­

Jazz Piano Lead Sheet Arrangement via Deep Learning

Elliot Hogg

200817750

David Herbert

Mark Turner

27/08/2021

18669 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 represented in the lead sheet format – which is an abbreviated representation. 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 (C-GAN) 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 and participant led 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 1](#_Toc80280732)

[1.1 Aim 2](#_Toc80280733)

[1.2 Objectives 3](#_Toc80280734)

[1.3 Structure 3](#_Toc80280735)

[1.4 Source Code 4](#_Toc80280736)

[2. Background Research 5](#_Toc80280737)

[2.1 Motivations 5](#_Toc80280738)

[2.2 Generative Deep Learning Models 5](#_Toc80280739)

[2.3 Generative Adversarial Networks (GANs) 7](#_Toc80280740)

[2.5 Conditional Generative Adversarial Networks (C-GANs) 7](#_Toc80280741)

[2.6 Available Datasets 8](#_Toc80280742)

[2.7 Existing Chord Scraping Tools 9](#_Toc80280743)

[2.8 Technologies 9](#_Toc80280744)

[3. Gathering a Library of Jazz Piano Solo Arrangements 10](#_Toc80280745)

[3.1 Initial Research 10](#_Toc80280746)

[3.2 Fully Arranged Jazz Piano Solo Library 10](#_Toc80280747)

[3.3 MusicXML 11](#_Toc80280748)

[4. Chord Scraper 12](#_Toc80280749)

[4.1 System Requirements 12](#_Toc80280750)

[4.2 System Overview 12](#_Toc80280751)

[4.3 Development 15](#_Toc80280752)

[4.3.1 Chord Extraction 15](#_Toc80280753)

[4.3.2 Chord Data Manipulation 18](#_Toc80280754)

[4.4. Results and Evaluation 19](#_Toc80280755)

[4.4.1 Jazz-Chords Dataset 19](#_Toc80280756)

[4.4.2 Evaluation of Chord Scraper’s Performance 27](#_Toc80280757)

[5. Chord Generator 29](#_Toc80280758)

[5.1 Objectives 29](#_Toc80280759)

[5.2 Preparing the Training Data 29](#_Toc80280760)

[5.3 Classification Task 34](#_Toc80280761)

[5.3.1 Deep Feedforward Network 35](#_Toc80280762)

[5.3.2 Convolutional Neural Network 38](#_Toc80280763)

[5.3.3 Results 41](#_Toc80280764)

[5.4 Chord Voicing Generation 42](#_Toc80280765)

[5.4.1 Developing the Conditional Generative Adversarial Network (C-GAN) 42](#_Toc80280766)

[5.4.2 Adapting the cDCGAN Model 44](#_Toc80280767)

[5.4.3 Optimising the cDCGAN Model 44](#_Toc80280768)

[5.4.4 Results 48](#_Toc80280769)

[6. Leadsheet Arrangement System (LSAS) 50](#_Toc80280770)

[6.1 Development 50](#_Toc80280771)

[6.1.1 data\_extractor 50](#_Toc80280772)

[6.1.2 chord\_generator 52](#_Toc80280773)

[6.1.3 leadsheet\_arranger 54](#_Toc80280774)

[6.2 Results and Evaluation 57](#_Toc80280775)

[6.2.1 Capabilities 58](#_Toc80280776)

[6.2.2 Limitations 59](#_Toc80280777)

[6.2.3 Selected Arrangements for Evaluation 59](#_Toc80280778)

[6.2.4 Technical Evaluation 60](#_Toc80280779)

[6.2.5 Participant led Evaluation 61](#_Toc80280780)

[7. Conclusion 62](#_Toc80280781)

[References 65](#_Toc80280782)

[Appendices 68](#_Toc80280783)

[Appendix A 68](#_Toc80280784)

# 1. Introduction

Lead sheet arrangement is a task whereby a musician takes an abbreviated representation of a song and adds harmonic and rhymical 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](https://en.wikipedia.org/wiki/Musical_notation#Modern_staff_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.

A black and white photo of a piano

Description automatically generated with low confidence

**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.

Diagram, engineering drawing

Description automatically generated

**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. Much of the literature presented conditional decision-based algorithms pre-programmed with a set of 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 led 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 of 8-bar segments of pop music lead sheets. Although the proposed system does not meet the requirements of jazz musicians, 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 that can be used for further chord data extraction tasks. The project will present a generative deep learning model that can be trained with a chord voicing 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 a 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:

* To gather a library of machine readable, fully arranged pieces of jazz piano music which have annotated chord symbols
* To develop a program that can scrape chord voicings and their associated chord symbols from fully arranged pieces of piano music in MusicXML format
* To present a novel dataset of jazz piano chord symbol-chord voicing pairs
* To create a generative deep learning model that can generate accurate and varied chord voicings for a given chord symbol
* To train the generative model using the novel dataset
* 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 an 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 on 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 solo piano music library of 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 the how initial aims and objectives were met, how the project fits within the 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.

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 embedding scripts and deep learning models including the trained cDC-GAN model developed in section 5.

And the *leadsheet\_arranger* houses the overarching Lead Sheet Arrangement System (LSAS) outlined in section 6. Please refer to the subdirectory readme files for instructions on how to execute the code.

# 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 in 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](#_Appendix_A_1).

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 lead sheets and then arrange them harmonically. As the title suggests, the model used was a conditional generative adversarial network or C-GAN. 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 a given chord symbol.

