This resulted in them sounding muddy and unpleasant.

There was a good amount of evidence of *spread voicings* across the arrangements. This is a voicing technique in which the notes of a chord are divided a spread across multiple octaves in a way that makes them sound more pleasurable (Mulholland & Hojnacki, 2013:218). Around 40% of chord voicings could be considered as *spread voicings*.

The presence of *voice leading* was not detected through the investigation. Voice leading is a technique which ensures that sequences of chords are smoothly connected and flow together in a way that is pleasurable (Mulholland & Hojnacki, 2013:5).

The lack of voice leading is a result of the cDCGAN model. The cDCGAN generator can only generate individual chord voicings and has no concept of how two or more voicings fit together. As explored in section 2.2, there has been some research in the recent music generation literature which employs Recurrent Neural Networks (RNN) in the task of generating sequentially related music. If a larger dataset was gathered, an RNN model could be combined with the cDCGAN model in order to produce a level of voice leading within the generated arrangements.

Some of the issues explored in this technical analysis could be solved by implementing a post chord-generation processing system. The system could iterate through each chord, remove any incorrect notes, and reconfigure the notes in way that ensured playability. It could also implement a degree of voice leading by reconfiguring each chord’s structure so that it was similar to its predecessor.

Both the implementation of an RNN to aid voice leading and post processing system to improve general performance could be explored in further research.

6.3.5 Participant led Evaluation

A total of 20 beginner, intermediate or advanced jazz pianists were contacted via a popular jazz piano Facebook group. The participants were provided with the table presented in table 6.1 and asked to answer a list of 8 questions. The questions are presented in figure 6.10.

Questions Asked
1. How would you define your level of jazz piano playing experience?
2. How would you define your level of jazz musical theory knowledge?
3. Please comment on the correctness of the chord voicings.
4. Please comment of the quality of the chord voicings.
5. Please add any further comments about the chord voicings.
6. Do you think the chords fit together?
7. What do you think of the whole arrangement?
8. Would the arrangement be of use to you as a pianist? If not, why?

Figure 6.10 A list of questions provided to each participant.

At this time, the author of the paper is still awaiting the results of this evaluation. The results will be added to the paper as soon as they are available.

One of the most notable capabilities of the LSAS is that it can arrange the same lead sheet in an immeasurably large number of ways. In practice, this means that if the outputted arrangement is undesirable, it can be re-arranged an infinite number of times until a desired outcome is produced.

7. Conclusion

This section will aim to provide an evaluation of the project as a whole. First, a summary of the achievements of the project are presented and compared with the initial aims and objectives. Following this, an analysis is provided on how the project and its results fit into the current field of research of deep learning and music, as well as if the project has provided any novel contributions. This is followed by an outline of some key insights that were gained through completion of this project. And finally, the limitations of the scope of the project are explored, with some suggests on how the scope of the project could be expanded by the author or in future research.

During this research project, all of the objectives that were set out were incrementally developed and achieved. This incremental development process eventually resulted in a fully functioning lead sheet arrangement system which satisfied the initial aim of the project.

In its beginnings, the project met its first objective by gathering a library of MusicXML fully arranged jazz piano songs which importantly all had annotated chord symbols. To meet objective 2, a chord scraper was developed that could successfully extract chord symbol-chord voicing pairs from MusicXML piano arrangements. This scraper allowed for the novel Jazz-Chords dataset to be presented. This dataset met the third objective of the project.

The Jazz-Chords dataset presented the opportunity to train a conditional generative adversarial network (cGAN) to be able to generate jazz chord voicings, something that to the author's knowledge of the current literature, has never been done before. An existing model, the cDCGAN, was adapted to fit the jazz-chords dataset, and successfully trained to be able to conditionally generate different kinds of jazz chord voicings. The capabilities of the trained cDCGAN model met objectives 4 and 5 of the project.

Finally, the overarching Leadsheet Arrangement System (LSAS) was developed with the trained cDCGAN model at its core. The LSAS is capable of taking any piano lead sheet in MusicXML format and outputting a full arrangement. The development and results of the LSAS met the final objective and overall aim of this research project.

The research of this project sits broadly within the field of deep learning in music and extends the research of two subfields – chord generation and lead sheet arrangement.

In the area of chord generation, the developed cDCGAN model along with the Jazz-Chords dataset aim to extend the research of Chen et al. (2020) and their *Chord Jazzification* system. The presented cDCGAN model can generate additional chord types to Chen et al's model. The results were also achieved using a different deep learning approach, i.e. cGAN vs RNN. As mentioned in the technical evaluation (section 6.3.4), the results of this research suggest that a combination of these two approaches could yield improved results.

In the area of lead sheet generation, this research project aimed to extend upon the research of Liu et al. (2018a, 2018b) and their *lead sheet generation and arrangement system.* This research provides an alternative, chord by chord approach to performing lead sheet arrangement. This research also provides an alternative framework for carrying out the task of lead sheet arrangement, which is through the manipulation of MusicXML as opposed to the manipulation of MIDI. The benefit of this framework is that MusicXML holds much more information. For example, annotated chord symbols can help to provide deep learning models with more data that can help to improve performance.

