THIRD YEAR PROJECT REPORT

3YP Title Here

Authors:
Kitty Fung, Edward Gunn,
Terence Tan, Di Wan

Supervisors:
David De Roure, Kevin Page

May 14, 2022

UNIVERSITY OF OXFORD

Abstract

Faculty Name

Department of Engineering Science

Third Year Project

3YP Title Here

by Kitty Fung, Edward Gunn, Terence Tan, Di Wan

The Project Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Contents

List of Figures

List of Tables

Chapter 1

Project Definition

1.1 Introduction

The process of writing music often begins with and idea for a melody which is followed by the development of chords to accompany it. The matching of chords to a melody is a skill which usually requires years of training in musical techniques such as harminisation. There is a large market of amateur musicians who lack the necessary training for this task but would otherwise enjoy the experience of developing music. In this report we will detail the design of **INSERT NAME HERE**, a system which facilitates the generation of an appropriately matched set of chords to a given monophonic melody. Users are able to record a melody using their microphone and regenerate the chord sequence until they feel the they have found one suitable for the intended feel of their song. Songs and generated chord accompaniments can be played back to the user, saved to a library of songs and shared using our songwriting community feature.

The problem of converting a recorded melody to a set of chords can be decomposed into a set of simpler sub-probblems, these being: The conversion of the recorded melody into a form which numerically represents its features. The generation of a set of chords from this numerical representation. The latter of these two problems can be solved in many ways many of which require a detailed knowlege of music to find patterns in melodies to match to chords. **WHY IS THIS A PROBELM FOR US** A method which requires little knowlege of music is the used of a machine learning model to extract these patters from data of known chord melody pairings from professionally composed music. The curation and processing of an appropriate dataset

requires significant effort and care in itself. This leads to a natural division of labour across the project into the catagories of: User Interface and general product design - detailed in **??** by Di Wan; Curation and preparation of a dataset in an appropriate format - detailed in **??** by Terence Tan; The design of an appropriate machine learning model - detailed in **??** by Edward Gunn; The conversion of recorded melody to the same format as expressed in the dataset - detailed in **??** by Kitty Fung.

1.1.1 Musical terminology

Throught this report we will use a variety of musical terminoligy which will be defined here. A piece of music is composed of a sequence of adjacent **measures**, periods of time in which notes and chords can be played. We will generally take a **measure** to mean a bar in the music, however it is not restricted to this. We restrict **chord** to refer to a triad of notes, the justification for this is explained in **reference to explanation**. The use of **melody** refers to a monophonic melody in which only one note is played at a time, excluding accompanying chords, unless otherwise stated.

1.1.2 Subsection 2

Morbi rutrum odio eget arcu adipiscing sodales. Aenean et purus a est pulvinar pellentesque. Cras in elit neque, quis varius elit. Phasellus fringilla, nibh eu tempus venenatis, dolor elit posuere quam, quis adipiscing urna leo nec orci. Sed nec nulla auctor odio aliquet consequat. Ut nec nulla in ante ullamcorper aliquam at sed dolor. Phasellus fermentum magna in augue gravida cursus. Cras sed pretium lorem. Pellentesque eget ornare odio. Proin accumsan, massa viverra cursus pharetra, ipsum nisi lobortis velit, a malesuada dolor lorem eu neque.

1.2 Project management

Sed ullamcorper quam eu nisl interdum at interdum enim egestas. Aliquam placerat justo sed lectus lobortis ut porta nisl porttitor. Vestibulum mi dolor, lacinia molestie gravida at, tempus vitae ligula. Donec eget quam sapien, in viverra eros. Donec pellentesque justo a massa fringilla non vestibulum metus vestibulum. Vestibulum

in orci quis felis tempor lacinia. Vivamus ornare ultrices facilisis. Ut hendrerit volutpat vulputate. Morbi condimentum venenatis augue, id porta ipsum vulputate in. Curabitur luctus tempus justo. Vestibulum risus lectus, adipiscing nec condimentum quis, condimentum nec nisl. Aliquam dictum sagittis velit sed iaculis. Morbi tristique augue sit amet nulla pulvinar id facilisis ligula mollis. Nam elit libero, tincidunt ut aliquam at, molestie in quam. Aenean rhoncus vehicula hendrerit.

1.3 Tech strat

1.4 Risk

1.5 Finance

1.6 Ethics

From the National Society of Professional Engineers Code of Ethics[1], it is of the utmost importance to ensure engineering work upholds honesty, impartiality, fairness, and equity, and must be dedicated to the protection of the public health, safety, and welfare. Since our design interacts with users through a mobile application, ethical concerns have arisen regarding the interactions between users as well as data handling on our end. In this section we will address issues related to data ethics.

Governmental organisations have published regulations[2] and frameworks[3] that are minimum standards for handling data ethically. The core values of data ethics is to treat data responsibly and righteously.
There are 3 overarching principles[4] to adhere to but in short, we as the developer should treat our users, data as we would for our own.

1.6.1 Transparency

We should be clear and publish specifications of our project in a way that users can understand and access easily. It is necessary to delineate a privacy policy to make

[1] **codeofethics**.
[2] **EUdataregulations2018**.
[3] **framework**.
[4] **framework**.

sure users read and agree to it before they use our application and keep the policy at a prominent location, i.e. in the footer for webpage, in the developer section of the app listing or on the sign-in interface of the app.
We would include the following items in our privacy policy:

1. Company licence and registration reference number: Registration at the Information Commissioner's Office is required before collecting data from the public

2. Details and purpose of data collected: We should limit data collection to what is necessary and to an explicit purpose. Thus, we will collect

   Personal details - Name and email address are the necessary fields. Gender, age and profile picture are optional inputs that will help improve user experience and accuracy of the model.

   Audio data - Audio signal and metadata to feed in our machine learning model to improve accuracy.

   Networks and connections - Collected for our interactive community features.

   Usage - Contents and functions that the users have viewed or engaged with, as well as the duration.

   Device information - Device attributes like signal strength, battery levels, version of operating system to adjust the operation of our app.

   Cookie data - Both first- and third-party cookies are needed. For the first-party cookies, to personalise and optimise user experience, we would save their usage preferences like language, dark or light background mode. There are 2 purposes for using third-party cookies. Firstly, since we will be displaying advertisements, these advertising companies will be able to place cookies on our app for standard users, but not the premium users who subscribed to our services. Secondly, we will be implementing social sign-on so users do not have to create a new account on our end. In this case, the social media platform will be placing their third-party cookies on our app.

It is important to note that users have the right to know the third parties that have access to their data.

3. Data retention policy: We will specify how long information will be kept and the procedure of disposing the data when an user deactivated his account or when the data is no longer necessary to collect. Moreover, we would have to erase or rectify inaccurate data without delay. Although there is no limitation on data storage, we will have to act ethically and decide the timeframe based on the genuine motivation of retaining the data. Data should only be collected when it is vital in the context of app operation, thus we should not keep data just in case it is needed in the future. For data that has expired (past the retention period), we would have to either delete it or anonymise it. If we are to delete the data, we have to ensure all digital and hard copies of the data are destroyed. This action requires careful documentation of data storage from the date we collect data from users as traces may often reside in forgotten databases. Anonymising data means that a piece of information cannot be associated with an identity, which will not help with improving user experience, but still can be used to monitor the entire application performance.

4. Access rights within the team: Different team members of our project will have access rights to different types of data collected from the users, and we should delineate who will be responsible for which part of the data we store.

5. Rights that users have over their data: Users should have the right to request a copy of data provided, request us to delete the data and object to our data processing.

6. Notification of changes to privacy policy: We will contact users to review and accept the revised policy through email.

7. Security standards: We should specify the encryption standards and the processes in place to test the confidentiality, integrity and availability of our system.

8. Contact information: Contact details like organisation contact number, email address, office and postal address.

1.6.2 Accountability

We should process personal data responsibly and systematically, as well as endeavour to reduce risks for individuals and mitigate social and ethical implications.[5] This can be done by creating a department that overviews the entire project and ensures data is managed ethically throughout the process. As we have an interactive community feature, we must ensure that our app does no harm in any way. A new Online Safety Bill is being drafted recently to fight online hate crime, and online companies are liable for failures to deal with inappropriate material postsed online.[6][7] Companies have the responsibilities to confine illegal and harmful discussions and a conventional and efficient method to guarantee this is to implement algorithmic censorship.

Moreover, to be publicly accountable, effective governance and oversight mechanisms that can be exercised by the public are necessary. We can enact this by hiring an independent third-party audit review our data processing.

1.6.3 Fairness

When processing data, we should ensure no societal, racial or health bias is involved. Our project collects minimal personal data so the problem of differential processing for distinct groups can be avoided. On the other hand, another perspective of fairness can be manifested in the form of **Responsible Research and Innovation (RRI)**:

RRI is a process that seeks to promote creativity and opportunities for science and innovation that are socially desirable and undertaken in the public interest.[8] Not only does research bring novelty and value to society, it may also bring forward ethical dilemmas, social transformations and adverse consequences to society. Thus it is key to put emphasis on innovating responsibly and creating changes that have positive societal and environmental impacts.

There are a few areas of RRI that we should keep in mind:

1. Anticipation: We should vision the consequences of our research and innovation conducted. It is to ensure that the consequences of undertaking the

[5] **principles**.
[6] **francis**.
[7] **parliamentlaw**.
[8] **ukri**.

research are considered and reflected in the research design. Although in this project we are not creating physical technology, we are building a platform that allows users to communicate with each other and if not dealt carefully, our app may become a breeding ground for online hate crime.

2. Reflection: researchers should always reflect on the research question they are investigating, the type of data collected, the method used to analyse the data and the implications of the findings. They should also judge if the research is required. An organisation can force reflection through organisational processes and structures like a project advisory board or quality assurance reviews. Reflection allows researchers to review the project from a macroscopic point of view.

3. Ethics: Researchers should uphold integrity and prevent misconduct. They should take accountability for both the undergoing research and behaviour.

4. Gender Equality: Since male engineers still dominate the engineering industry, it is easy to be biased in a project design, i.e. building models and interfaces that better suit male users. Thus it is paramount to have opinions from the underrepresented group as to create a comprehensive innovation.

5. Open Access: Making the research output publicly available to everyone so the whole society can be benefitted by reducing wasteful duplication, increasing transparency and reproducibility of results.

6. Governance: Organisations should establish practices that foster RRI, i.e.

    - Having transparent and reflective internal procedures
    - Promoting participatory governance
    - Fostering stakeholder engagement exercises
    - Encouraging future-oriented governance
    - Valuing responsiveness

7. Public Engagement: Researchers should settle upon a motivation and suitable audience before any public engagement. For our project, the goal would be to allow amateurs to enjoy composing music without music professionalism.

1.7 Sustainability

As technology advances, energy consumed by computers and digital personal devices is taking up a larger portion in worldwide energy consumption. According to an article published by **energypy** and a research done by GO-Globe[9], there are about 8,918,157,500 active mobile devices consuming a total of 3.46e12 kWh energy per year. While a lot of effort has been put into reducing the energy consumption from the hardware perspective, it is also prime to focus on the impact of software implementation as well. Other than writing energy-efficient codes (i.e. optimised searching and sorting algorithms), the programming language chosen can make a substantial difference too. Energy, time and memory size are classified as the major resources required for running a program. At first glance, since $Energy = Power * Time$, there seems to be a correlation between energy and time. Yet, **energyplanguage** suggests that since *Power* is not a constant, we cannot draw the conclusion that when *Time* increases, *Energy* must increase. Pereira ranked the performance of different programming languages according to its energy, time and memory size consumption (p.7). Surprisingly, there is no strong correlation between these 3 components. Instead, the performance depends on the category of the programming language. The programming languages can be divided into 3 execution types: compiler, interpreter and virtual machine.

Compiler converts the entire high-level language code to machine-readable language all at once. Some examples are C, C++, Rust and Go.