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 dataset of pop arrangements. In order to gain further insight into the C-GAN model architecture, the project’s source code was downloaded from GitHub. However, attempts to train and run the model failed. This was due 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 types of C-GANs or generative adversarial networks (GANs) in the task of music generation (Dong et al., 2018; Liu et al., 2018; Yang et al., 2017). 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 model using a dataset of pop songs arranged for a 4-piece band. The trained model was able to generate new, unique 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 numerical labels and encoding chord voicings as binary vectors. This approach provided a useful framework for which this research project could encode its own data.

The model used to generate the chords voicings were variations of recurrent neural networks (RNNs). 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 6 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; Szelogowski 2021; Teng et al., 2017; Zhao et al., 2020; Zhu et al., 2020;). The metainformation of these papers can be found in the table at [Appendix A](#_Appendix_A_1).

This suggests that RNNs would not be suitable to the requirements of this project as the sheer amount of 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. 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 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 struggles 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 C-GAN model would be the most effective deep learning model for this project. In order to learn more about C-GANs and their use in generation tasks, a wider scope of literature was searched. There were two papers of note which both used C-GANs to perform tasks that were fundamentally the same as harmonic lead sheet generation.

The first paper was titled ‘Conditional generative adversarial nets’ (Mirza & Osindero, 2014). The authors proposed the C-GAN 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 C-GAN model was trained using the MNIST handwritten digit dataset (Deng, 2012). The trained model 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 C-GAN architecture to a number of more advanced generation tasks. Radford et al.’s C-GAN 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 distribution space as the training images.

The results of these papers suggest that a C-GAN model could be used to conditionally generate chord voicings if trained on a large database 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 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 (C-GANs).

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. 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 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 a random number as input, 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 fake images that were indistinguishable from 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 is unable to effectively classify real instances as real, and fake instances as fake (Goodfellow et al., 2014).

## 2.5 Conditional Generative Adversarial Networks (C-GANs)

Conditional generative adversarial networks (C-GANs) allow the output of the generator model to be conditioned on a given input (Mirza & Osindero, 2014). An example of this would be a C-GAN 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 C-GAN model, the numerical labels would be given as an additional input to both the generator and discriminator. When the trained generator model was given one 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 original dataset.

## 2.6 Available Datasets

Deep learning generation models require a large amount of data for training (Goodfellow et al., 2014). C-GANs 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 C-GAN model in this project, an extensive survey of available 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-jazz 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** A list of 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 is the only existing dataset that contains chords presented as pairs of chord symbols and chord voicings. The only limitation is that most of the chords are taken from pop chords, and do 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 C-GAN. Gathering the dataset would involve scraping fully arranged jazz piano solos which had chord symbol annotations.

## 2.7 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.

## 2.8 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 images 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 C-GAN’s require 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.

Diagram

Description automatically generated with medium confidence

A picture containing table

Description automatically generated

**Figure 3.1** A 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 2.6 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 jazz piano solos 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. As MIDI files cannot contain chord symbols (reference), 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 chord symbols. The rest 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 below.

## 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 and bass clef 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. A detailed list on library can be found at [Appendix B](#_Appendix_B).

## 3.3 MusicXML

In order to contextualise the following section, 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 below shows how the first chord symbol and voicing pair in figure 3.1 are represented in MusicXML format.

Looking at figure 3.2, the chord symbol is represented by the use of a harmony element. The inner elements indicate the bottom note in the chord voicing, and the chord type. The harmony element can contain additional elements for more complex chords.

The notes that make up the chord voicing are represented by the note elements that proceed the harmony element. The “default-x” attribute is helpful for determining which notes make up a chord voicing, as the notes that make up a chord voicing are often aligned vertically.

<harmony print-frame="no">

<root>

<root-step>C</root-step>

</root>

<kind text="M7">major-seventh</kind>

</harmony>

<note default-x="92.47" default-y="-40.00">

<pitch>

<step>E</step>

<octave>4</octave>

</pitch>

<duration>4</duration>

<voice>1</voice>

<type>whole</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** A C Major 7th chord symbol and notated voicing 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. Sections 3.2 and 3.3 are recommended precursors to this section.

## 4.1 System Requirements

- To extract chord symbol-chord voicing pairs from MusicXML solo piano scores.

- To output raw data as well as data that has been partially cleaned and refined (see section 4.1.2)

- To work on all piano solos in MusicXML format including previously unseen solos

-To be able to scrape a large number of scores (>100) in a reasonable amount of time (< 5 minutes)

- To extract the group of notes that represent a chord symbol with a high degree of accuracy (> 97%)

- To have a highly usable command line interface that enables other researchers to use the tool.

## 4.2 System Overview

As seen 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 as the second transforms the chords into a structure that is more suited to machine learning model inputs. Examples of these two input types are explored below and displayed in figures 4.4 and 4.5.

Diagram

Description automatically generated

**Figure 4.1** A 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 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 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 label and MusicXML harmony element, and labels 2 & 12-15 point to the notes and MusicXML 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 data format seen in figure 4.4 stores chord symbols in the *root, type,* & *extensions* columns. For example, the first line of data shows that the root is G, and the type is minor-seventh. The raw 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 data seen in figure 4.5 represents the raw chord data after undergoing two data processing steps. Firstly, all of the substandard chord pairs are removed. 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.

Diagram, schematic

Description automatically generated

**Figure 4.2** A 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** A 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’:9,'alter':-1,[Gb2,Bb2,C3,E3,Bb4] [22,26,28,32,50]

'type':'add'}]

F dominant [] [F2,A2,C3,Eb3,A4] [21,25,28,31,49]

**Figure 4.4** Raw chord data representing score in figure 4.2 (CSV format)

**type extensions note\_numbers**

minor-seventh [] [16,19,27,26,45]

dominant [{'degree':9,'alter':-1,'type':'add'}] [22,26,28,32,50]

dominant [] [16,20,23,26,44]

**Figure 4.5** Cleaned and transposed chord data representing chords in figure 4.2 (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 would be written in order to transform the extracted 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.