The LSAS also provides a springboard for further research, as it provides a model that can be retrained using different datasets, potentially from different genres of music. The model can also be replaced by another generative model and still make use of the surrounding systems.

The Chord Scraper system and the Jazz-Chords dataset offer novel contributions to the broader field of deep learning in music. The Chord Scraper system can be used on any MusicXML files to extract infinitely large datasets for use as training data in further research.

The Jazz-Chords dataset can also be employed by researchers across the field of music in deep learning in order to train a wide variety of generative models.

The Chord Scraper could also be employed in the area of data science in music, as it is able to extract and present the harmonic and meta-information from piano music in a machine-readable format.

The LSAS also has the potential to offer a contribution to the jazz piano music community. One of the motivations of this project was to create a system that could help to improve the accessibility of jazz music to beginner pianists. The LSAS will be made publicly available, with the hope that beginner jazz pianists will use it to aid them in arranging lead sheets. It also has the potential to be used as an educational tool which can help beginner pianists improve their knowledge and exposure to jazz-chord voicings.

Carrying out this research project has provided the author with some important insights into the field of deep learning in music. Unless otherwise stated, these insights are present in the current literature, however they have been further highlighted throughout the results of this research. They are presented below.

The first of these insights is the general lack of available data within the field of music research, and in particular jazz music research. There is both a lack of primary data – sheet music in machine readable formats, and secondary data – music related datasets that have been extracted from primary data sources. This issue is widely acknowledged within the current literature (Ji et al., 2020), and was further highlighted in this paper when conducting an extensive review of the publicly available data in section 2.5.

This project has shown the effectiveness of representing chord voicings as binary images. To our knowledge of the current literature, this encoding method provides a novel approach to chord voicing encoding.

An issue that this project has highlighted is the challenge of how generative models are able to differentiate between different chord types. As shown in figure 5.6 on page 36, major and major-seventh chord voicings share a very similar note distribution. This is reflected in formal music theory, which defines them as only having one note differential (Mulholland & Hojnacki, 2013:viii). Representing chord voicings in a different data domain could help to polarise their differences. One example of this could be representing chord voicings as audio recordings, in which each chord voicings unique harmonic overtones (additional high-pitched sound) would be present. These distinguishing overtones could help deep learning models to differentiate between similar chord types.

This project has also highlighted the effectiveness of implementing perceivable metrics for optimising generative adversarial networks. The ability to track chord accuracy and uniqueness in real-time throughout each training iteration made Goodfellow’s optimisation principle far easier to apply.

The scope of this research project was to create a system that could create full arrangements of a given jazz lead sheet. As stated in the objectives, one of the restrictions of the project was that the system would only perform the harmonic subtask of lead sheet arrangement. This restriction has meant that arrangements produced by the LSAS have a lack of rhythmical interest. The reason for imposing this restriction was two-fold. Firstly, the task of generating sequential data through the use of deep learning is complex. This is reflected in the current literature, in which no one has proposed a system that successfully performs both harmonic and rhythmic lead sheet arrangement (Ji et al., 2020). And secondly, as explored previously, a network that could generate chord voicings that were sequentially arranged (RNN) would require a large amount of data, which in turn would require a much larger library than the MusicXML library that could be acquired in this project.

Overcoming the above stated barriers and extending the LSAS to being able to perform both harmonic and rhythmic arrangement is a high priority for future iterations of the LSAS’s development. Doing so would have a significant impact on the lead sheet arrangement literature as it would help to progress the field towards the goal of creating a highly intelligent musical arrangement AI. An iteration of the LSAS that could perform harmonic and rhythmic lead sheet arrangement would also be of significant interest to the jazz piano community, especially if the arrangements possessed a high degree of musicality.

The scope of this project with regards to data extraction was limited to retrieving a dataset of chord symbol-chord voicing pairs in order to train the cDCGAN model. However in order to progress the field of deep learning in music, there needs to be a significant breakthrough in the availability of data.

One solution to this issue would be the creation of a system that could parse images or PDFs of music notation into machine readable data representations such as MusicXML or JSON. Optical music recognition (OMR) systems have received large attention and development for many years; however, an effective system has to date not been proposed (Revelo et al., 2012). Solving this issue would arguably be the most significant breakthrough in the field, as it would in provide researchers with significantly more data than is currently available.

References

Borghuis, V., Angioloni, L., Brusci, L. and Frasconi, P., 2020, July. Pattern-Based Music Generation with Wasserstein Autoencoders and PRC Descriptions. In IJCAI (pp. 5225-5227).

Briot, J.P., Hadjeres, G. and Pachet, F.D., 2017. Deep learning techniques for music generation--a survey. arXiv preprint arXiv:1709.01620.

Brownlee, J., 2016. Master Machine Learning Algorithms: discover how they work and implement them from scratch. Machine Learning Mastery.

Brownlee, J., 2019. Generative adversarial networks with python: deep learning generative models for image synthesis and image translation. Machine Learning Mastery.