Interpreter translates one statement of the high-level language code to machine-readable language at a time. Common interpreted languages are PHP, Python and JavaScript. Thus, even if an interpreter takes less time to analyze the source code, it takes longer time to execute the process.

Virtual machine executes an intermediate code translated from the high-level language inputted. Some of the popular languages that uses virtual machines are Java, Scala, JRuby.

[9] **energy**.

Since our application has no requirements on the computational time and memory size, we can solely focus on the energy consumption between different languages. Even though the energy needed to run different algorithms varies for different languages, it is certain that compiled language requires less energy compared to virtual machine and interpreted languages. Our team has used Python for the design and implementation so it is rational to switch from Python to C/ C++/ Rust to code in the future.

Also, according to a research done by **cloudpercent**, we can reduce energy consumption by 87% by employing a cloud architecture. Yet, researches have shown that cloud computing is more energy efficient than relying on local devices (i.e. using our private data centers, computing on users, smartphones).

Firstly, taking Amazon Web Services (AWS), the leading service provider in the cloud infrastructure market as an example, it aims to fully power the operation with renewable energy by 2025[10]. Since it is complex and cost-ineffective for individuals or small-scale companies to switch to clean energy, using the cloud services provided by eco-conscious companies would greatly reduce the carbon footprints.

On the other hand, centralised cloud provider allocates resources more efficiently since there is a higher utilization rate. If we choose an on-premise computing method, we will suffer from low utilization rates since the purchased equipments may only be fully utilized when there are usage spikes. Yet, cloud providers have a huge customer base and it is paramount for their busienss to optimise server usage. Thus sharing cloud servers can increase average server utilization to 50 percent or higher[11].

[10] **aws**.
[11] **cloud**.

Chapter 2

Product Design

App design combines User interface (UI) and User Experience (UX), while UI concerns about how the app pages look like and feel, such as the fonts, colours and arrangements of icons, UX focuses on the functionality and usability. The best app design process comprises research, ideation, problem identification, design, feedback and problem evaluation. **(need explanation)** Currently, we only focus on the mobile app design or specifically IOS design based on Apple Platform and Android Design.

2.1 UX Design

UX Design is the process of deciding how someone will use an app and creating a viable product. It's during the UX process that mobile app design ideas are generated and validated, with an aim to make sure that all your choices are going to work so that our app works.

The quality of user experience is the crucial factor measuring the quality of the design and usually it's the key distinguishing a successful app design from a bad one. Customers are becoming more and more picky about which app to use and so quick to abandon the app which they don not enjoy, so it's essential to invest time and effort in creating a great user experience.

2.1.1 Minimum Viable Product (MVP)

A minimum viable product, or MVP, is a product with enough features to attract early-adopter customers and validate a product idea early in the development cycle. To decide which features belong the MVP design, we use MoSCoW method to represent all the features we want to include in our design, which M (Must Have) o S (Should Have) C (Could Have) o W (Won't Have). The table seperates the features for our MVP from the advance functions that we want to include in our design.
We also take Impact Effort matrix and AARRR framework evaluation method into consideration, where Impact Effort matrix is shown in figure **??**. AARRR stands for Acquisition, Activation, Retention, Referral, Revenue. **need to explain**

MUST HAVE FEATURES FOR MVP	SHOULD/ COULD HAVE FEATURES
Sign-up and sign-in with email	Sign-in with third party accounts
Upload Audio	Real-time generation
Chord output	Chord regeneration
Chord layout	Export to PDF
Share posts	Message channel
Recommendation Engine	Lock Chords when regenerating
Help and support	Pitch Tracking
File storage	File backup

TABLE 2.1: MoSCoW Table

2.1.2 User Flow and Funtionality

User flows are flowcharts that illustrate the movement or journey of a user through your app. User flow diagrams are indispensable in mastering user experience. They allow you to understand how users interact with your app or website, the steps they take to complete a task or achieve a goal on your website. This will help you create a superior user experience for the user and meet their needs more efficiently.
There are two user flow diagrams representing two of the most important journey, one is sign-in flow figure[—] and the other one is main function page flow figure[—]. We use rounded rectangle to represent the termination, diamond to represent the decision, rectangle to represent the process and arrow to represent the flow direction.

Main Functionality

- **Sign-up and Sign-in with Email**

  From the start, our user can sig-up with their email and then login in with our app accounts. The reason why we want our user to create an account is that we can profile our users more easily.

- **Sign-in with with third-party Accounts**

  The purpose is to reduce the barriers for our users to enter our app. OAuth (Open Authorisation) is an open standard for access delegation, commonly used as a way for internet users to grant websites or applications access to their information on other websites but without giving them the passwords.This mechanism is used by companies such as Amazon, Google, Facebook, Microsoft, and Twitter to permit the users to share information about their accounts with third-party applications or websites.

- **Ask for song tag preference**

  After our user sign-in, a question with selective answers will show on the main page, the answers will be stored and feed to our recommendation engine, **Link to recommendation engine section**.

- **Store and backup files on Cloud**

  The purpose is to enable better synchronisation between devices and accessibility to the files.

  We will include CloudKit in our IOS design to allow users to store their saved files in iCloud, (merit: grate integration, recordings can also be stored)

- **Recording input**

  16khz

- **Chord generation and regeneration**

  Unlike Chordify, which generates the same chords each time for the same audio source, we provides the option for users to regenerate the chords and we aim to provides sightly different chords when regeneration button is pressed.

- **Community Page**

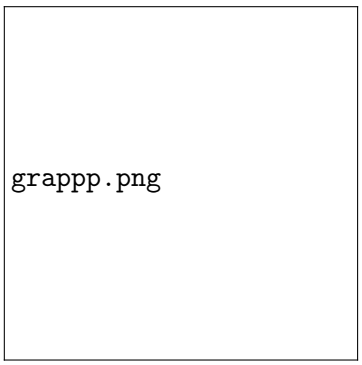
  Inspired by the Chinese music app, NetEase, one of our goals is to create a community page for our users to share their works and collaborate, we believe that the community element can will bring continuous traffic to our app, which can benefit the monetisation of our app in the future. As we mentioned before, the user experience is the key factor determing the successfullness of the app design and here the quality of the posts recommended to our users is a curcial element affecting our user experience. Therefore, it reuqires our recommendation engine to be well designed. **??**

- **Message Channel** The message channel allows our users to contact each directly and share the orginal chord files

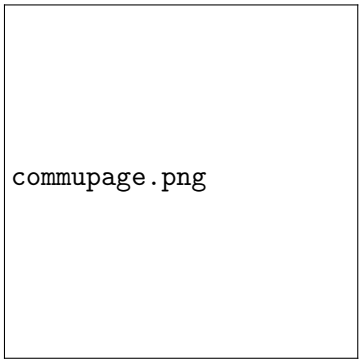
2.2 UI Design

After having the map of user flow, we started to design the prototype using Figma, which is a design tool

(A) Log-in page



(B) Main function page



(C) Community page

FIGURE 2.1: UI design for our app

2.3 Testing

Testing stage offers us an chance to discover the problems in our current designs and According to Compuware, 48% of users are less likely to use an app again if they are troubled with the app's performance. As reported by Compuware, only 16% of users can give the app a try for a second or third time.[1]

We divide testing stage into three stages,

[1] **lowtole**.

Chapter 3

Data

3.1 Dataset

The dataset used to train and test the machine learning model was obtained from another paper that was working on a similar project. The authors of that paper made their dataset available online. There are 2252 songs in this dataset, 1802 of which had been categorised as the training set while the rest had been categorised as the test set. Each song is in major key and only have a single chord per bar. For each song, all the relevant features had been extracted and placed into a single *CSV* file. These files can then be read and converted to DataFrame format using Python *Pandas* as shown in Figure **??**.

As can be seen in Figure **??**, the rows each contain information about a single note. Each bar is taken to be a single measure. The columns each represent a different piece of information about that particular note. *time* refers to the time signature, *measure* refers to the measure to which that particular note belongs to, *key_fifths* indicates the number of sharps/flats (e.g. -1 for one flat and 1 for one sharp), *chord_root* is the root of the chord with *chord_type* indicating the type of chord, *note_root* identifies the particular single note of that row, *note_octave* is the octave of that note, and *note_duration* indicated the duration of the note (4.0 for a quarter note).

3.2 Preprocessing of dataset

The dataset has to be preprocessed in order to make things simpler later on.

	time	measure	key_fifths	key_mode	chord_root	chord_type	note_root	note_octave	note_duration
0	4/4	1	0	major	C0	major-seventh	E0	4	12.0
1	4/4	1	0	major	C0	major-seventh	E0	4	2.0
2	4/4	1	0	major	C0	major-seventh	D#	4	2.0
3	4/4	2	0	major	F#	dominant	E0	4	12.0
4	4/4	2	0	major	F#	dominant	E0	4	2.0
...	...	...	...	...	...	...	...	...	...
88	4/4	32	0	major	Bb	dominant	rest	0	4.0
89	4/4	32	0	major	Bb	dominant	rest	0	8.0
90	4/4	33	0	major	C0	major	C0	5	16.0
91	4/4	34	0	major	C0	major	C0	5	12.0
92	4/4	34	0	major	C0	major	rest	0	4.0

93 rows × 9 columns

FIGURE 3.1: A song in DataFrame format after being read from CSV file

1. All songs are transposed to C major key. The key of a song determines the notes and the set of chords present in the song. Transposing all songs to a common key will basically normalise the different features of melodies and chords in different songs. The number of chord types present in the dataset will be reduced, which will decrease the number of chord types during the training process. Each song can be shifted to a different key without loss of the song's subjective character by shifting all the pitches equally.

2. The time signatures are all normalised. Different songs have different time signatures. To do so, each *note_duration* is multiplied by the reciprocal of the time signature *time* to give a normalised note duration.

3. Chord types are restricted to major and minor chords. All other chord types are converted to their most similar major or minor chords.

4. Some measures in the dataset contain rest notes. These measures are removed from the dataset.

5. Octave information is not required and is removed from the dataset.

6. There are also some irregular notes present in the dataset such as 'B-2' and 'A2'. The numbers after the letters do not seem to represent octave information and the paper from which this dataset was obtained made no mention of them. Given that they represent a very small portion of the dataset, measures containing these irregular notes are also removed.

Using *Pandas* to remove the unwanted measures mentioned above and to normalise the note durations is a straightforward task. However, shifting all the songs to C major key is trickier. We would need to know the original key of the song, and then transposed the *note_root* and *chord_root* appropriately to C major key. The original keys of the songs are stated implicitly by their *key_fifths*; since we know all the songs are in major key and that each major key has a unique number of sharps/flats, the numberical value of *key_fifths* can be mapped to a specific major key as shown in Table **??**. Note that there exist more values of *key_fifths* than shown, but preliminary analysis of the dataset shows that only integer values of *key_fifths* from -6 to 7 are present within it. We also create a mapping of the 12 notes to a numerical representation as shown in Table **??** to make the processing easier later.

Using Table **??** & **??**, we can list all the major keys present in the dataset and convert the pitches that exist within each major key into their numerical representations (which goes from 1 to 12 and then loops back to 1) as shown in Table **??**. As expected, the differences between the pitches of the same key are consistent across all the major keys (e.g. the difference between Pitch 1 and Pitch 3 is always 4 for every major key), which shows that we can indeed tranpose a song to a different key by just shifting all the pitches equally. For each DataFrame row, we just have to convert *key_fifths* to the corresponding major key using **??** to obtain the numerical representaion of Pitch 1 of that major key. We also convert *note_root* and *chord_root* to numbers using Table **??**. Next, the numerical representation of Pitch 1 is subtracted from those of *note_root* and *chord_root*, and the differences are added to Pitch 1 of the C major key (which is 1) to obtain the shifted pitches in notes and chords respectively. Note that the differences may be negative, which would lead to a shifted note/chord that is outside of the 1-12 range. This is easily rectified by using *if* statements to check if the shifted note/chord is non-positive and to add 12 (since a zero would loop back to 12) to it if

TABLE 3.1: Mapping of *key_fifths* to major key

key_fifths	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
Major key	Gb	Db	Ab	Eb	Bb	F	C	G	D	A	E	B	F#	C#

TABLE 3.2: Mapping of music notes to numerical representations

C/B#	C#/Db	D	D#/Eb	E/Fb	F/E#
1	2	3	4	5	6

F#/Gb	G	G#/Ab	A	A#/Bb	B/Cb
7	8	9	10	11	12

so.