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 before the bass clef 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 chord voicings was 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 could be found. For example, looking at figure 4.6, the “default-x” attribute of note *e + 1* could be used to locate and extract notes *e+9* to *e+12.*

The issue with this approach is that there are cases in which some of the notes in a chord voicing are offset on the *x* axis.

The first cause of this is when there are 2 notes are next to each other, and vertically aligning them would cause the notes to overlap, and thus be unreadable. Figure 4.7 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 other notes in the voicing.

Diagram, schematic

Description automatically generated

**Figure 4.7** A score highlighting the notes selected to make up a chord voicing using “x-location” value.

The second cause of this issue is when chord voicings are temporally arranged to form a rhythmic pattern. This means that the notes that make up the chord voicing are not aligned on the x-axis, but adjacent to one another. This is highlighted in figure 4.8, in which only two of the 5 notes in the chord voicing have the same “default-x” attribute.

Diagram, schematic

Description automatically generated

**Figure 4.8** A chord voicing in which the notes are arranged in a rhythmic pattern.

In order to solve the first issue, a 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” of 100, and the *x-axis deviation* 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 deviation* value of 20 yielded the best results. For the first issue of closely positioned notes, the *x-axis deviation* 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 4.9.

Diagram, schematic

Description automatically generated

**Figure 4.9** The use of *x-axis deviation* to extracted all of the notes in a chord voicing.

It was not possible to use *x-axis deviation* to successfully extract chord voicings arranged in a rhythmic pattern. For example, an *x-axis deviation* value that could extract rhythmically arranged chord voicings would result in a majority of use cases in which unwanted notes were extracted.

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 would then be able to be traversed in order to find and extract 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 oppose 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 4.10. The technique 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.

Diagram

Description automatically generated

**Figure 4.10** MusicXML as an Ordered Tree Data Structure

### 4.3.2 Chord Data Manipulation

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.

Also, the notes of each chord voicing were additionally represented using note numbers, a system in which the numbers 1-88 represent each note on the piano. Outputting chord voicings in this way would allow other users to easily encode the data for further use. The chord voicings were also subjected to a small degree of pitch normalisation, in which any very high or low notes were moved further towards the middle of the piano.

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 above the note C. This then 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 major-sevenths. Therefore, when using the dataset to train a machine learning model, there would be much more examples of each chord type. There is also no downside to this approach, as any generated chords can be transposed back to the root note of the user’s choice 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 easy. 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.

## 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 will also be evaluated.

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

### 4.4.1 Jazz-Chords Dataset

To download the dataset, please go to (github link)

In order to gather the jazz-chords dataset, a chord scraper 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 and 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 item 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 4.11 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

Chart, bar chart

Description automatically generated

**Figure 4.11** 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-seventh, and major.

Jazz musical theory defines a standardised set of possible notes that can be included in chord voicings. For example, the chord C major-seventh can contain the following notes – C, E, G, B. These notes make up the standardised base voicing of a C major seventh chord. As mentioned in section 1, 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. 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.

In order for the presented dataset to be of use, its contained chord voicings need to have a high degree of accuracy, and therefore cannot contain any incorrect notes. The four 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 4.12 - 4.15 show the note distribution of all of the chord voicings for each chord type.

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

**Table 4.3** Incorrect notes for 4 chord types

Chart, bar chart

Description automatically generated

**Figure 4.12** A stacked bar-chart showing the total note distribution for all dominant chord voicings

Chart, bar chart

Description automatically generated

**Figure 4.13** A stacked bar-chart showing the total note distribution for all minor-seventh

chord voicings

Chart, bar chart

Description automatically generated

**Figure 4.15** A stacked bar-chart showing the total note distribution for all major chord voicings

Chart, bar chart

Description automatically generated

**Figure 4.14** A stacked bar-chart showing the total note distribution for all major-seventh chord voicings

As seen in each stacked bar-chart, some unwanted notes are present in each of the chord types. 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** A table showing chord the jazz-chords dataset chord voicing accuracy

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 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 clearly concluded that, for the most part, the chord scraper met all of its requirements. It was able to successfully extract pair of chord symbols and chord voicings. It was able to output both raw and refined datasets. The system was able to successfully scrape 166 unseen MusicXML files, having been designed using 5 test files. This suggests it will be able to successfully scrape further unseen MusicXML files. The system was also able to scrape all 171 MusicXML files in around 40 seconds. As mentioned in section 4.3, the chord scraper also features a CLI.

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 voicings were manually compared with the MusicXML file from which they were obtained.

A total of 94 major chord pairs were randomly selected from the Jazz-Chords dataset for manual comparison. The results of this comparison showed that 100% 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 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 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 dataset. 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.

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 not all notes could 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 (C-GAN) 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 prepared for use as training data for the C-GAN model. The section will then present some initial deep learning experiments that served as a precursor to developing the C-GAN model. Following this, the development and implementation of the C-GAN is presented. Finally, the results and capabilities of the C-GAN model are presented and evaluated.

Sections 2.3, 2.4, and 2.5 and recommended precursors to this section. All models were trained using a Nvidia Tesla v100 GPU.