Chen, T.P., Fukayama, S., Goto, M. and Su, L., 2020. Chord Jazzification: Learning Jazz interpretations of chord symbols. In Proc. Int. Society for Music Information Retrieval Conf.

Deng, L., 2012. The mnist database of handwritten digit images for machine learning research [best of the web]. IEEE Signal Processing Magazine, 29(6), pp.141-142.

Dong, H.W., Hsiao, W.Y., Yang, L.C. and Yang, Y.H., 2018, April. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. In Thirty-Second AAAI Conference on Artificial Intelligence.

Emura, N., Miura, M. and Yanagida, M., 2006. Machine Arrangement in Modern Jazz-style for a Given Melody. In Proceedings of the 9th International Conference on Music Perception and Cognition (pp. 710-715).

Engel, J., Resnick, C., Roberts, A., Dieleman, S., Norouzi, M., Eck, D. and Simonyan, K., 2017, July. Neural audio synthesis of musical notes with wavenet autoencoders. In International Conference on Machine Learning (pp. 1068-1077). PMLR.

Eremenko, V., Demirel, E., Bozkurt, B. and Serra, X., 2018. Audio-Aligned Jazz Harmony Dataset for Automatic Chord Transcription and Corpus-based Research. In ISMIR (pp. 483-490).

Good, M., 2001. MusicXML for notation and analysis. The virtual score: representation, retrieval, restoration, 12(113-124), p.160.

Goodfellow, I., Bengio, Y. and Courville, A., 2016. Deep learning. MIT press.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y., 2014. Generative adversarial nets. Advances in neural information processing systems, 27.

Hadjeres, G., Pachet, F. and Nielsen, F., 2017, July. Deepbach: a steerable model for bach chorales generation. In International Conference on Machine Learning (pp. 1362-1371). PMLR.

He, J., Spokoyny, D., Neubig, G. and Berg-Kirkpatrick, T., 2019. Lagging inference networks and posterior collapse in variational autoencoders. arXiv preprint arXiv:1901.05534.

Ji, S., Luo, J. and Yang, X., 2020. A Comprehensive Survey on Deep Music Generation: Multi-level Representations, Algorithms, Evaluations, and Future Directions. arXiv preprint arXiv:2011.06801.

Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25, pp.1097-1105.

Liang, X., Wu, J. and Cao, J., 2019. MIDI-Sandwich2: RNN-based Hierarchical Multi-modal Fusion Generation VAE networks for multi-track symbolic music generation. arXiv preprint arXiv:1909.03522.

Liu, H.M., Wu, M.H. and Yang, Y.H., 2018, September. Lead sheet generation and arrangement via a hybrid generative model. In Proceedings of the 19th International Society for Music Information Retrieval Conference, Paris, France (pp. 23-27).

Liu, H.M. and Yang, Y.H., 2018, December. Lead sheet generation and arrangement by conditional generative adversarial network. In 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA) (pp. 722-727). IEEE.

Lucas, A., Lopez-Tapia, S., Molina, R. and Katsaggelos, A.K., 2019. Generative adversarial networks and perceptual losses for video super-resolution. IEEE Transactions on Image Processing, 28(7), pp.3312-3327.

Martin, J.L., 2014. Semiotic resources of music notation: Towards a multimodal analysis of musical notation in student texts. Semiotica, 2014(200), pp.185-201.

Mogadala, A., Kalimuthu, M. and Klakow, D., 2019. Trends in integration of vision and language research: A survey of tasks, datasets, and methods. arXiv preprint arXiv:1907.09358.

Mirza, M., & Osindero, S. (2014). Conditional generative adversarial nets. arXiv preprint arXiv:1411.1784.

Mulholland, J. and Hojnacki, T., 2013. The Berklee Book of jazz harmony. Hal Leonard Corporation.

O'Shea, K. and Nash, R., 2015. An introduction to convolutional neural networks. arXiv preprint arXiv:1511.08458.

Radford, A., Metz, L. and Chintala, S., 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434.

Sun, C., Shrivastava, A., Singh, S. and Gupta, A., 2017. Revisiting unreasonable effectiveness of data in deep learning era. In Proceedings of the IEEE international conference on computer vision (pp. 843-852).

Szelogowski, D., 2021. Generative Deep Learning for Virtuosic Classical Music: Generative Adversarial Networks as Renowned Composers. arXiv preprint arXiv:2101.00169.

Teng, Y., Zhao, A. and Goudeseune, C., 2017. Generating nontrivial melodies for music as a service. arXiv preprint arXiv:1710.02280.

Valenti, A., Carta, A. and Bacciu, D., 2020. Learning a latent space of style-aware symbolic music representations by adversarial autoencoders. arXiv preprint arXiv:2001.05494.

Wang, S., Liu, W., Wu, J., Cao, L., Meng, Q. and Kennedy, P.J., 2016, July. Training deep neural networks on imbalanced data sets. In 2016 international joint conference on neural networks (IJCNN) (pp. 4368-4374). IEEE.

Watanabe, J., 2008. System generating jazz-style chord sequences for solo piano. Proc. ICMPC10, 2008.

Xiao, H., Rasul, K. and Vollgraf, R., 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747.