The next step is to convert all the chord types to either major or minor chords using the mapping shown in Table **??**. This mapping has been checked by the supervisors of this project and deemed to be reasonable. Do also note that Table **??** does not contain an exhaustive list of all chords in existence, but only those that were found to be present within the dataset.

3.2.1 Code

As mentioned above, we first use *Pandas* to clean up the data. A nested loop can then used to loop through each DataFrame row for each song. The steps outlined above can then be applied to each row. At the end of the nested loop, the dataset is now fully preprocessed.

3.3 Model input format

Now that the dataset has been preprocessed, it can now be converted into a format that is appropriate for input to the machine learning model.

3.3.1 LSTM format

The LSTM data input format is a matrix with 37 columns, with each row containing informaion about a single measure. The first 12 columns represent the 12 note (C,C#, etc.), the next 24 columns represent the 12 major chords and the 12 minor chords, and the last column correspond to the absence of a chord. If a particular note exists within a particular measure, the element that corresponds to that particular measure

TABLE 3.3: The component notes/pitches of each major key

Major key	Pitch 1	Pitch 2	Pitch 3	Pitch 4	Pitch 5	Pitch 6	Pitch 7
C#	2	4	6	7	9	11	1
F#	7	9	11	12	2	4	6
B	12	2	4	5	7	9	11
E	5	7	9	10	12	2	4
A	10	12	2	3	5	7	9
D	3	5	7	8	10	12	2
G	8	10	12	1	3	5	7
C	1	3	5	6	8	10	12
F	6	8	10	11	1	3	5
Bb	11	1	3	4	6	8	10
Eb	4	6	8	9	11	1	3
Ab	9	11	1	2	4	6	8
Db	2	4	6	7	9	11	1
Gb	7	9	11	12	2	4	6

TABLE 3.4: Mapping of chords present within the dataset to major/minor chord.

Major	Minor
Dominant-ninth	Minor-seventh
Major-sixth	Minor-sixth
Major-seventh	Diminished
Dominant	Half-diminished
Suspended-fourth	Minor-ninth
Augmented-seventh	Diminished-seventh
Major-ninth	Minor-eleventh
Dominant-seventh	Minor-major
Augmented	Major-minor
Dominant-thirteenth	Minor-thirteenth
Power	Minor seven flat five
Suspended-second	
Dominant-eleventh	
Pedal	
Major 6/9	
Augmented-ninth	
Sixth	

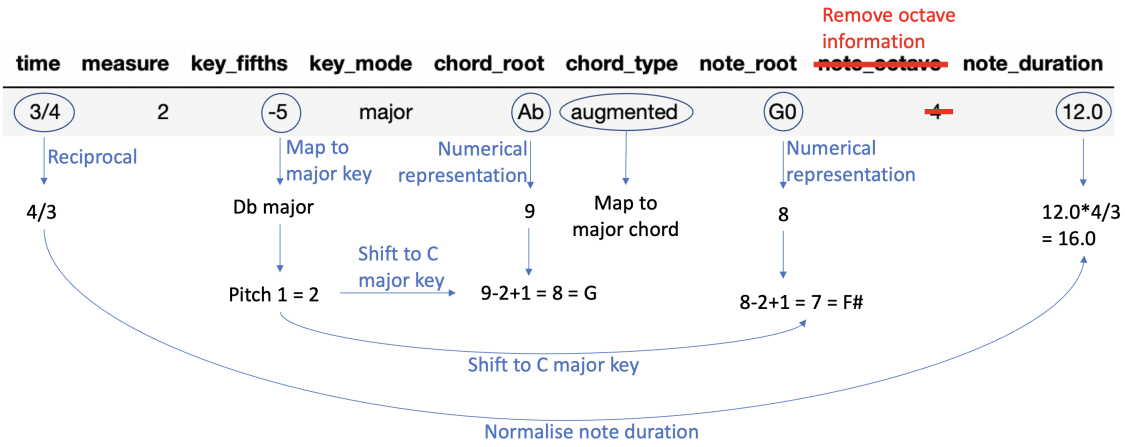


FIGURE 3.2: Pictorial representation of the preprocessing of the dataset.

(row) and particular note (column) will be the normalised duration of that note. Similarly, the element that corresponds to the chord of a particular measure will be a '1'. Notes and chords that are not present within that measure will be marked with a '0'. If no chords are present within a measure, the last column will be marked as a '1'. A pictorial representation of the transformation is shown in Figure **??**.

Code

We again use a nested loop to loop through each DataFrame row for each song. Two 1 by 37 arrays *LSTM_data* and *new_row* are initialised. As we loop through each row, we keep track of the meaasure. As long as the measure does not change, *new_row* is progressively updated with the normalised note durations of the present notes in this manner: $new\_row[0, note - 1] = new\_row[0, note - 1] + normalised\_note\_duration$, where *note* is the numerical representation of the note present in each DataFrame row and *normalised_note_duration* is the normalised note duration of that note. This works because the mapping between the notes and their numerial representation starts at 1 and ends at 12 (as can be seem in Table **??**) and indexing starts at zero in Python. Hence, the (*note*-1)th column of the LSTM format (and thereby *new_row*) will correspond to *note*.

Once the measure changes, the now complete *new_row* for the previous measure is concatenated with *LSTM_row* along the row axis, i.e. *new_row* is added to the bottom of *LSTM_row*. The elements of *new_row* are reset to zero before *new_row* is

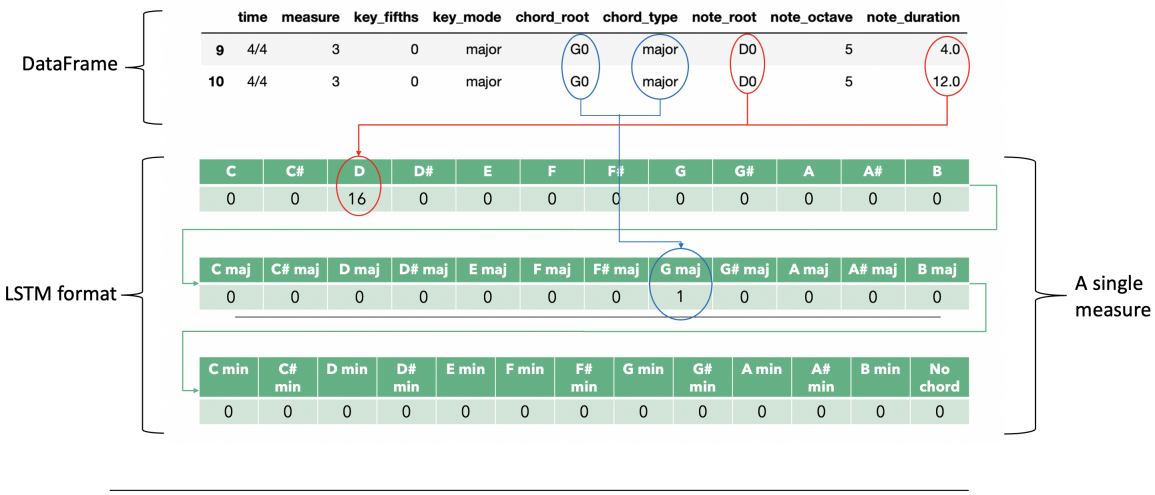


FIGURE 3.3: Transforming DataFrame to LSTM data input format.

updated with the note and chord information of the current measure. The note information can be updated as explained earlier. For the chord information, we can use *if* statements to set $new\_row[0, chord + 11] = 1$ if *chordtype* = 'major', or to set $new\_row[0, chord + 23] = 1$ if *chordtype* = 'minor', or to set $new\_row[0, -1] = 1$ otherwise (no chord present), where *chord* is the numerical representation of *chord_root* (mapped using Table **??**). Since we know that there is only one chord type per measure, we only have to update the chord information for *new_row* once, at the start of a new measure. The start of the first measure can be taken to be a special case of a measure change (in this case, transition from a measure initialised as 'unknown' to the first measure of the DataFrame).

3.3.2 Transformer format

The transformer requires two data inputs: a single sequence of notes for each song, and a separate sequence of chords for eaach song. Both sequences are constructed by going down the Dataframe rows, and adding the notes/chords of each row to the respective sequences based on the *note_duration* value, e.g. a sequence of 4 'C#'s for a 'C#' with a *note_duration* of 4.0, and a sequence of 16 'Bmaj' for a 'B major' of *note_duration* of 16.0. Another example is presented in Figure **??**. Of course, this means that only integer values of *note_duration* are accepted and measures with non-integer values have to be removed beforehand.

It is obvious that the sequences will be very long given that just the three rows in

DataFrame:

	time	measure	key_fifths	key_mode	chord_root	chord_type	note_root	note_octave	note_duration
0	4/4	1	0	major	A0	minor	C0	5	4.0
1	4/4	1	0	major	A0	minor	A0	4	8.0
3	4/4	2	0	major	C0	major	C0	5	4.0

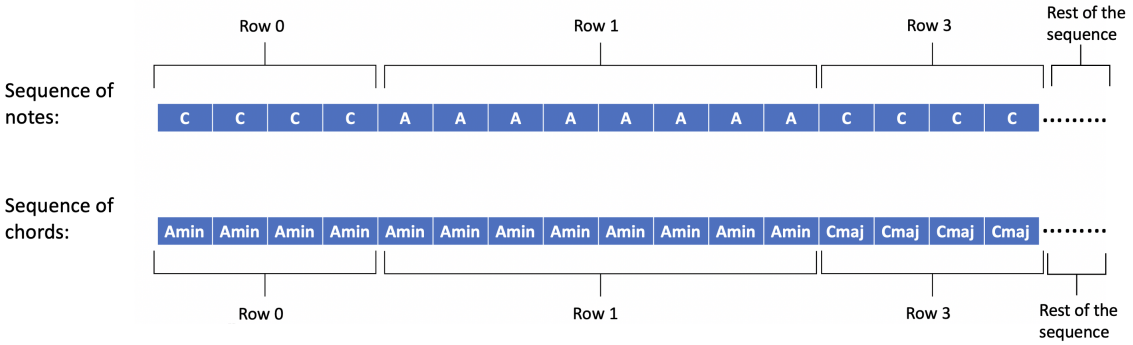


FIGURE 3.4: Converting DataFrame to Transformer data input format.

Figure **??** resulted in 16 elements for each sequence. As shown in Chapter 4, the time complexity of the Transformer is n^2. Hence, it is crucial to reduce the sequence length to shorten the training time. This can be achieved by dividing all *note_duration* by their largest common factor for all measures within a single song. This may not reduce the length of the sequences for all songs (songs with *note_duration of 1.0* will have a trivial largest common factor of 1.0), but it will still reduce the sequence length for some songs, as shown in Figure **??**, which will decrease the training time significantly.