## 5.1 Objectives

- To prepare and encode the Jazz-Chords dataset for use as training data for the C-GAN model

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

- To report and evaluate the results of each experiment

- To implement a C-GAN capable of generating accurate, musical, and varied 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. A C-GAN model capable of generating jazz chord voicings needs a large dataset of chord label-chord voicing pairs. For example, for a C-GAN 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 2.5, C-GAN’s are capable of generating multiple different types of output, meaning that they can be trained to generate multiple different types of chord voicings.

Looking at the chord distribution of the jazz-chords dataset in table 4.1 on page 21, there are only 4 chord types that have enough chord voicings - dominant, minor-seventh, major, and major-seventh.

The four chord types also have differing numbers of voicings, meaning that the distribution of the overall data is imbalanced. Training the C-GAN 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 in the training data. 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 an equally distributed dataset.

As presented in section 4.4.1, each of the chord type’s voicings contained a small amount of inaccuracy. These inaccuracies were removed before the data was used to train the C-GAN model. In order to do so, all of the unwanted notes seen in table 4.3, page 22, were programmatically removed from all of their associated chord type’s voicings. This involved iterating through each of the chord types 818 voicings and removing any inaccurate notes. 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. The resulting refined dataset is presented in table 5.1 and figures 5.1and 5.2

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

**Table 5.1** A table showing the chord label distribution of the training data

Chart, bar chart

Description automatically generated

**Figure 5.1** Dominant and Minor-seventh note distributions from the Refined Jazz-Chords dataset

Chart, bar chart

Description automatically generated

**Figure 5.2** Major and Major-seventh note distributions from the Refined Jazz-Chords dataset

Looking at the note distributions above, all chord voicings now have no unwanted notes, and no chords have notes that are F#4 and above.

The final step in preparing the training data was to represent the chord labels and chord voicings in a format that the C-GAN could comprehend. This process is called data embedding or data encoding. Initially, a simple integer encoding was applied to the chord labels; this can be seen in table 5.2. The input layers of the C-GAN could further encode these integer labels.

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

**Table 5.2** A table showing 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).

Two different chord voicing embeddings 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 . 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 5.5 highlights this embedding.

Chart, box and whisker chart

Description automatically generated

**Figure 5.5** Chord voicing embedded as an 88 binary number vector

The second embedding approach has 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 contain these notes, they can be discarded. Thus, the voicings are embedded in 7x12 matrices. Figure 5.6 shows this embedding.

A picture containing diagram

Description automatically generated

**Figure 5.6** Chord voicing embedded as 12 x 7 binary number matrix

In order to determine which of these encodings better facilitates the mapping of a mathematical relationship between chord label and chord voicing, a classification task was conducted. This is presented in the next section.

## 5.3 Classification Task

As explained in sections 2.4 and 2.5, C-GAN’s consist of 2 independent neural networks. The architecture of these networks depends on the type of data that is being generated. For example, in the task of image generation, convolutional neural networks (CNN’s) are generally used for both the generator and discriminator. CNN’s have layers that apply convolution to input images in order to extract patterns such as edges. These patterns can be used to learn unobservable mathematical relationships (reference). If the C-GAN is tasked with generating 1D vectors, then a simpler Deep Feedforward Network (DFN) is generally more suitable. DFN’s are “vanilla” neural networks and consist of layers of fully connected neurons. Each of these neurons has a weight and a bias. Through training, the values of these parameters self-adjust in order to achieve the desired outputs for given inputs (reference).

As presented in the above section, the chord voicings were encoded as both 1D vectors and as 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 will be used in a classification task experiment. Classification network architectures are much simpler than C-GAN’s and offer a simple but effective way to test the performance of the differing chord voicing embeddings. The classification networks will take as input an unlabelled chord voicing and output a chord label prediction.

To test the 1D vector encoding, a DFN classification network will be constructed. To test the binary image encoding, a CNN classification network will be constructed. Both networks will be optimised equally to try to ensure that any difference in performance is a result of the chord embeddings.

By comparing the results of this experiment, the most effective chord voicing embedding approach can be determined.

### 5.3.1 Deep Feedforward Network

The initial network configuration was taken from a vanilla DFN found on the TensorFlow website (citation). Then, in order to find the ideal network architecture, Goodfellows (2016) principle was used – “The ideal network architecture for a task must be found via experimentation guided by monitoring the validation set error.”

The initial 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 output gave a prediction of the inputted chord voicings chord-label input. For example, an output of [1, 0, 0,0] would mean that the model predicted the input as being a dominant chord.

To train the model, two thirds of the dataset was given to the model over a number of training iterations called epochs. The remaining one third of the dataset was then used to test the model’s accuracy in making predictions. The initial model had a test set accuracy of 60%, which seemed unusually low. The total loss of the model in respect to the test data was also high, at 0.6. A large total loss value means that inaccurate predictions are highly inaccurate, and usually indicates that there is some issue with the training dataset (reference). Common issues such as unclean or unbalanced data could be ruled out. However, a chapter in Goodfellow’s book suggested that high total loss in classification networks could be due two or more sets of labelled data having overlapping distributions to the point where the network struggled to tell them apart. In order to plot and examine the distributions of the chord voicings, a function was created that iterated through each chord voicing in the dataset and counted the number of major and minor notes it contained.

For context, there a certain notes on the piano that are defined as either major and minor based on the type of sound they make. In musical theory, major notes are used in major chords, and minor notes are used in minor chords. Some chords such as dominant chords used a mixture of major and minor.

By plotting each chord voicings major and minor note count, it could be seen that the “major” and “major-seventh” chord voicings had very similar distributions. This can be seen in figure 5.7.

How does a point gets its position on plot?