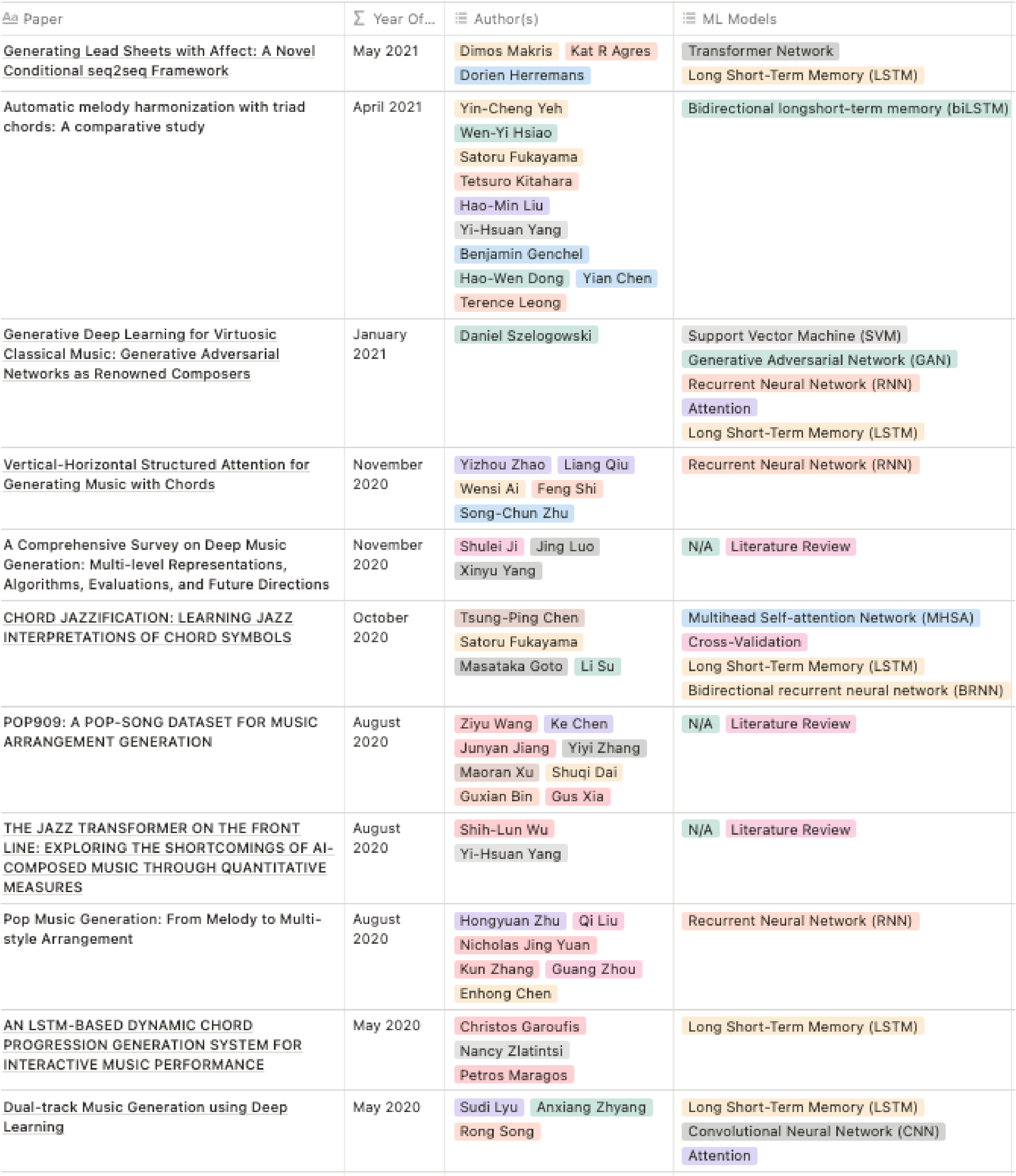
Zhao, Y., Qiu, L., Ai, W., Shi, F. and Zhu, S.C., 2020. Vertical-Horizontal Structured Attention for Generating Music with Chords. arXiv preprint arXiv:2011.09078.

Zhu, H., Liu, Q., Yuan, N.J., Zhang, K., Zhou, G. and Chen, E., 2020. Pop music generation: From melody to multi-style arrangement. ACM Transactions on Knowledge Discovery from Data (TKDD), 14(5), pp.1-31.