Code

The same nested loop as before is used again for this. Four empty lists *note_duration_li*, *note_li*, *chord_li*, *chordtype_li* are initialised. As we loop through the DataFrame rows of a single song, we append the information from each row to the respective lists. At the end of this inner loop, the largest common factor of *note_duration_li* can be found using the *math.gcd* function. Since *math.gcd* only accepts two arguments, we initialise *lcf* as the index 0 element of *note_duration_li*, and loop from the index 1 element to the last element of *note_duration_li*. Within this loop, *math.gcd* takes in *lcf* and *note_duration_li*[*j*] as its two arguments, where *j* is the currect loop index. In essence, we are just finding the largest common factor of the first two elements of

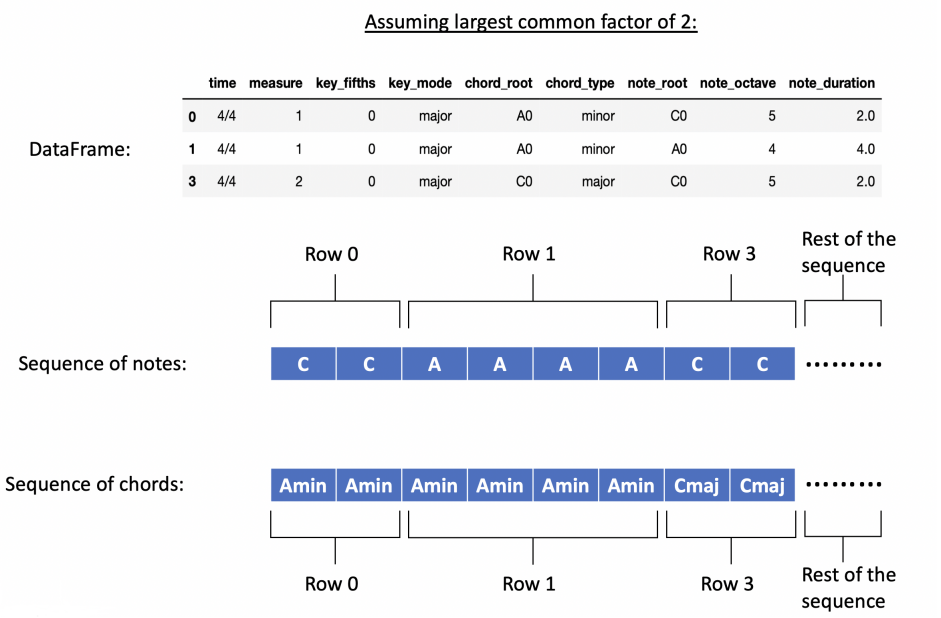


FIGURE 3.5: Converting DataFrame (normalised by the largest common factor) to Transformer data input format.

note_duration_li, and finding the largest common factor of the previous largest common factor and the third element, and so on. This will give us the largest common factor of all the values stored in *note_duration_li*, which are all then divided by this largest common factor to give *normalised_note_duration*.

We can now start to construct the sequences of notes and chords. Two empty arrays *row_note* and *row_chord* are initialised, and we loop through *normalised_note_duration* with loop index k. For the k^{th} element of *normalised_note_duration*, a 1 by *normalised_note_duration*[k] array filled with '1's is initialised and multiplied by *note_li*[k]. The result is concatenated with *row_note* along the column axis. For the chord sequence, a 1 by *normalised_note_duration*[k] array filled with '1's is also initialised. This array is multiplied by *chord_li*[k] if chordtype_li[k] is 'major', by $(chord\_li[k] + 12)$ if it is 'minor', and zero if there are no chords. The resulting array is then concatenated with *row_chord* along the column axis.

After looping through all the DataFrame rows of a song, *row_note* and *row_chord* are the now complete sequences of notes and chords respectively for that song. These will be the input to the Transformer model.

Chapter 4

Model

4.1 Introduction

The problem of generation of a set of chords from a melody is very similar to that of translating one language to another. The translation problem is one that is very popular in machine learning research and thus there is many resources on it. However the music related problem is harder to solve due to the extra dimension each of its elements contains. Each element in a melody has both pitch, represented by a descrete symbol or note, and duration whereas each element in the sequence of language is composed of only the discrete symbols or words. Therefore in order to use techniques developed for natural language processing it is necessary to encode the melody and chords in such a way that their dimensions are collapsed into one. Since this collapsing of dimensions has already been conducted in **REF TO DATA CHAPTER** we are able to take full advanatges of NLP techniques. There has been many attempts to apply some of these techniques to the chord generation problem. In order to find the best model we evaluate these attempts against a defined criteria and develop a novel model to evalutate against the same criteria.

4.2 Model Requirements

4.2.1 MVP Requrements

The MVP requirements were defined at the beginning of the project to ensure it is compatible with the other parts of the product that were simaltaniously in development and to ensure the possibbility of the creation of the prototye for proof of

concept. We required the model to be a black box in which we could input a melody and it would output chords which sounded good. The notion of sounding good is intentionally vague as what sounds good or bad is usually down to individual taste in music. There is never a definitive answer to which chords would sound the best. However, we felt **NEED STRONGER PROOF** that even people without any musical training would be able to judge whether chords fit the melody relatively accurately. Within this overarching requirement we defined tighter constraints for the sake of both practicality and user experience. The model would have to be designed for sequential data such that the temporal relationship of the different notes being played were taken into account. It would be possible to simply learn a function where for a given measure a chord suited to the notes played within that measure was generated without taking into account surrounding measures. This approach could produce reasonable results and be significantly more simple that other options, however, the quality and variation of chords generated would be lower than that of a model for sequential data, hence our choice to forego it. The model would also have to be conditional. For a given input it should produce a catered output. This is an obious contraint however for models such as a GAN it recquires changes to the format of the input data. To maximise user experience the models should be non-deterministic. This allows users to regenerate the accompaniment multiple times to obtain new chords and thus means they can choose what they feel is best. For practicality sake we constrained our MVP to only require one chord to be played each measure as the problem of determining when a chord should be played is of similar difficulty to choosing which chord to play.

4.2.2 Other possible Requrements

There were some constraints which were not used in our project which would provide better quality chords but provided too large a practical barrier to be implemented. The act of collapsing the pitch and duration of the notes into a single dimension removes information from the melody. In our case this information is the order in which the notes are played and the rhythmic intention of the composer. It is likely a model would be able to produce more suitable chords given this information. Thus, for a stricter set of constrains we could include the requirement that the

order and duration of notes are both maintained in the data. The accompaniment for music is not limited to a single chord played at the start of each bar. A much more interesting accompaniment would be generated if it were not limited to this restrictive format. As shown in **ReinforcementLearning** it is possible to create a model which learns the divisons within the music and thus learns when would be most appropriate to play a chord. Therfore, in order to allow for more interesting accompaniment, the requirement of one chord per bar could be changed such that chords should be played with closer freedom to that of a human composer.

4.2.3 Evaluation criteria

Approach Many attempts to solve the chord generation problem have been made each of which containing variants on models and implementations which result in large variations in mertrics used for evaluation. There is also a large variation in features for which quantative evaluation is impossible. As it would be impractical to implement each relevent model ourselves to allow for full standardistation of evaluation we will evaluate models built ourselves and from previous work based on a set of criteria defined below.

Data One of the biggest influences on the effectiveness of a model is the dataset on which it is trained. In general a larger dataset results in a better generalisation of learning **Need proof**. Most datasets are comprised of lead sheets which can be translated into chord melody pairs. Many datasets are then further divided down into measures in which there is one chord played per measure and the melody within that measure is somehow encoded. Thus for best comparison the number of measures used in the dataset is a good criterion for evaluation. This is a particularly important area for evaluation as it will strongly affect the output of the model and thus the results of other

Quantiative Evalutaion There is difficulty in quantative evaluation for this problem as there is no strictly correct chords for a given melody and thus comparison to any specific set of chords gives a skewed interpretation of the output. However, the use of conventional quantative evaluation does still correlate strongly with the

sentiment given by a human judging the quality of chords and thus can be cautionsly used in evalution. The test set from the data can be used to compare the outputs from the model to previously assigned chords. Accuracy can be found by finding the number of chords successfully predicted and dividing by the total number of chords produced.

$$\text{Accuracy} = \frac{\text{Number of Chords Correctly Assigned}}{\text{Total Number of Chords Generated}} \tag{4.1}$$

As this was the only quantative measure consistently used across previous implementations this is the only one we will consider for evaluation.

Qualitative Evaluation Some previous work **MySong, BLSTM** carried out experiments to judge the sentiment of untrained musicicans to the generated chords. Participants were played the melody accompanied by a varying set of chords and asked to judge which chords they felt were best out of a set containing chords generated by models and some human written accompaniment. The results from these experiements can be used to compare models to each other as well as evlauate them relative to the standard set by human written chords. As well as evaluation based on the quality of chords produced we will discuss extra functionality made possible by some models and the effect this has on other evaluation metrics.

Criteria The final criteria used for evaluation are thus:

- Number of measures in the dataset
- Accuracy in testing
- Human sentiment towards generated chords
- Extra functionality the model provides

4.3 Models

The generation of chords to accompany a monphonic melody is an area which has developed as machine learning techniques are improved. An early approach to

this problem from **ChordPrediction** used a standard MLP to learn the relationship between melodies and accompanying chords. MySong, the first attempt focusing specifically on a vocal melody in **MySong** used an augment hidden markov model with parameters such that users could adjust the "Happy factor" and "Jazz factor" to alter the mood of the generated chords to their preference. A weakness of the hidden markov model is that at each timestep only the previous timestep is considers when suggesting a chord. **BLSTM** utilises a BLSTM to increase the effect of long term dependencies on the output rather than relying only on the previous output. **MLForChords** tested and evaluated a number of more simple models such a Logistic Regression, Naive Bayes, SVM and Random Forest trained on data from 43 lead sheets

ReinforcementLearning proposed a model that learns a structured representation for use in symbolic melody harmonization. It utilises two layers of LSTMs to detemine when each chord should be played then utilise a policy gradient RL method to select each chord. MuseGAN

ChordGAN

CLSTMGAN for melody Generation

4.3.1 Model evaluation template

Model explanation/history - why is it generally applicable

Previous implementaitions

General backgorund

Evaluation

Data

Quantative

Qualitative

Extra features

4.3.2 HMMs

Explanation and Applicability

Hidden Markov Models or HMMs were first put forward by Leonard E. Baum in a series of statistical papers **list papers from wikipedia**. They model a stochasitc process in which the desired states, $\boldsymbol{X}$, are not directly observable but related states, $\boldsymbol{Y}$, which directly influence the desired states are. By modeling the $\boldsymbol{X}$ and $\boldsymbol{Y}$ Markov Processes **references?** we are able to infer the hidden state. To apply an HMM to our problem we take the hidden state, $\boldsymbol{X}$, to be the chords played, and the observable state $\boldsymbol{Y}$ to be the Melody. It is notable that the conditional probability distribution of the hidden variable $x(t)$ at time t, only depends on that of the previous time step $x(t-1)$ and that $y(t)$ only depends on the hidden distribution at the current time step $x(t)$. For our puroses this limits the "memory" the model could have to a single measure and thus seriously reduces the effect of relationship between chords across time.

Implementations

MySong The first application of an HMM to this problem while also being the the first attempt focusing specifically on a vocal melody from **MySong** used an augmented hidden markov model with parameters such that users could adjust the "Happy factor" and "Jazz factor" to alter the mood of the generated chords to their preference. No measure was given for the accuracy of the model, however a study which compared the quality of MySong to Manually assigned chords was carried out. In 264 comparisons the participants prefered MySon 95 times, manual 121 times and had no preference 48 times The additional user input possible with the "Happy factor" and "Jazz factor" parameters are reported to significantly improve the user experience, however, this is thier only implementation and so nothing can be infered about the quality of the HMM itself.

Chord Generation from Symbolic Melody As a comparison for the main model in **BLSTM** an HMM was implemented. The HMM is trained on a reduced version

of the **Include link?** wikifonia.org dataset which includes and array of Western music genres with 2,252 lead sheets, 1802 of which are used for training. This overall comes to 72,418 measures for the training set and 17,768 for the test set. They tested the accuracy of the HMM on sequences of length 4, 8, 12, and 16 bars long gaining an accuracy of 0.4033, 0.4043, 0.4041, and 0.4045 respectively and an average accuracy of 0.4041. They also carried out a user subjective test with 25 participants each evaluating 18 sets, each set containing a melody acompanied by three generated chord sequences and an original chord sequence. The users would listen to the music played and judge the accompaniment on a scale of 1, being not appropriate, and 5, being very appropriate. The HMM achieved an average score of 2.31 and the original chords achieved an average score of 4.04.