Chart, scatter chart

Description automatically generated

**Figure 5.7** A scatter plot showing major and minor note distributions of the datasets chord voicings

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

|  |  |
| --- | --- |
| ***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 the 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 voicings.

In order to improve the accuracy of the model, further dense layers were incrementally added whilst observing the test set accuracy. 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 (reference). This prevents co-adaptation (glossary) and leads to a more balanced network that has a higher likelihood of generalising. The resulting DFN’s architecture is presented in figure 5.8.

Each layer had a Rectified Linear Unit (RELU) activation function, which outputs all negative values as 0 whilst leaving positive values unaffected. The model has a Sparse Categorical Cross-entropy loss function, which computes the cross-entropy loss between the labels and predictions. An Adam algorithm was used to optimize the network, which is a stochastic gradient descent method (reference). The loss function and optimizer are used to update the networks parameters in a direction which reduces the loss of the network (reference). The batch size of each training epoch was initially set to 32, and increased in powers of 2 to 1028, which was found to be optimal. Figures 5.9 and 5.10 show the accuracy and loss curves of training the network for 300 epochs. 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. These two points indicate the beginning of the model overfitting.

Diagram

Description automatically generated

**Figure 5.8** The DFN architecture used in the classification task

Graphical user interface, chart

Description automatically generated

**Figure 5.9** The accuracy of the DFN after training for 300 epochs

Chart, line chart

Description automatically generated

**Figure 5.10** The total loss of the DFN after training for 300 epochs

The final iteration of the model was trained for a total of 45 epochs. The results can be seen in table 5.4.

|  |  |
| --- | --- |
| ***Test Accuracy*** | 0.8617 |
| ***Training Accuracy*** | 0.8631 |
| ***Test Loss*** | 0.3435 |
| ***Training Loss*** | 0.3280 |

**Table 5.4** The results of the final iteration of the DFN

This experiment has provided insight into the performance of representing the chord voicings as 88-note binary number vectors. The performance can now be compared with that of the chord matrices to determine which embedding to design the c-gan around. The experiment has also helped to develop a DFN architecture that can be used for both the discriminator and generator in the c-gan if it is decided that the chord voicings will be embedded as 88-note vectors.

### 5.3.2 Convolutional Neural Network

As mentioned in the previous section, CNN’s are used for machine learning tasks that involve images. They contain layers that apply convolution to the input images in order to extract patterns that can be used to learn unseen mathematical relationships between inputs and outputs (reference). The majority of previous research in the field of music and deep generation represents chord voicings as 1D arrays, like in the previous section. However, the presence of a repeating pattern on the piano makes the encoding of chord voicings as 2D arrays seem intuitive. The result is essentially a binary image; some examples can be seen in figure 5.10. By creating and optimising a CNN and performing a classification task, it will be determined whether or not this 2D encoding is more meaningfully understood by the computer than its 1D alternative.

Diagram

Description automatically generated

**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 (reference). The output of the last max pooling layer is then flattened to a 1D vector and fed through a fully connected dense layer. This allows for a 3-neuron output layer, which indicates the model’s classification prediction.

As with the DFN described above, Goodfellows principle will be followed to improve the performance of the CNN through the guidance of the test set accuracy. It must also be ensured that the test set accuracy is not lower than the training set accuracy.

Some initial training runs showed that the model was overfitting to the training data, therefore a dropout layer was added after each convolutional + max pooling layer. The model’s loss optimizer, loss function, and batch size were the same as the DFN final model. The CNN model is presented in figure 5.11. Figures 5.12 & 5.13 show the performance of the CNN after training for 300 epochs.

Diagram

Description automatically generated

**Figure 5.11** The CNN architecture used in the classification task

Chart

Description automatically generated

**Figure 5.12** CNN accuracy after training for 300 epochs

Chart

Description automatically generated with medium confidence

**Figure 5.13** CNN total 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 over-fitting 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 tasks was to determine the best way in which to embed the chord voicings. Looking at table 5.6 it can be seen that the binary image embedding, and CNN performed best. It must be noted however that there was no way to ensure that both networks were equally optimised.

The results suggest that representing chord voicings as binary images embeds a meaning closer to the ground truth, therefore that approach to embedding will be used in the c-gan model in the following section.

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

**Table 5.6** A comparison of classification task results

## 5.4 Chord Voicing Generation

As described in section 2.5, c-gan’s can be conditioned to generate outputs on a given input. 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 have variance, a noise distribution is also passed into the model alongside the chord label. 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.

This subsection will present the development of a conditional generative adversarial network (C-GAN) that is capable of generating accurate “dominant”, “minor-seventh”, and “major-seventh” chord voicings. The C-GAN should also be able to produce a good number of unique voicings. For example, if the model generated 100 “dominant” chord voicings, then at least 90% of those should be accurate, and 20% of those should be unique.

First, the process of building and optimizing the c-gan model will be described. Then, the results and capability of the developed model will be presented and evaluated.

### 5.4.1 Developing the Conditional Generative Adversarial Network (C-GAN)

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). The model uses convolutional layers in both the generator and discriminator and specialises in image generation tasks. As the chord voicings are represented as binary images, the DCGAN is a highly suitable model. Brownlee (2019, p?) presents a working conditional DCGAN implementation built using TensorFlow; the source code presented was copied and used as a basis for this projects model.

Brownlee’s DCGAN model was built to conditionally generate images of 10 different items of clothing. The model was trained using the MNIST fashion dataset (reference), 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)

The MNIST generator took as input a vector of 100 values which represented a random noise distribution, as well as one of the clothing type integer labels. The generator then transformed and concatenated the noise distribution and label into a 3D matrix of size 7 x 7 x 256. This allows the generator to apply convolutions across the 128 stacked feature maps and select features that can be used to create a convincing 28x28 image. This process is typical of DCGAN generators and can be seen in figure 5.15.