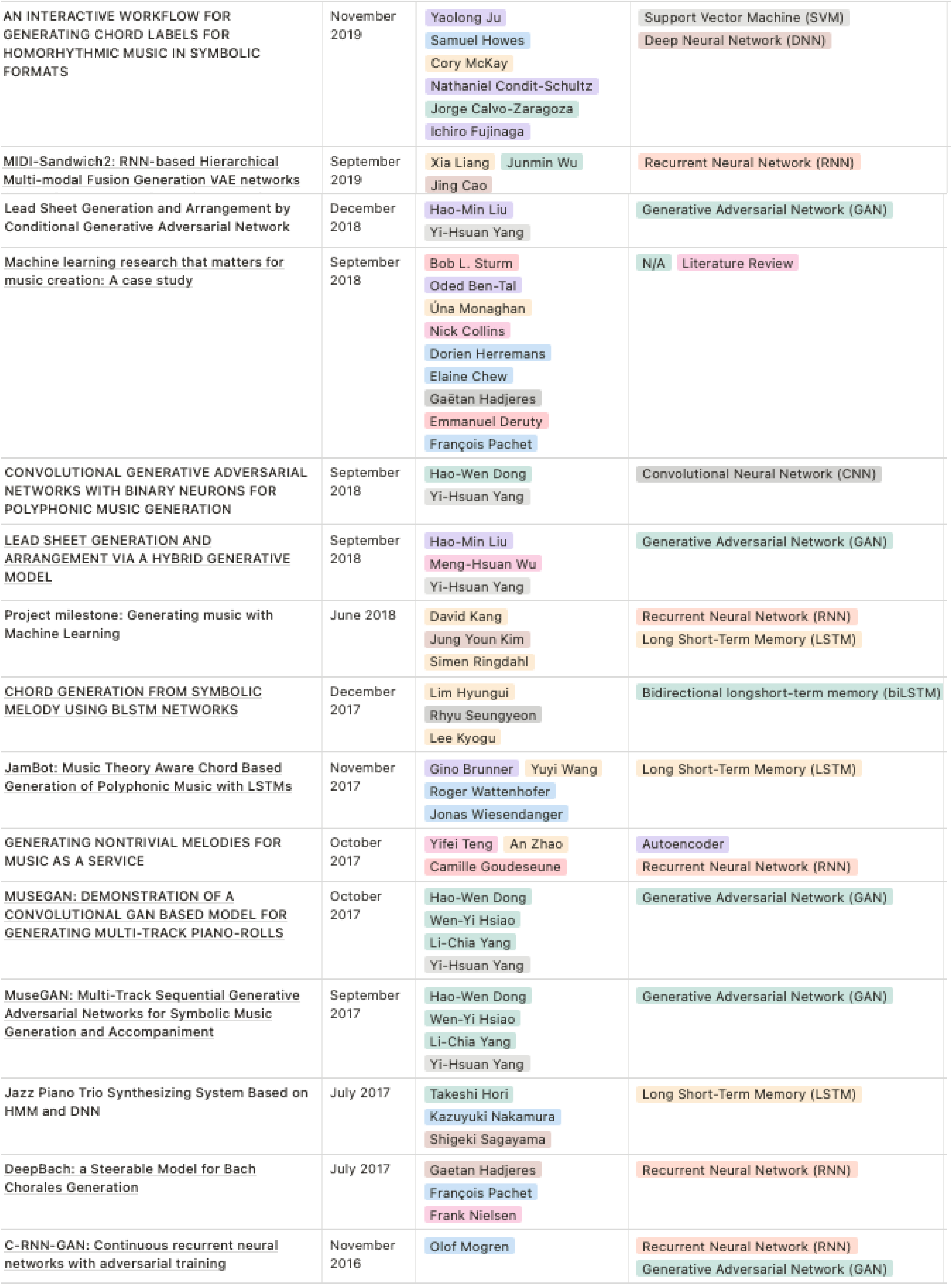
Appendices

Appendix A ↩

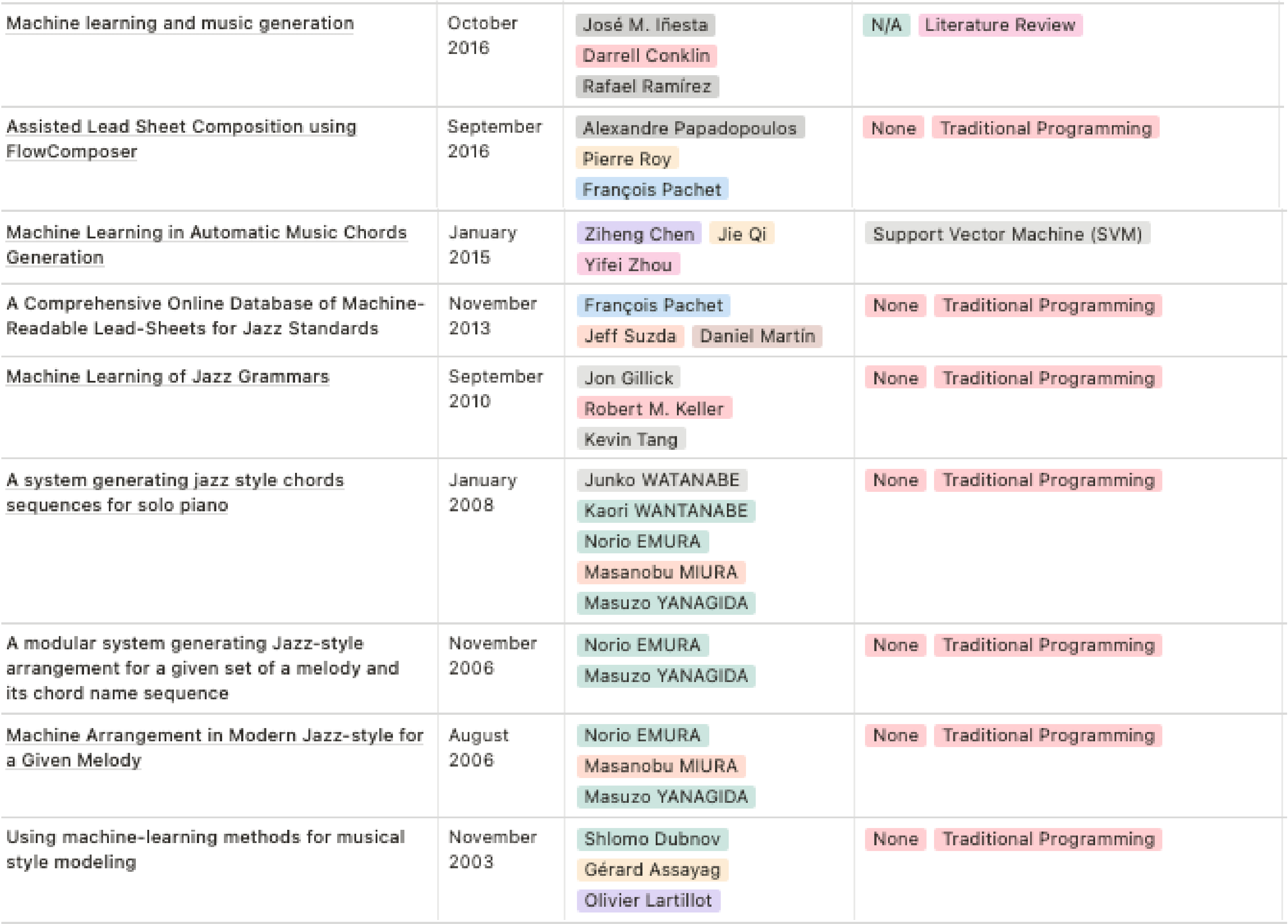
Table showing 35 deep learning in music research papers (full version)



Aa Paper	Σ Year Of...	☰ Author(s)	☰ ML Models
Generating Lead Sheets with Affect: A Novel Conditional seq2seq Framework	May 2021	Dimos Makris Kat R Agres Dorien Herremans	Transformer Network Long Short-Term Memory (LSTM)
Automatic melody harmonization with triad chords: A comparative study	April 2021	Yin-Cheng Yeh Wen-Yi Hsiao Satoru Fukayama Tetsuro Kitahara Hao-Min Liu Yi-Hsuan Yang Benjamin Genchel Hao-Wen Dong Yian Chen Terence Leong	Bidirectional longshort-term memory (biLSTM)
Generative Deep Learning for Virtuosic Classical Music: Generative Adversarial Networks as Renowned Composers	January 2021	Daniel Szelogowski	Support Vector Machine (SVM) Generative Adversarial Network (GAN) Recurrent Neural Network (RNN) Attention Long Short-Term Memory (LSTM)
Vertical-Horizontal Structured Attention for Generating Music with Chords	November 2020	Yizhou Zhao Liang Qiu Wensi Ai Feng Shi Song-Chun Zhu	Recurrent Neural Network (RNN)
A Comprehensive Survey on Deep Music Generation: Multi-level Representations, Algorithms, Evaluations, and Future Directions	November 2020	Shulei Ji Jing Luo Xinyu Yang	N/A Literature Review