Machine Learning in Automatic Music Chords Generation An HMM was tested in **MLForChords** along with other simpler models. It was trained on 43 lead sheets, with 813 measures total. An accuracy of 0.4844 was achieved, however an choice of only 7 chords were used unlike the usual 24.

4.3.3 DNN-HMMs

Explanation and Applicability

A deep neural network HMM makes it possible to assume a posterior for the HMM using the output from the softmax output layer of the DNN. This allows for the posterior to be learned in training.

Implementations

Chord Generation from Symbolic Melody This was implemented much like the HMM from **BLSTM** mentioned above in **??**. The same dataset was used and an accuracy of 0.4502, 0.4482, 0.4495, and 0.4468 was obtained on the 4, 8, 12 and 16 bars respectively. This results in an average accuracy of 0.4487. On the user subjective test the DNN-HMM achieved a score of 2.48.

4.3.4 GANs

Explanation and Applicability

Generative Adversarial Netwoks or GANs were first proposed in the landmark paper **GANs**. Unlike most models GANs consist of two separate agents working against each other. The first model, the Generator, takes an input of random noise and outputs data which mimics the training data. The second model, the Discriminator, takes as input an example from the training data or an output from the Generator and outputs a value between 0 and 1 indicating whether it thinks the input is real or generated. The Binary Cross Entropy loss is used for the Discriminator avaraged across real and generate examples. The loss for the Generator is the log complement of that for the Discriminator. Thus the two models play the following minmax game:

$$\min_{D}\max_{G} V(D,G) = \mathbf{E}_{x\sim p_{data}(\mathbf{x})}[logD(\mathbf{x})] + \mathbf{E}_{z\sim p_{data}(\mathbf{z})}[log(1-D(G(\mathbf{z})))] \tag{4.2}$$

Theoretically the Generator and Discriminator can be any differentaible function thus leaving much room for flexibility. The problem with GANs for our uses is that they are not conditional, the only input to the Generator is noise. A proposed remody for this was presented in **CGANs**. By concatinating the label data, in our case the melody, with the noise as input to the Generator and doing the same with the training data and the label data, in our case a chord and melody, as input to the Discriminator GANs can be made conditional. Thus they would be suitable for our uses.

Implementations

ChordGAN **ChordGAN** uses a conditional GAN along with Chroma Feature Extraction to generate chords for a specific genre of music based on the dataset it was trained on. They used three different data sets one for each Pop, Jazz and Classical music each shorter than 1600 measures (specific lengths are not given). The model achieved an acuracy of 0.68, 0.74 and 0.64 respectively across the datasets.

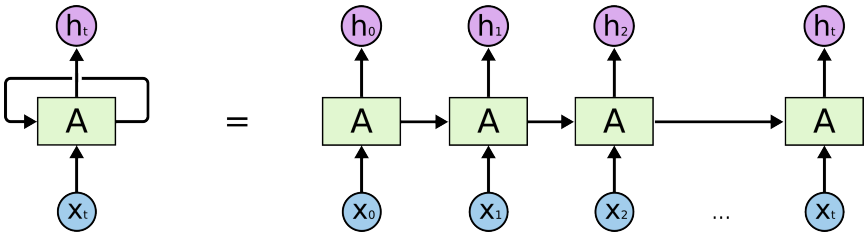


FIGURE 4.1: A many inputs to many outputs diagram of an RNN from **oinkina**

4.3.5 RNNs

Recurrent Neural Networks or RNNs have become a staple in the machine learning engineer's library of models. They are structured much like a normal MLP, however, they contain an extra connection to the state of the network in the timestep before. This means that the state of the network at each timestep depends that of the previous timestep and thus the state at the current timestep is affected by the state in all previous timesteps. This makes RNNs ideal for processing sequential data as temporal relationships are taken into account. The gradient of RNNs can be found using a variation of the backpropagation algorithm usually know as backpropagation through time. RNNs that operate on large sequences often experience the probblems of exploding or vanishing gradients leading to a saturation of learning.

4.3.6 LSTMs

The Long Short Term Memory model or LSTM was proposed in **LSTMs** in order to overcome the gradient problems related to RNNs and thus allow for faster training on long sequences. They are also capable or learning longer dependencies due to their internal memory. An LSTM cell has three gates: input, forget, and output. The state of these gates determines whether the cell allows new input, forgets old information, and affects the output at the current timestep. At timeset t, the states of the gates are given by:

$$i_t = \sigma(w_i[h_{t-1}, x_t] + b_i) \tag{4.3}$$

$$f_t = \sigma(w_f[h_{t-1}, x_t] + b_f) \tag{4.4}$$

$$o_t = \sigma(w_o[h_{t-1}, x_t] + b_o) \tag{4.5}$$

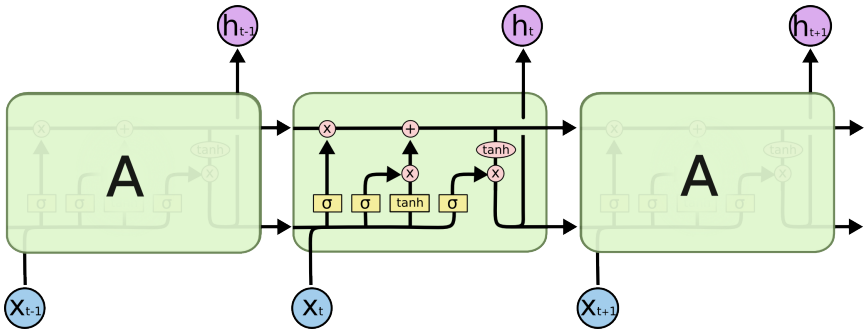


FIGURE 4.2: The repeating module in an LSTM from **oinkina**

where i_t, f_t and o_t denote the input, forget, and output gates state respectively, h_{t-1} is the output at the previous timestep. w and b represent weights and biases of each gate, x_t is the input to the LSTM cell, and $\sigma(.)$ is the sigmoid function applied elementwise. The current output of the cell is computed by:

$$h_t = o_t \circ tanh(c_t) \tag{4.6}$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \widetilde{c}_t \tag{4.7}$$

$$\widetilde{c}_t = tanh(w_c[h_t, x_t] + b_c) \tag{4.8}$$

Implementations

Chord Generation from Symbolic Melody This was the main model under scrutiny in **BLSTM** mentioned above in **??** and **??**. They build a bidirection LSTM with two hidden layers and used a hyperbolic tangent activation function. The softmax function is applied to the output in order to represent the probaility that each of the 24 chords is played. The same dataset as the HMM and DNN-HMM was used and an accuracy of 0.5055, 0.5032, 0.4923, and 0.4990 was obtained on the 4, 8, 12 and 16 bars respectively. This results in an average accuracy of 0.5000. On the user subjective test the DNN-HMM achieved a score of 3.55.

Automatic Melody Harmonization A particularly interesting model heaviliy relying on LSTMs surrounded by a reinforcement learning framework was proposed

in **ReinforcementLearning**. The simultaniously trained a Structured Representation Module responsible for learning note-level, phrase-level and segment level representaitions, a Segmentation Module acting as a reinforcement learning agent to decide whether the current note is the boundary of a phrase or segment and a Harmonisation Module responsible for generating chords for each segment. They used the Hooktheory Lead Sheet Dataset with 10,000 songs, no number of measures is given but with the differnece in model architecture so drastic a comparison on this metric would be inappropriate anyway. The achieved an accuracy of 0.3742 and compared that to an SVM, CNN, LSTM and BLSTM with blocked Gibbs sampling which achieved an accuracy of 0.2516, 0.2664, 0.2802, 0.2933 respectively.

4.3.7 Transformers

Transformers have been a revolutionary development in the field of NLP, first proposed by **Transformers** they have become very widely used and regularly produce state of the art performance. Transformers utilise the encoder decoder model for sequence to sequence translation. They also heavily use attention which is a mechanism for weighting the importance of different elements of a sequence of data. Specifically they propose the Scaled dot-product attention shown in **??**

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{4.9}$$

where Q, K and V represent the queries, keys and values respectively and d_k is the dimension or queries and keys.

4.4 Our Model

We propose the use of a conditional BLSTM GAN to learn the association between melodies and chords. The Discriminator consists of a number of LSTM layers followed by a linear output layer. The Generator consists of a linear layer followed by a number of LSTM layers and then by another linear layer. The Binary Cross Entropy loss is used to determine the Discriminator loss for each example. This is averaged across every measure and then between real and generated examples

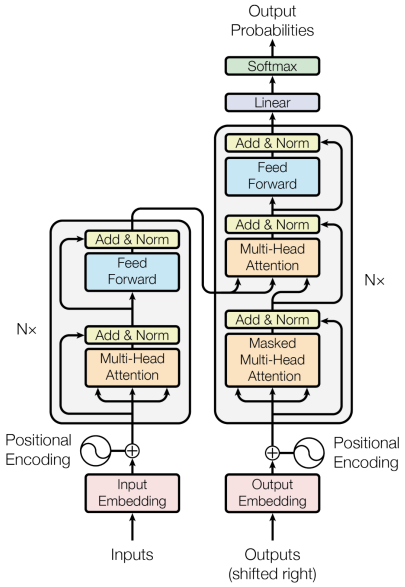


FIGURE 4.3: The architecture of a transformer from **Transformers**

4.4.1 Model Functionality

As proof of concept for the design we developed a command line based interface for the model such that it was easy to train, vary parameters and test. In production the user would be able utilise a subset of this interface, such as the option to generate accompaniment and have it played back, through a graphical user interface. The options available in the interface are show in table **??**

4.5 Training

Optimiser We used the Adam optimiser **Adam** which is recommended in the Stanford CS231n: Convolutional Neural Networks for Visual Recognition[1] as the default optimiser. **EXPLAIN ADAM WHEN NOT FALLING ASLEEP**

Loss As we are using a GAN we use the Binary Cross Entropy or BCE as our loss function

$$\frac{1}{N}\sum_{i=1}^{N} log(p(y_i)) \tag{4.10}$$

[1] https://cs231n.github.io/

TABLE 4.1: The parameters of the command line iterface for the model

Parameter	Options	Default	Description
-input_size	[input_size]	12	The size of the input vector to the discrimiator representing the melody
-output_size	[output_size]	25	The size of the output vector representing the chord played in each measure
-h_size	[h_size]	128	The size of the hidden layers in the LSTM layers
-n_layers	[n_layers]	2	The number of LSTM layers in the generator and discriminator
-noise_size	[noise_size]	12	The size of the noise vector concatinated with the input vector to the generator
-max_seqlen	[max_seqlen]	200	The maximum length of song in measures used in training
-src_data	[src_data]		The default path to the training data
-batch_size	[batch_size]	10	The size of the batches used in the stochasitc gradient descent algorithm
-epochs	[epochs]	100	The number of epochs used in training
-printevery	[printevery]	10	The number of epochs between printing an example during training
-load		False	Whether to load a model in or not
-load_dir	[load_dir]		The path to the folder containing pretrained models
-model_num	[model_num]	1	The number of the model to be loaded in
-save		False	Whether to save the model after training
-save_dir	[save_dir]		The path to the save directory
-playback		False	Whether to play an example of generated music at the end of training

which for the disciminator takes the form

$$\frac{1}{N}\sum_{i=1}^{N} log(D(conc(x_i, z_i))) + log(1 - D(conc(G(z_i), z_i))) \tag{4.11}$$

where x_i is are the real examples, z_i are the melodies and the *conc* function concatinates the two vector arguments. The loss is the sum of the loss of the discriminator on real data and generated data. The generator loss is

$$-\frac{1}{N}\sum_{i=1}^{N} log(1 - D(conc(G(z_i), z_i))) \tag{4.12}$$

which results in tring to maximise the loss of the discriminator on generated data. This is equivalent to finding the discriminator loss on generated data when labeled as real which is the method we use in our implementation.