Diagram

Description automatically generated

**Figure 5.15** MNIST Fashion DCGAN generator (Brownlee, 2019) (Image adapted from Radford et al. 2015: 4)

The discriminator functions in a similar way to the CNN classification model proposed in 5.2.2. However instead of trying to predict the label of its input, the model tries to identify whether or not the input image is real (from the training data), or fake (from the generator). The MNIST cDCGAN takes as input a 28 x 28 image with its associated integer label and has an output layer consisting of a single sigmoid activated neuron. If the output is >0.5, then the discriminator thinks there is more chance that the input image is real, if it is <0.5, then the discriminator thinks there is more chance that the input image in fake.

The aim of the discriminator is to output values close to 1 for real images, and values close to 0 for fake images. The generator improves by moving its parameters in a direction that brings both of the discriminator’s outputs close to 0.5, meaning that it can no longer discriminate between real and generated images. This min max game can be expressed as:

Log(D(*x|y*)) is the discriminator's loss when estimating the probability that real data instance *x* with respect to label *y* is real. It can be thought of as the scaler that is outputted by the discriminator when inputted real images. D(G(*z|y*)) is the discriminator's loss when estimating the probability that a fake instance with respect to its label *y* is real. It can be thought of as 1 minus the scaler that is outputted by the discriminator when inputted generated images.

### 5.4.2 Adapting the cDCGAN Model

The initial phase in adapting Brownlee’s MNIST cDCGAN model to the needs of this project was to change the model to fit the shape of the jazz-chords dataset. The first adaption was to change the input image shape of both the generator and discriminator from 28 x 28 to 7 x 12. Also, as the generator was built to generate images that are 28x28, the architecture of the generator needed to be adjusted. The generator was changed so that it concatenated the 100-vector noise distribution and integer label into a 7 x 1 x 129 3D array???, and then applied 2D convolutions to output a 7 x 12. A sigmoid activation was also applied to the neurons of the output layer to ensure that the image pixel values were between 0 and 1.

### 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 not producing convincing images or improving over time. Despite tweaking the architecture of the generator model, this did not improve. Further research pointed towards the issue most likely being an issue with the training data (reference).

Looking back at figure 5.7 on page 27, it was clear that each chord labels voicings had a distinct enough distribution. However, 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 31, and in the note distributions shown in figures 5.1 – 5.4 on pages 24 & 25, where C1-B1 and C5-B7 represent the 4 unused rows within the chord voicing matrices. It was theorised that this could be causing additional noise within the generator and subsequently causing the output binary images to not represent the training data distribution.

To combat this issue, 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 is visualised in figure 5.16.

A picture containing graphical user interface

Description automatically generated

**Figure 5.16** Transforming chord binary images from 7x12 matrices to 3x12 matrices

After changing the shape of the chord images, the generator was updated to concatenate the noise vector and integer label into a 1 x 3 x 129 3D array, which then applied the same convolutions whilst up sampling the 2D axis to 3x12. This can be seen in the presented generator architecture in figure x.

This change in the image shape greatly improved performance of the adapted cDCGAN model, with the discriminator loss in respect to generated images stabilising at around 0.2 after 200 epochs.

However, this highlighted another issue in that the adversarial losses gave little insight into the real performance of the generator. This is a well-known issue within the literature, where recent works in deep learning literature have shown the benefits of also using perceptual metrics in improve the performance of C-GANS (Lucas et al., 2019: 1).

In order to implement some perceptual metrics, a function was created that checked the accuracy and uniqueness of each chord labels conditionally generated chords voicings after each training epoch. The function generated 100 chord voicings for each of the 3 chord labels. The accuracy was a measure of how many unwanted notes (as presented in table 4.3, p.20) were present, and the uniqueness was implemented as 1 minus the number of chord voicings that were identical.

By tracking these 4alongside the adversarial losses, Goodfellow’s principle could be more easily followed in order to optimise the network architecture. Through a process of experimentation, the adapted generator model was simplified to have only one 2D convolutional layer. Experimentation with the discriminator did not improve performance, and thus its architecture remained unchanged. The hyper parameters of the models, as shown previously, also remained unchanged. The final adapted cDCGAN generator and discriminator architectures are presented below in figures 5.17 and 5.18. The results of the model are presented in the next section.

Diagram

Description automatically generated

**Figure 5.17** The adapted cDC-GAN Generator network architecture

Diagram

Description automatically generated

**Figure 5.18** The adapted cDC-GAN discriminator network architecture

### 5.4.4 Results

After adapting Brownlee’s cDC-GAN model to fit the jazz-chords binary image dataset and implementing system to track measure perceivable metrics, Goodfellow’s principle was followed in order to maximise the accuracy and uniqueness of the generators output chord voicings. Figure 5.19 shows the accuracy and uniqueness of the generators output after 250 training epochs. The performance of the trained model is presented in table 5.7. The adversarial losses of the adapted model after 250 training epochs are presented in figure 5.20.

Chart, histogram

Description automatically generated

**Figure 5.19** 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 cDC-GAN model

Graphical user interface

Description automatically generated with medium confidence

**Figure 5.20** Discriminator and Generator loss after 250 training epochs

As can be seen, the trained adapted cDC-GAN model is able to conditionally generate chord voicings of a given chord label with high accuracy and satisfactory uniqueness. After 250 epochs, the discriminators loss with respect to generated chords stabilised at around 0.2. This suggests that the generator performance is sub optimal, as it was only able to fool the discriminator 20% of the time. The discriminators loss with respect to real chords also stabilised at around 0.2, meaning it was able to correctly identify real chords in 80% of cases.