CHORD JAZZIFICATION: LEARNING JAZZ INTERPRETATIONS OF CHORD SYMBOLS	October 2020	Tsung-Ping Chen Satoru Fukayama Masataka Goto Li Su	Multihead Self-attention Network (MHSA) Cross-Validation Long Short-Term Memory (LSTM) Bidirectional recurrent neural network (BRNN)
POP909: A POP-SONG DATASET FOR MUSIC ARRANGEMENT GENERATION	August 2020	Ziyu Wang Ke Chen Junyan Jiang Yiyi Zhang Maoran Xu Shuqi Dai Guxian Bin Gus Xia	N/A Literature Review
THE JAZZ TRANSFORMER ON THE FRONT LINE: EXPLORING THE SHORTCOMINGS OF AI-COMPOSED MUSIC THROUGH QUANTITATIVE MEASURES	August 2020	Shih-Lun Wu Yi-Hsuan Yang	N/A Literature Review
Pop Music Generation: From Melody to Multi-style Arrangement	August 2020	Hongyuan Zhu Qi Liu Nicholas Jing Yuan Kun Zhang Guang Zhou Enhong Chen	Recurrent Neural Network (RNN)
AN LSTM-BASED DYNAMIC CHORD PROGRESSION GENERATION SYSTEM FOR INTERACTIVE MUSIC PERFORMANCE	May 2020	Christos Garoufis Nancy Zlatintsi Petros Maragos	Long Short-Term Memory (LSTM)
Dual-track Music Generation using Deep Learning	May 2020	Sudi Lyu Anxiang Zhyang Rong Song	Long Short-Term Memory (LSTM) Convolutional Neural Network (CNN) Attention