4.5.1 Avoiding Overfitting

Dropout GAN techniques Reducing number of LSTM layers

4.5.2 Issues

Training is conducted in batches of a given size. It is unlikely that the size of the dataset is divisble by the batch size so the final batch is likely to be shorter than the others. This initially casued problems, however, we were able to set the dimensions of relevant objects, such as the input noise to the generator, to be relative to the current batch size rather than the usual batch size thus solving the problem. Softmaxing of outputs of generator Device managment

4.5.3 Testing

4.6 Results

Chapter 5

Audio Pre-processing

Before we feed the audio clip to our machine learning model, it is crucial to pre-process the signal to achieve higher accuracy and avoid further deterioration. The choice and implementation of the noise filter will then be explained in **??**. We then feed the filtered output to a pitch detection algorithm (PDA) in **??** and then a key detection algorithm (KDA) in **??**. Figure **??** shows the flowchart of the audio processing part of our project.

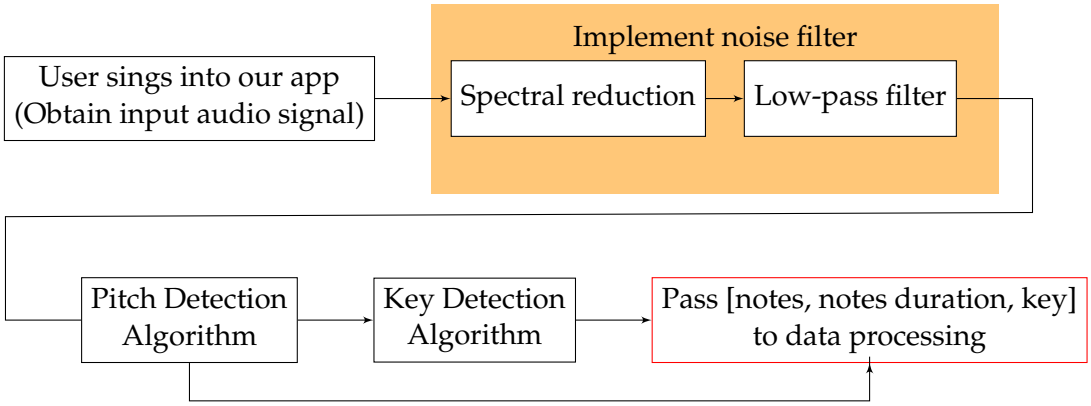


FIGURE 5.1: Flowchart of audio processing

5.1 Assumptions

Before we delineate the approach to audio processing, there are some assumptions that our model is built on:

- **Assumption 1:** Users' audio input device does not contain active noise cancelling functions.

- **Assumption 2:** Users do not sing with the technique of polyphonic overtone singing.
- **Assumption 3:** Signal and noise are uncorrelated.
- **Assumption 4:** Noise is a stationary or slowly varying process.
- **Assumption 5:** Noise spectrum does not change drastically during the recording.

They will be explained in the later sections.

5.2 Noise Filter

Noise filtering is essential as it reduces or eliminates the noise present in the input signal. A conventional method to quantify noise is to use the signal-to-noise ratio (SNR), which is often represented in decibels.

$$SNR = 10 * log_{10} \frac{P_{signal}}{P_{noise}}$$

As its name suggests, SNR is the power ratio between the desired signal and undesired noise. Effectively, we would like to use noise filters to achieve a higher SNR. There are a few sources of noise when a user records himself with a microphone. Firstly, there exists self-noise, which is the instrument noise produced by the microphone itself. Noise may be induced or created when the signal passes through electronic components like transistors and printed circuit boards[1]. The second source, ambient noise, contributes to a large portion of noise present in a recording. Room reflections, extraneous noise, electromagnetic interference and mechanical noise are some causes of ambient noise.

5.2.1 Possible Models

Most of the noise filters work in the frequency and spectral domain, here we are going to inspect and compare three noise reduction mechanisms.

[1] **selfnoise**.

1. Low-pass filter (LPF)

   LPF passes signals with frequency $f < f_c$, where f_c is the cut-off frequency, and attenuates signals with $f > f_c$. In order to implement an LPF, we have to transform the signal from time domain to frequency domain using Fourier Transform (FT). An ideal LPF would completely remove frequencies that are higher than f_c and is a non-casual linear time-invariant system.

$$H(f) = rect(\frac{f}{2B})$$

$$h(t)) = \mathfrak{F}^{-1}H(f) = \int_{-B}^{B} e^{2(\pi)ift}\, df = 2Bsinc(2Bt)$$

   The impulse response of an LPF is a sinc function that extends to [-∞,∞]. This is why it is impossible to realize an ideal LPF since that will take infinite time and memory.

2. Wavelet transform

   Wavelet transform creates a representation of the signal in both time and frequency domain so localized information of the signal can be efficiently accessed. It is often compared with FT, which has the following limitations:

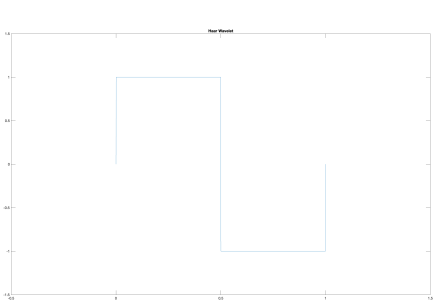
   (a) For windowed FT, if the feature is larger or shorter than the window, it cannot be captured completely.

   (b) Time resolution for high frequencies is the same for low frequencies. As frequency increases, the rate of change of the signal increases, and high-frequency signals contain more information in a window than that of low frequency, thus we need a higher time resolution for that.

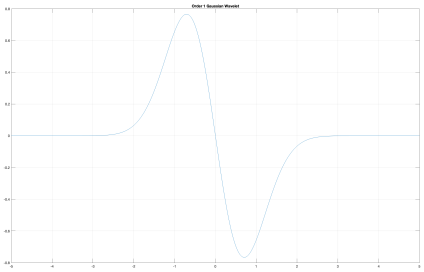
   Wavelet transform analyzes a signal by its different frequency components at multiple resolutions so features that are undiscovered at one resolution may be obvious at another. There are mainly 2 types of wavelet transforms, namely continuous wavelet transform (CWT) and discrete wavelet transform (DWT).

   CWT finds how alike a wavelet is in a signal, given the dilation and translation parameter of the wavelet[2]. This can be found by convolving the mother

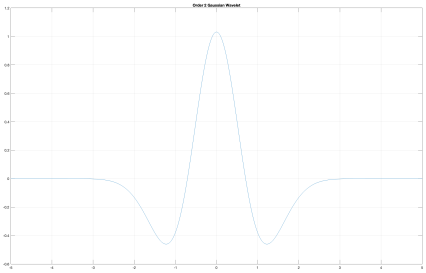
[2] **wavelet**.



(A) Haar wavelet



(B) Gaussian wavelet of order 1



(C) Ricker wavelet - Gaussian wavelet of order 2

FIGURE 5.2: Examples of wavelets

wavelet with our signal.

$$\mathrm{CWT}(a, b; x(t), \psi(t)) = \int_{-\infty}^{\infty} [x(t) \frac{1}{a} - \psi^*(\frac{t-b}{a})] dt$$

where $x(t)$ is the original signal, $\psi(t)$ is the mother wavelet, a is a dilation parameter and b is a translation parameter[3]. Dilation factor represents how dispersed the wavelet is (similar to scaling) while the translation factor tells us where the wavelet is positioned in time (similar to shifting).

The major difference between DWT and CWT is how the scale parameter is discretized. DWT discretizes scale parameters to integer power of 2 while CWT is more refined since the scale parameter is often raised to different fractional powers.

$$\begin{aligned} \mathrm{DWT}\ [n, a^j] &= \sum_{m=0}^{N-1} x[m].\psi_j{}^*[m-n], \\ \psi_j[n] &= \frac{1}{\sqrt{a^f}} \psi\left(\frac{n}{a^f}\right) \end{aligned} \tag{2}$$

[3] **wavelet_denoise**.

where n is delay parameter, N is the length of signal, ψ is the discretized mother wavelet[4].

DWT is often preferred in the context of real-time audio processing since computation is done on discrete wavelets which require less computational resources.

3. Spectral reduction

Spectral noise gating learns from a noise profile and removes slow-changing tonal noise or hiss from the signal. In fact, this method is used by Audacity in its noise reduction algorithm[5]. Suppose noise is additive, and we can represent our noisy audio as

$$y(n) = x(n) + d(n), \textit{ for } 0 \leq n \leq N-1$$

where $x(n)$ is our original signal (the signal we wish to recover), $d(n)$ is the noise, n is the discrete time index, N is the number of samples. Assuming $d(n)$ and $x(n)$ have no correlation, and we perform a short-time fourier transform on equation **??**:

$$Y(\omega, k) = X(\omega, k) + D(\omega, k)$$

where k is the frame number. Each frame will be of length N. We then have the desired signal in frequency domain:

$$X(\omega, k) = Y(\omega, k) - N(\omega, k)$$

Since the statistics of the noise is unknown, we try to find an estimate of noise spectrum by calculating the time-averaged noise spectrum using parts of the recording that only contain ambient noise[6].

$$\hat{N}(\omega, k) = \mathbf{E}[|N(\omega, k)|] = (1/N) \sum_{i=0}^{N-1} |N_i(\omega, k)|$$

[4] **wavelet_denoise**.
[5] **audacity**.
[6] **reductionmanual**.

We then get the estimated signal spectrum

$$\hat{X}(\omega, k) = Y(\omega, k) - \hat{N}(\omega, k)$$

We then set a gain control for each frequency band so if the sound exceeded the threshold, the gain is set to 0 dB or a user-defined constant.

5.2.2 Choice of model and implementation

After trying to implement all 3 methods, a major difficulty encountered is that it is hard to set the parameters to implement for LPF and wavelet transform. For example, since f_c depends on the pitch range of the user and the melody he/ she is inputting, finding an adequate f_c that separates desired frequencies from undesired ones is hard. As for wavelet transform, finding an adequate mother wavelet is a difficult task.

Although wavelet transform works better for real-life non-stationary signals compared to conventional frequency-based filters, if we do not feed a suitable mother wavelet, the performance is unsatisfactory, the model cannot distinguish between desired and undesired signals and will decrease P_{signal} at the same time, which is unfavourable when it comes to improving the SNR.

According to **complexwt**, there are 2 major concerns with using wavelet transform. Firstly, it is sensitive to shifts in time, even a minor shift will cause an unpredictable change in transform coefficients which will then cause variations in the output signal. Secondly, wavelet transform suffers from poor directionality easily. For example, a 2-D DWT can only reveal 3 spatial-domain feature orientations, which limits the optimal representation of the signal.

Therefore, we decided to use spectral reduction as our noise filter algorithm since it is the most effective in removing the ambient noise and wideband noise. Note that this method heavily relies on the assumption that the noise spectrum magnitude is staying locally stationary. If this assumption is not satisfied, this method will result in poor performance of either not being able to filter a majority of the broadband noise or removing the features of the signal. The implementation of spectral noise gating can be summarised according to **spectralflowchart** as shown in **??** The noise

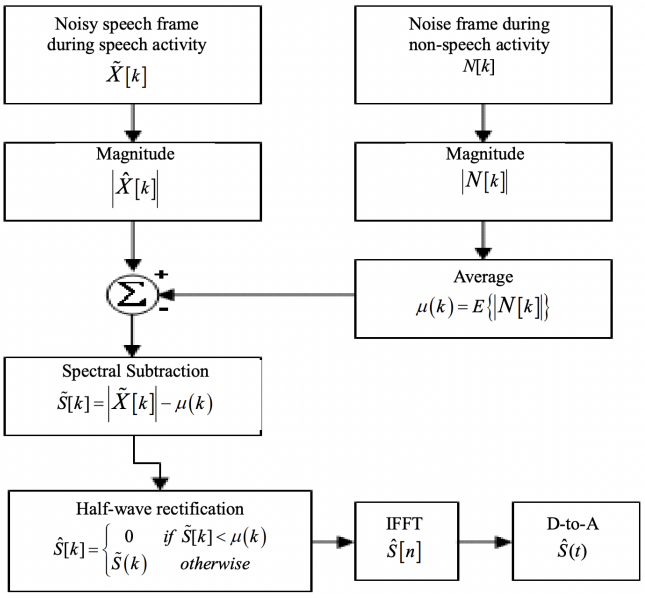


FIGURE 5.3: Spectral noise gating flowchart

spectrum $N(\omega, k)$ and its statistical measures are obtained by asking users to record at least 3 seconds of silence before they sing into the app.