Some research from the cGAN literature suggests that adversarial losses are not always a meaningful indicator of the model’s performance, as there is no way to ensure that both the generator and discriminator models are equally performant (reference).

The adapted cDC-GAN model can now be used in the task of harmonic lead sheet arrangement, as it can take successfully take chord symbols and output chord voicings to represent them.Figure 5.21 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 scalar represents a probability of that note being in the chord. Figure 5.22 shows the output chord matrix as note numbers and as note labels. Note that the first value in the output matrix is 16, or C2.

A picture containing table

Description automatically generated

**Figure 5.21** 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 matrices

Shape

Description automatically generated

**Figure 5.22** The generated “dominant” chord 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 lead sheet arranger system performs 3 main operations. First, it extracts the chord symbols from the lead sheet as well as some meta information. It then passes those chord symbols into the cDC-GAN generator model trained using the jazz-chords dataset. Finally, it takes the generated chords, and uses the 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.

Diagram

Description automatically generated

**Figure 6.1** LSAS system diagram

## 6.1 Development

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 the 3 main operations outlined above. The following subsections will outline how each subsystem was developed.

### 6.1.1 data\_extractor

The data extractor subsystem takes as input a MusicXML lead sheet, performs some data parsing and extraction functions, and outputs the lead sheets 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 cDC-GAN 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. The 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](#_4.3.1_Chord_Extraction). 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”. These chord type labels are then embedded using the integer encoding set out in figure 5.3 on page 16. The integer encodings are then passed to the chord\_generator subsystem and given as input to the cDC-GAN generator.

The system 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 below 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 considerable 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.

Diagram, schematic

Description automatically generated

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

### 6.1.2 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:

[[16, 27, 32], [28, 32, 35, 39], [16, 27]]

The development of the chord\_generator system was straightforward. The system imports the trained cDC-GAN generator model and passes it the list of chord type labels, which in turn outputs generated chord voicings. As shown in figure 5.21 on page 29, 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 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.

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

As outlined in section 5, the trained cDC-GAN 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 subsystem, 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 bass voicing configuration, with only 1 note slightly differing. Therefore, generating major-seventh chords for major-sixth chord 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 6.3 by looking the *chord\_types\_dict*’s keys.

The similarity-mapping approach was implemented by creating a dictionary which had 11 key-value pairs. The keys were made up of 11 chord-type labels, and the values were any of the 3 chord-type labels that the trained cDC-GAN model supports. The dictionary can be seen in figure 6.3. Note that looking up one of the supported chord-type labels simply returns itself.

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 6.3** A Python dictionary showing how similar chord labels were assigned to 1 of the 3 cDC-GAN’s support 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 cDC-GAN 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-types that the generator could not support due to them being too dissimilar to the 3 supported chord types. Instead of giving these incompatible chord types no voicing, they were instead given the notes C2 (16) and C3 (28), which act as neutral bass notes. This adaptation allowed the LSAS to avoid any unwanted gaps in the arrangements. An example of an unsupported bass note voicing is shown below in figure 6.4 below. Looking at the figure, the D augmented-seventh chord voicing is replaced by the notes D2 and D3, which as mentioned, act as simple bass notes.

The implementation details of this substitution system were as follows. A function was created to validate each chord label before it was given to the cDC-GAN model. The function looked up each chord label in the *chord\_types\_dict*. 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 6.3 it can be seen that a lookup of “major-sixth“ would return a cDC-GAN 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 explained above.

Diagram

Description automatically generated with low confidence

**Figure 6.4** A 1-bar extract of a full arrangement produced by the LSAS and highlighting a work-around for unsupported chord types

### 6.1.3 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 all of the above listed inputs. 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 located as a child of the *attributes* element - shown in figure 6.2 on page 31. 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 below:

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 G is 5 notes (semi-tones) above C, meaning all of the notes in the chord voicing will be reduced by 5:

*Generated chord voicing (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 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 harder 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 was flat, and the other when the key signature was sharp.

The advantage of creating look up dictionaries is they allow for each conversion to be performed in constant time (O(n)). An example of this process 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 was 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’]

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. The *stem* element indicates which way the notes stem points and is dependent on the location of the note on the lines.

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 bar (measure) element. The *note* elements were inserted after the last 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.

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 6.5.

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 6.5** 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 directory.

## 6.2 Results and Evaluation

The lead sheet arrangement system was tested using the Wikifonia dataset, which is a collection of 7000 jazz lead sheets. Figures 6.6 and 6.7 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 whole system are presented.

Graphical user interface

Description automatically generated with medium confidence

**Figure 6.6** The first 8 bars 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 shown the LSAS’s arrangement.

Calendar

Description automatically generated

**Figure 6.7** The first 8 bars 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 shown the LSAS’s arrangement.

### 6.2.1 Capabilities

- The system can successfully perform harmonic arrangement on a given lead sheet

- The harmonies (chord voicings) are highly accurate (<93%)

- Due to the nature of the cDC-GAN generator, the number of different arrangements possible for a lead sheet is immeasurably large

- The system can arrange any unseen lead sheet as long as it is in standard and correct MusicXML format with the extension “.musicxml” or “.xml”.