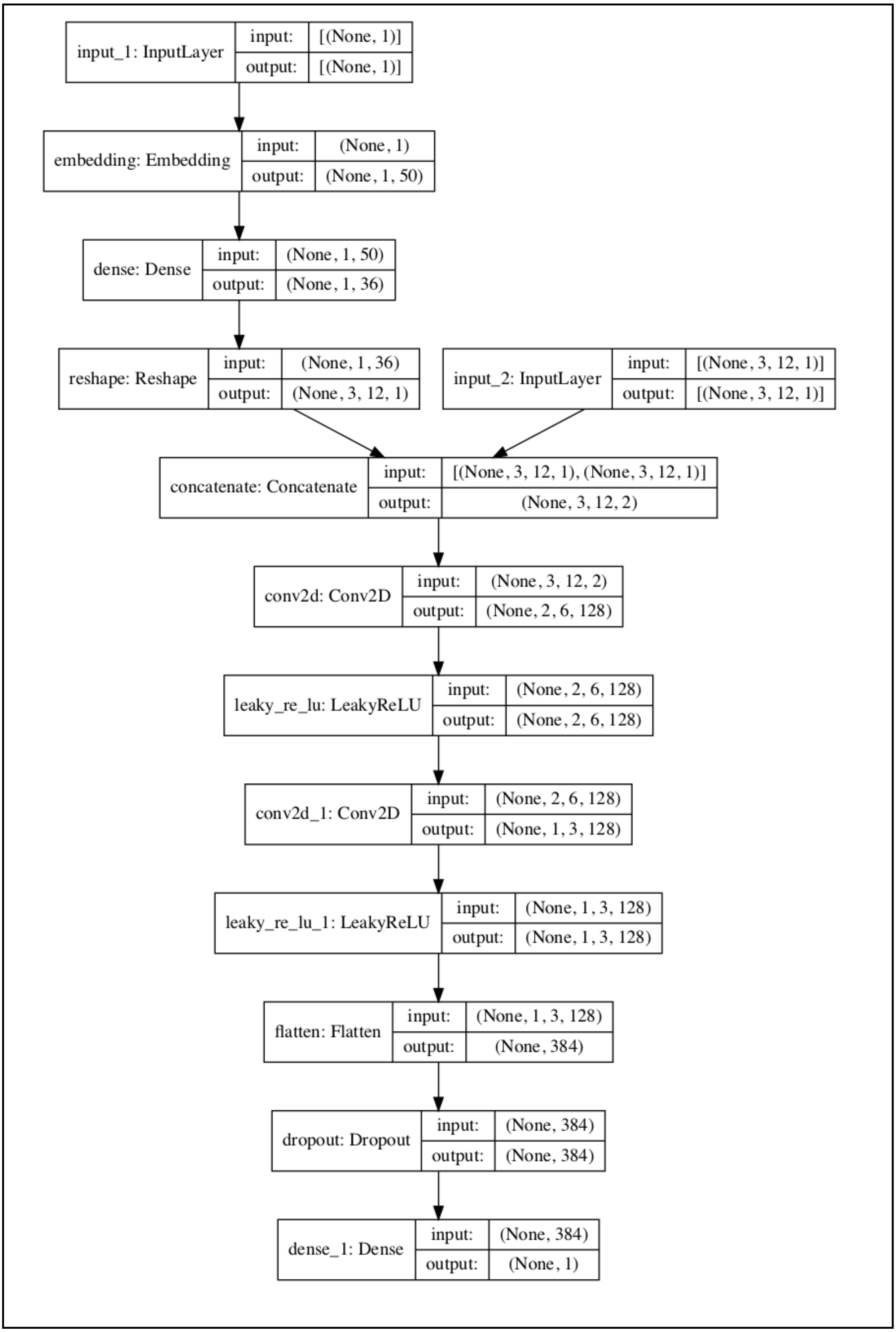
AN INTERACTIVE WORKFLOW FOR GENERATING CHORD LABELS FOR HOMORHYTHMIC MUSIC IN SYMBOLIC FORMATS	November 2019	Yaolong Ju Samuel Howes Cory McKay Nathaniel Condit-Schultz Jorge Calvo-Zaragoza Ichiro Fujinaga	Support Vector Machine (SVM) Deep Neural Network (DNN)
MIDI-Sandwich2: RNN-based Hierarchical Multi-modal Fusion Generation VAE networks	September 2019	Xia Liang Junmin Wu Jing Cao	Recurrent Neural Network (RNN)
Lead Sheet Generation and Arrangement by Conditional Generative Adversarial Network	December 2018	Hao-Min Liu Yi-Hsuan Yang	Generative Adversarial Network (GAN)
Machine learning research that matters for music creation: A case study	September 2018	Bob L. Sturm Oded Ben-Tal Úna Monaghan Nick Collins Dorien Herremans Elaine Chew Gaëtan Hadjeres Emmanuel Deruty François Pachet	N/A Literature Review
CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS WITH BINARY NEURONS FOR POLYPHONIC MUSIC GENERATION	September 2018	Hao-Wen Dong Yi-Hsuan Yang	Convolutional Neural Network (CNN)
LEAD SHEET GENERATION AND ARRANGEMENT VIA A HYBRID GENERATIVE MODEL	September 2018	Hao-Min Liu Meng-Hsuan Wu Yi-Hsuan Yang	Generative Adversarial Network (GAN)
Project milestone: Generating music with Machine Learning	June 2018	David Kang Jung Youn Kim Simen Ringdahl	Recurrent Neural Network (RNN) Long Short-Term Memory (LSTM)
CHORD GENERATION FROM SYMBOLIC MELODY USING BLSTM NETWORKS	December 2017	Lim Hyungui Rhyu Seungyeon Lee Kyogu	Bidirectional longshort-term memory (biLSTM)
JamBot: Music Theory Aware Chord Based Generation of Polyphonic Music with LSTMs	November 2017	Gino Brunner Yuyi Wang Roger Wattenhofer Jonas Wiesendanger	Long Short-Term Memory (LSTM)
GENERATING NONTRIVIAL MELODIES FOR MUSIC AS A SERVICE	October 2017	Yifei Teng An Zhao Camille Goudeseune	Autoencoder Recurrent Neural Network (RNN)
MUSEGAN: DEMONSTRATION OF A CONVOLUTIONAL GAN BASED MODEL FOR GENERATING MULTI-TRACK PIANO-ROLLS	October 2017	Hao-Wen Dong Wen-Yi Hsiao Li-Chia Yang Yi-Hsuan Yang	Generative Adversarial Network (GAN)
MuseGAN: Multi-Track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment	September 2017	Hao-Wen Dong Wen-Yi Hsiao Li-Chia Yang Yi-Hsuan Yang	Generative Adversarial Network (GAN)
Jazz Piano Trio Synthesizing System Based on HMM and DNN	July 2017	Takeshi Hori Kazuyuki Nakamura Shigeki Sagayama	Long Short-Term Memory (LSTM)
DeepBach: a Steerable Model for Bach Chorales Generation	July 2017	Gaetan Hadjeres François Pachet Frank Nielsen	Recurrent Neural Network (RNN)
C-RNN-GAN: Continuous recurrent neural networks with adversarial training	November 2016	Olof Mogren	Recurrent Neural Network (RNN) Generative Adversarial Network (GAN)