Note that half-wave rectification is necessary after the noise removal process of subtracting the average magnitude of the noise spectrum.

$$\left|\hat{X}(\omega,k)\right| = \begin{cases} ||| \hat{X}(\omega,k) & \text{if} \hat{X}(\omega,k) \leq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{5.1}$$

This is to target frequencies that have a higher average magnitude of noise spectrum $\mathbf{E}[|N(\omega,k)|]$ compared to that of the noisy speech spectrum $|\hat{X}(\omega,k)|$. For those frequencies, we would replace the negative values with 0 with a half-wave rectification.

5.2.3 Improvements

A drawback with spectral reduction is that it does not handle extreme responses nicely. It does not reduce noises like squeaks. Also, since a half-wave rectification is included in the implementation process, **spectral_drawback** has pointed out that residual noise will be created during the process of spectral reduction. Half-wave rectification introduces nonlinearity in the $\hat{X}(\omega,k)$ spectrum and results in frequencies changing abruptly between frames.

	Male	Female
Pitch range (Hz)	60-180	160-300
Praat pitch range (Hz)	50-300	100-600

TABLE 5.1: Praat pitch range

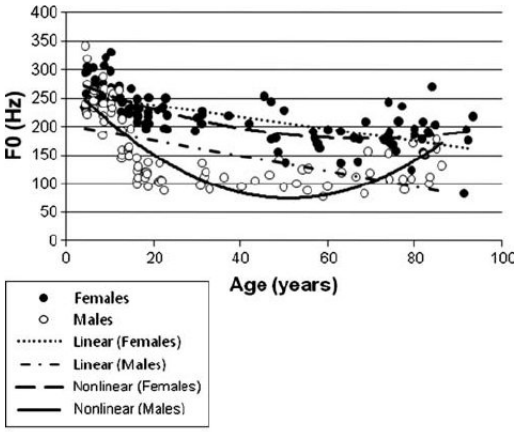


FIGURE 5.4: Scatter plot of fundamental frequency by age

To reduce the residual noise, we can modify the rectification so that when the estimated signal spectrum is negative (i.e. when the magnitude of noise spectrum is larger than that of the noisy signal spectrum), we use the noisy signal spectrum magnitude.

$$\left|\hat{X}(\omega,k)\right| = \begin{cases} \left|\hat{X}(\omega,k)\right| & \text{if} \hat{X}(\omega,k) \leq 0 \\ \left|Y(\omega,k)\right| & \text{otherwise.} \end{cases} \tag{5.2}$$

On the other hand, as mentioned above, spectral reduction is built on the assumption that the noise is stationary or slowly varying. Yet in reality, there may be sudden squeaky noise in the background which is not recorded in the noise profile. In this case, spectral reduction cannot remove the squeak. To improve the situation, we will introduce a low-pass filter to filter out high-frequency noise. The reason for choosing LPF over a bandpass filter is that spectral reduction is effective in targeting the reduction of ambient noise, which is usually low frequency. Thus as to avoid removing low-frequency desirable features, a low-pass filter will suffice.

To determine f_c for the LPF, it would be plausible to refer to the biological features of the users, i.e. their age and gender. For males, the pitch level generally reduces from infancy to middle age, while a reversal of the trend occurs after middle age. On the other hand, as pointed out by **womenprange**, "Females in their 30s and 40s showed obviously lower frequencies than those in their 20s. Across all age groups, including the 80s, fundamental frequencies tended to decrease markedly in association with aging"

After referring to the measurements taken from 192 participants[7] and the pitch range set by Praat[8] (a software for speech analysis), a simple modelling of f_c according to age and gender is as below:

$$f_{c,male}(n) = 0.07n^2 - 7.5n + 280, \text{ for } 4 \leq n \leq 93$$

$$f_{c,female}(n) = 0.02n^2 - 3n + 287, \text{ for } 4 \leq n \leq 93$$

where n is the age of the user In deciding which filter to implement, there are 3 filters in consideration:

(a) Type 1 Chebyshev filter

$$G_n(\omega) = |H_n(j\omega)| = \frac{1}{\sqrt{1 + \varepsilon^2 T_n^2(\frac{\omega}{\omega_0})}}$$

where ε is the ripple factor, ω_0 is the cut-off frequency and T_n is a Chebyshev polynomial of the nth order.

(b) Butterworth filter

$$G_n(\omega) = |H_n(j\omega)| = \frac{1}{\sqrt{1 + (\frac{\omega}{\omega_0})^{2n}}}$$

where ω_0 is the cut-off frequency and n is the order of filter.

(c) Bessel filter

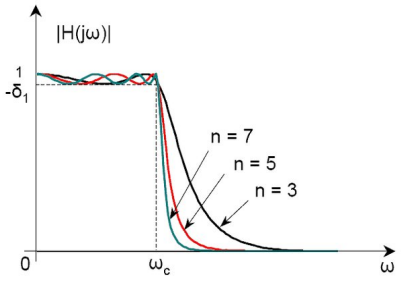
$$G_n(\omega) = |H_n(j\omega)| = \frac{\theta_n(0)}{\theta_n(\frac{j\omega}{\omega_0})}$$

where $\theta_n(j\omega)$ is a reverse Bessel polynomial and ω_0 is the cut-off frequnecy

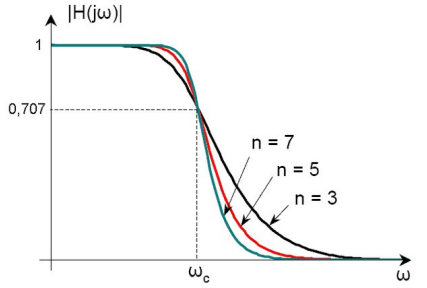
Chebyshev filter has a steeper roll-off compared to Butterworth and Bessel filter, but it also brings passband and stopband ripples, unlike Butterworth and Bessel filter which have a flat passband and stopband as they roll off towards zero. Moreover, Butterworth and Bessel filters have a better step response. Meanwhile, Bessel filters perform the best in step response since the overshoot is minimal. Another benefit

[7] **f0age**.
[8] **praat**.

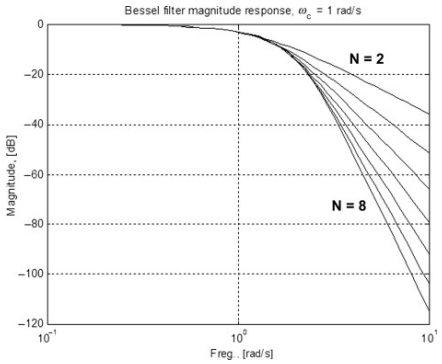
Bessel filter has is it introduces a linear phase and constant delay for $f < f_c$. This feature allows us to preserve the waveshape since all frequencies are delayed by the same amount. Thus, a Bessel filter is preferred in this context.



(A) Type 1 Chebyshev filter frequency response



(B) Butterworth filter frequency response



(C) Bessel filter frequency response

5.3 Pitch Detection Algorithm (PDA)

After removing noise, we pass the processed signal to a PDA to estimate the fundamental frequency (f_0) of the signal.

5.3.1 Possible Models

There are 4 approaches to detect f_0, which can be classified into the time domain and frequency domain.

1. Zero crossings (time domain)

   This is the most intuitive method to detecting pitch although it suffers from low accuracy. Assuming the input is monophonic, the fundamental frequency is estimated as:

$$f_0 = \frac{P_{zcr} f_s}{2N}$$

where P_{zcr} is the number of zero-crossing points, f_s is the sampling rate, N is the number of samples.

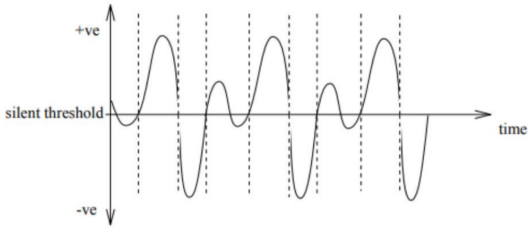


FIGURE 5.6: An example signal with zero-crossings marked in dotted lines[9]

2. Harmonic Product Spectrum (HPS) (frequency domain)

   As aforementioned, human voice is not of pure tone, a musical note sung will consist of a series of peaks in its frequency spectrum, in which the peaks correspond to f_0 with other peaks indicating the harmonic components of integer multiples of f_0. Exploiting this fact, HPS algorithm creates multiple down-sampled signal spectrums and compares them with the original spectrum as shown in figure **??**. The strongest harmonic peak will line up no matter how many times we compress the spectrum.

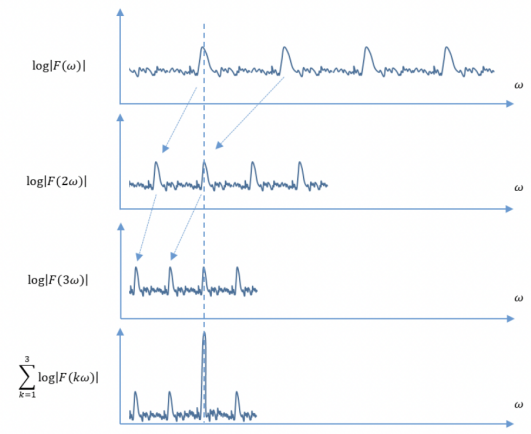


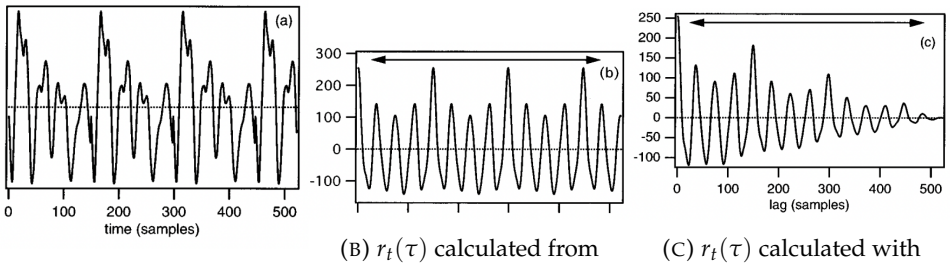
FIGURE 5.7: Harmonically compressed log spectra[10]

Firstly, we convolve the signal with a Hanning window to segment the input:

$$w(n) = \frac{1 + cos(2\pi n/N - 1)}{2}, \text{ for } 0 \leq n \leq N - 1$$

where N is the number of samples.

We then convert it from time-domain to frequency-domain by computing the



(A) Signal waveform

(B) $r_t(\tau)$ calculated from figure **??** using normal ACF

(C) $r_t(\tau)$ calculated with equation **??**

short-time Fourier Transform:

$$STFT\{x[n]\}(k,\omega) = X(k,\omega) = \sum_{n=-\infty}^{\infty} x[n]w[n-k]e^{-j\omega n}$$

Lastly we compute the product of spectrum at harmonics of various frequencies and f_0 is estimated by:

$$f_0 = argmax \prod_{k=1}^{n} |X(kf)|$$

3. YIN algorithm/ autocorrelation (time domain)

   As outlined by **yin**, YIN algorithm is based on a slightly altered autocorrelation method:

   $$r_t(\tau) = \sum_{j=t+1}^{t+W-\tau} x_j x_{j+\tau}$$

   where $r_t(\tau)$ is the autocorrelation function (ACF) of lag τ calculated at time index t, W is the integration window size. Note that as τ increases, W decreases and the envelope of the function decreases, as shown in figure **??**.