- The system can arrange any genre of lead sheet, and is not limited to jazz lead sheets

- 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

### 6.2.2 Limitations

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

- The chord voicings generated for 7 of the 10 supported chord types are slightly inaccurate (see 6.1.2)

- 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

In order to evaluate the effectiveness of the system, 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 cDC-GAN has already been analysed and presented in section 5.3.2, table 5.7 page 40, however it will be briefly revisited here.

**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 intermediate and advanced jazz pianists will be asked to examine and listen to 5 arrangements created by the lead sheet arrangement system. They will be asked a series of qualitative questions relating to the musicality of the arrangements.

Both of these evaluations are presented below.

### 6.2.3 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

### 6.2.4 Technical Evaluation

As proposed above, the technical evaluation is conducted based on the three criteria listed above. Each of the 5 selected arrangements will be 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-advanced 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* (reference viii). 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 is dependent 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 cDC-GAN 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 6.8 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 only 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.

Diagram

Description automatically generated

**Figure 6.8** An 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 the dataset used to train the cDC-GAN model. 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. 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 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. There were however 15 voicings 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 (reference). 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 (reference).

The quality of the chord voicings is largely a result of the quality of the dataset on which the cDC-GAN model was trained. For example, if each voicing in the dataset was a *spread voicing*, then it can be assumed that the majority of the generated voicings would also be *spread voicings*. The issue of voice leading however is the result of the cDC-GAN model. The cDC-GAN generator can only predict individual chord voicings and has no concept of how two or more voicings fit together. As explored in section 2.3, 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 cDC-GAN model in order to produce a level of *voice leading* within the generated arrangements.

A post-generation algorithm could be implemented to solve the above issues. The algorithm 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.

### 6.2.5 Participant led Evaluation

A total of 20 intermediate and advanced jazz pianists were contacted in order to participate in this evaluation. 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 6.9.

|  |
| --- |
| ***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 6.9** A list of questions provided to each participant

(Awaiting results)

# 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 in deep learning in the area of 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, achieved, and then presented. 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 gathered a library of MusicXML fully arranged jazz piano songs which importantly all had chord labels. A scraper was then developed that could successfully extract chord voicing and chord label pairs. This scraper allowed a novel dataset, the jazz-chords dataset, to be presented. The dataset consists of 7510 jazz chord voicing-label pairs. The jazz-chords dataset presented the opportunity to train a conditional generative adversarial network 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 cDC-GAN, was adapted to fit the jazz-chords dataset, and successfully trained to be able to conditionally generate different kinds of jazz chord voicings. Finally, the overarching lead sheet arrangement system was developed, with the trained cDC-GAN model at its core. The lead sheet arrangement system is capable of taking any piano lead sheet in MusicXML format and outputting a full arrangement.

One of the most notable capabilities of the LSA system 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.

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 cDC-GAN model along with the jazz-chords dataset aim to extend the research of Chen et al. (2020) and their *Chord Jazzification* system. The type of model, the conditional GAN, provides an alternative to Chen et al.’s RNN model. The results of the cDC-GAN chord generator also provide future researchers with an opportunity to explore a combination of the two approaches.

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 MusicXML holds much more information. The lead sheet arrangement system 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 C-GAN implementation 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 music and deep learning. The chord scraper system can be used on any MusicXML files and can be used to extract infinitely large datasets for use as training data. 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 data science in Music, as it is able to extract and present the harmonic and meta-information from MusicXML files in a short time. It could also be improved to extract rhythmical information.

The lead sheet arrangement system itself 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. ……

Carrying out this research project has provided the author with some important insights into the field of deep learning in the 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 – musical datasets that have been extracted from primary data. This issue is widely acknowledged within the current literature (reference that review paper), and was further highlighted in this paper when conducting an extensive review of the publicly available data in section 2.6, page 8,

This project has shown the effectiveness of stacking piano octave vectors rows to form binary image encodings of chord voicings. Both Liu et al. (2018b) and Dong et al. (2018) trained GANs using 3D matrix encodings of chord voicings in MIDI format, however most of the literature presents chord voicings as 1D vectors (references). To our knowledge of the current literature, the embedding method presented in this research provides a novel approach to chord voicing encoding.

An issue that this project has highlighted is the challenge of how deep learning models can differentiate between different chord types. As presented in section 2.5.1 on page 27, “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 (reference). 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 (glossary) 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 (reference). 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 introduction, 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 attempted to perform both harmonic and rhythmic lead sheet generation. And secondly, a network that could generate chord voicings that were sequentially arranged 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 of high priority for future iterations of the LSA’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 LSA that could perform complete lead sheet arrangement would also be of significant interest to the jazz piano community, especially if the arrangements possessed a high degree of musicality.

Another restriction of this research project was a general lack of data. The cDC-GAN model of the LSA requires chord voicings that are labelled, and these chord labels are only available in the MusicXML format. In order to progress the field of deep learning in music, there needs to be a significant improvement in the amount and quality of available data (reference).

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.

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.

Isola, P., Zhu, J.Y., Zhou, T. and Efros, A.A., 2017. Image-to-image translation with conditional adversarial networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1125-1134).

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.

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.

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

Rebelo, A., Fujinaga, I., Paszkiewicz, F., Marcal, A.R., Guedes, C. and Cardoso, J.S., 2012. Optical music recognition: state-of-the-art and open issues. International Journal of Multimedia Information Retrieval, 1(3), pp.173-190.

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.

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

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 [⏎](#_2.2_Generative_Deep)

##### A picture containing graphical user interface Description automatically generated

A picture containing graphical user interface

Description automatically generated

Table

Description automatically generated with medium confidence

Table

Description automatically generated with medium confidence

Graphical user interface

Description automatically generated