Machine learning and music generation	October 2016	José M. Iñesta Darrell Conklin Rafael Ramírez	N/A Literature Review
Assisted Lead Sheet Composition using FlowComposer	September 2016	Alexandre Papadopoulos Pierre Roy François Pachet	None Traditional Programming
Machine Learning in Automatic Music Chords Generation	January 2015	Ziheng Chen Jie Qi Yifei Zhou	Support Vector Machine (SVM)
A Comprehensive Online Database of Machine-Readable Lead-Sheets for Jazz Standards	November 2013	François Pachet Jeff Suzda Daniel Martín	None Traditional Programming
Machine Learning of Jazz Grammars	September 2010	Jon Gillick Robert M. Keller Kevin Tang	None Traditional Programming
A system generating jazz style chords sequences for solo piano	January 2008	Junko WATANABE Kaori WANTANABE Norio EMURA Masanobu MIURA Masuzo YANAGIDA	None Traditional Programming
A modular system generating Jazz-style arrangement for a given set of a melody and its chord name sequence	November 2006	Norio EMURA Masuzo YANAGIDA	None Traditional Programming
Machine Arrangement in Modern Jazz-style for a Given Melody	August 2006	Norio EMURA Masanobu MIURA Masuzo YANAGIDA	None Traditional Programming
Using machine-learning methods for musical style modeling	November 2003	Shlomo Dubnov Gérard Assayag Olivier Lartillot	None Traditional Programming

Appendix B↵

The discriminator network architecture from the adapted cDCGAN model


input_1: InputLayer | input: [(None, 1)] | output: [(None, 1)]
embedding: Embedding | input: (None, 1) | output: (None, 1, 50)
dense: Dense | input: (None, 1, 50) | output: (None, 1, 36)
reshape: Reshape | input: (None, 1, 36) | output: (None, 3, 12, 1)
input_2: InputLayer | input: [(None, 3, 12, 1)] | output: [(None, 3, 12, 1)]
concatenate: Concatenate | input: [(None, 3, 12, 1), (None, 3, 12, 1)] | output: (None, 3, 12, 2)
conv2d: Conv2D | input: (None, 3, 12, 2) | output: (None, 2, 6, 128)
leaky_re_lu: LeakyReLU | input: (None, 2, 6, 128) | output: (None, 2, 6, 128)
conv2d_1: Conv2D | input: (None, 2, 6, 128) | output: (None, 1, 3, 128)
leaky_re_lu_1: LeakyReLU | input: (None, 1, 3, 128) | output: (None, 1, 3, 128)
flatten: Flatten | input: (None, 1, 3, 128) | output: (None, 384)
dropout: Dropout | input: (None, 384) | output: (None, 384)
dense_1: Dense | input: (None, 384) | output: (None, 1)