   We then select the highest peak by exhaustive search within a user-defined range of lags, the corresponding time lag will be the inverse of our estimated f_0.

   To improve the error rates and target periodicity, de Cheveigné & Kawahara introduced a cumulative mean normalized difference function (CMNDF) to replace ACF.

   $$CMNDF(\tau) = \begin{cases} 1 & \text{if } \tau = 0 \\ \frac{DF(\tau)}{(1/\tau)\sum_{j=1}^{\tau} DF(j)} & \text{otherwise.} \end{cases} \tag{5.3}$$

We then find τ that minimises $CMNDF(\tau)$ and the corresponding f_0.

4. CREPE (Convolutional Representation for Pitch Estimation) (Machine learning method)

   CREPE is a data-driven algorithm developed by Kim et al. that operates directly on the time domain. It consists of a deep convolutional neural network (CNN) trained by synthesized audio from the RWC Music Database[11] and MedleyDB[12].

   CNN is often seen in image-processing applications and is a network that makes use of convolution instead of the typical matrix multiplication. It consists of an input layer, hidden layers and an output layer. Hidden layers include convolution layers, pooling layers and fully connected layers.

   As shown in figure **??**, there are 6 hidden layers and each layer is followed by a dropout layer with a dropout probability of 0.25.

   Convolution layers are the building blocks of CNN since it performs feature extraction through convolution and activation functions like $ReLU$. Two hyperparameters defining a convolution operation are the kernel size and the number of kernels. Kernels in the convolutional layer context are convolutional filters, so kernel size refers to the size of the filter mask and the number of kernels relates to the number of output features desired. Figure **??** shows the hyperparameters used in CREPE.

   Max pooling is often used in the operation of the pooling layer and is the operation used in CREPE. It outputs the maximum value in a patch extracted from the input tensor and discards the non-maximum values. One advantage of using max-pooling is that it suppresses noise better than other dimensionality reduction methods like average pooling.

   A fully connected layer maps all extracted features in one layer to every activation unit of the next layer. In the context of CNN, it is often seen in the last few layers to compile the features for final output.

[11] **rwcdb**.
[12] **medleydb**.

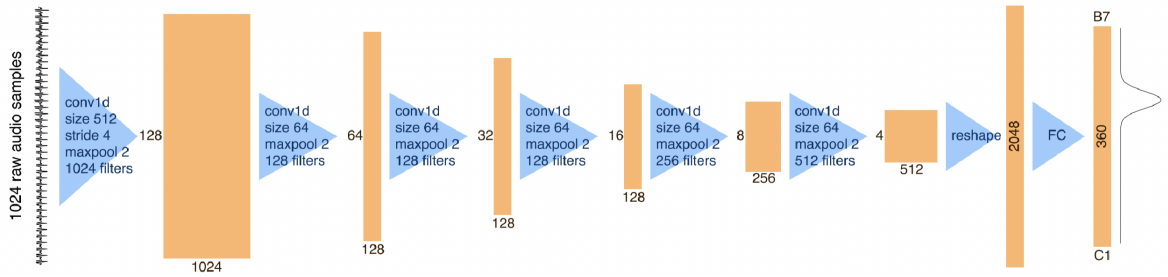


FIGURE 5.9: The architecture of the CREPE algorithm[13]

5.3.2 Comparison between models and implementation

Time domain models are more intuitive and easier to understand when compared to frequency domain models. Also, the time domain implementation often takes up less computational resources and time. In exchange, frequency domain models are less sensitive to noise and perform well on polyphonic singings.

The zero crossings method will not be considered though it has the cheapest computational cost. One limitation is that the threshold is fixed at 0, making it susceptible to noise and the vocal timbre of the user. Moreover, the method heavily relies on the assumption that the input audio is of pure tone but unlike a tuning fork, human voice is not a pure tone. It is composed of a fundamental frequency and upper harmonics[14]. This characteristic makes this method extremely unreliable.

Like the zero-crossings method, HPS is intuitive and has a low computational cost. But since it builds on the characteristics that the signal contains amplitudes at harmonics, if the input signal does not have sufficient magnitudes at other harmonics, the performance of HPS will suffer. Moreover, the resolution of the method depends on the length of the short-time Fourier Transform, which determines the number of discrete frequencies that we can consider. But if we want a higher resolution and less graininess in our pitch output, more time is needed in performing the transform.

YIN has a more satisfactory performance but when the pitch varies rapidly (for example if the user has a breathy timbre), it cannot estimate the pitch correctly as the algorithm uses a constant threshold. On the other hand, Kim et al. mentioned CREPE has a higher tolerance for inputs with different timbres since CREPE

[14] **humanmono**.

is trained with MedleyDB which contains recordings of heterogeneous timbres. Figure **??** concludes the Raw Pitch Accuracy and Raw Chroma Accuracy for CREPE and other 2 PDA by Kim et al. and we will choose CREPE to implement in our model.

Dataset	Metric	CREPE	pYIN	SWIPE
RWC-synth	RPA	**0.999±0.002**	0.990±0.006	0.963±0.023
	RCA	**0.999±0.002**	0.990±0.006	0.966±0.020
MDB-stem-synth	RPA	**0.967±0.091**	0.919±0.129	0.925±0.116
	RCA	**0.970±0.084**	0.936±0.092	0.936±0.100

FIGURE 5.10: Raw Pitch Accuracy and Raw Chroma Accuracy with the standard deviations for the 3 PDA tested by Kim et al.[15]

The CREPE code uploaded by Kim et al. takes in the samples y, sampling rate sr and a boolean parameter (True/ False) for Viterbi, which is a smoothing algorithm. It calculates pitch every 10 ms and outputs the timestamp, estimated frequency $\hat{f}_0$, confidence c and an activation matrix for visualization of outputs.

The sampling rate of training data used by CREPE is at 16 kHzssec:crepe thus our input audio will need to be resampled to 16 kHz if the original $sr \neq 16kHz$. If the user is using the iPhone built-in microphone to record, we need to set the preferred sampling rate to 16 kHz[16] when coding for our iOS app.

One thing to note is that the algorithm centers the first frame at $t = 0$ instead of starting the first frame at $t = 0$ to avoid misalignment.

Continuing from the noise filtered signal, we use *[y, sr]= scipy.io.wavfile.read(filename, mmap=False)* function to read the input and feed $[y, sr]$ into *crepe.predict(y, sr, viterbi=True)*. The output will be of the form [timestamp, $\hat{f}_0$, c, activation matrix].

We then manipulate the $\hat{f}_0$ array such that

$$\hat{f}_0 = \begin{cases} \hat{f}_0, & \text{if } c \geq 0.5 \\ 0, & \text{otherwise} \end{cases}$$

In order to output the predicted frequencies as notes, we need to prepare a dictionary of notes with its corresponding frequencies, i.e. *notesdict = "A4": 440, "A#4":*

[16] **iphoneaud**.

466.16, "B4": 493.88,... and map the predicted frequency to the closest frequency that corresponds to a note with *min(notesdict.items(), key=lambda (,freq) : abs(freq−f0))*

Finally, to target notes that lasted less than 10 timeframes (0.1s) (since they are likely to be misinterpreted/ noisy notes), we neglect the misinterpreted note and add half of its duration to the frequency in front and behind respectively, i.e.

$$T_{i-1} = T_{i-1} + \frac{T_i}{2} T_{i+1} = T_{i+1} + \frac{T_i}{2}$$

Where T_i represents the duration of the ith frequency.

5.3.3 Improvements

On top of the convolutional neural network model, we can integrate Bayesian statistics to improve the model.

Bayes' rule states:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

where $P(A \mid B)$ is the posterior probability or the updated probability with the consideration of the evidence.
$P(B \mid A)$ is the likelihood, which is the probability of observing the evidence given the event has happened.
$P(A)$ is the prior probability, which is the probability before taking into consideration the evidence.
$P(B)$ is known as the marginal probability, which is the probability of an event irrespective of the outcomes of any evidence.

With enough audio samples sung by the user, we can collect the range of pitches that he/she manages to sing. Then we can model the probability distribution $P(x)$ to update the likelihood of observing a certain pitch given the previous audio samples.

The prior $P(A)$ can be obtained from the demographic characteristics collected, i.e. age and gender, with the afore modelled equation **??** and equation **??**.

5.4 Key Detection Algorithm (KDA)

Other than detecting the pitch/ note of the input audio, it is also necessary to estimate the key of the music to aid the identification of chords with the machine learning model in the later step.

From a music theory point of view, the most intuitive method to identify the key of a piece is to look at the key signature which contains the number of sharps/ flats in a piece, and the number defines which key the piece is in. But since our app targets amateurs, we will not expect them to be able to know what key will they be singing in.

Krumhansl-Schmuckler algorithm (template-based)[17] is an experimentally measured tonal hierarchy is introduced by **templatedata**, in which it contains 24 tonal profiles for major and minor keys in total. Each of the profiles contains 12 values which correspond to 12 notes in an octave. The values were obtained from the experiment where they asked the listeners to judge how well a note fits in a key on a scale from 1 to 7 (1 stands for very bad, 7 stands for very good). We then calculate the correlation of distribution with these 12 major and 12 minor templates. The key with the highest correlation will be the estimated key.

Yet, one drawback is that the sense of key changes over time as affected by the evolution of music. While the method can still be utilised, the template will have to be updated from time to time.

5.4.1 Implementation

Assuming we are using the template from Krumhansl and Kessler[18] in our model and it is represented as 24 vectors $\vec{P_{i,k}} \in \mathbb{R}^{12\times 1}$, where i is the tone of the profile, j indicates whether the profile is in major or minor. We also define the pitch class distribution (our input) as a vector $D \in \mathbb{R}^{12,1}$, where the 12 entries are the frequencies of the 12 notes that appeared in the audio. $\vec{D}$ can be obtained by looping over the $\hat{f}_0$ array obtained in **??** and using the Python function *collections.Counter([iterable-or-mapping])* to count the number of occurrences. We then calculate the correlation

[17] **template**.
[18] **templatedata**.

Noise filter	Strengths	Limitations
Low Pass Filter	Easy and fast to implement	Cannot target broad band n Hard to set the cut-off frequ different environments
Wavelet Transform	Based on both time and frequency domain so localised information of the signal can be efficiently accessed.	Sensitive to shifts in time Poor directionality Hard to set a mother wavele
Spectral Reduction	Based on noise profile of the input recording so it is more customized	Requires user to input a noi background noise

PDA	Strengths	Limitations
Zero crossing	Intuitive Cheapest to compute	Susceptible to oise and voca
Harmonic Product Spectrum	Low computational cost	Unsatisfactory performance Performance relies heavily o
YIN/ Autocorrelation	Intuitive Less susceptible to noise	Inflexible since it is based or
CREPE	Robust and tolerant to singing with different timbres	Hard to interpret since it is a

between $\vec{P_{i,k}}$ and $\vec{D}$, which is essentially the dot product between the two.

$$corr(\vec{P_{i,k}}, \vec{D}) = \vec{P_{i,k}} \cdot \vec{D}$$

The estimated key (i, j) is found by looping over the 24 $\vec{P_{i,k}}$ vectors and finding the one that gives the maximum correlation.

5.5 Conclusion

In this chapter, we presented methodologies to filter noise, detect pitch and key. We will compile the strengths and weaknesses in this section.

Appendix A

Frequently Asked Questions

A.1 How do I change the colors of links?

The color of links can be changed to your liking using:

`\hypersetup{urlcolor=red}`, or

`\hypersetup{citecolor=green}`, or

`\hypersetup{allcolor=blue}`.

If you want to completely hide the links, you can use:

`\hypersetup{allcolors=.}`, or even better:

`\hypersetup{hidelinks}`.

If you want to have obvious links in the PDF but not the printed text, use:

`\hypersetup{colorlinks=false}`.