

## What's that Chord?

Submitted April 2023, in partial fulfilment of  
the conditions for the award of the degree BSc (Hons) Computer Science  
with Artificial Intelligence.

**20216424**

School of Computer Science  
University of Nottingham

I hereby declare that this dissertation is all my own work, except as indicated  
in the text:

Signature C Q

Date 24 / 04 / 2023

I hereby declare that I have all necessary rights and consents to publicly  
distribute this dissertation via the University of Nottingham's e-dissertation  
archive.\*

\*Only include this sentence if you do have all necessary rights and consents. For example, if you have including photographs or images from the web or from other papers or documents then you need to obtain explicit consent from the original copyright owner. If in doubt, delete this sentence. See [Copyright Information](#) for more details.

\*\*Only include this sentence if there is some reason why your dissertation should not be accessible for some period of time, for example if it contains information which is commercially sensitive or might compromise an Intellectual Property claim. If included, fill in the date from which access should be allowed.

# Acknowledgements

I would like to take this opportunity to extend my sincerest gratitude to my supervisor, Prof. Tony Pridmore for his invaluable assistance. He not only gave me patient guidance on the project as a professor but also taught me much musical knowledge as a good guitarist. Without his support, I would not be able to finish the project on time and reach this far.

# Abstract

Recognizing guitar chords in standard notation can be challenging for many guitarists, as most of them are accustomed to reading tab scores specifically designed for guitar. Therefore, this study focused on creating a system capable of automatically recognizing chords from sheet music. To enhance its utility, the system was also designed to handle camera-based input and process multiple chords simultaneously.

The implementation of the system involved three main parts. Firstly, a deep learning model based on CNN (Convolutional Neural Network) and LSTM (Long Short-Term Memory) using CTC (Connectionist Temporal Classification) loss was created to recognize musical symbols from input scores. A pre-processing pipeline was employed to enhance the image quality, and a data augmentation technique was applied to extend training dataset and simulate camera-based photos. Secondly, musical skills were used to encode the results from first part into numerical representations, while also reducing uncommonly-used chords. Subsequently, multiple machine learning models, including SVM (Support Vector Machine), Random Forest, and ANN (Artificial Neural Network) were designed and compared for chord recognition using the encoded data. Techniques such as focal loss and super-sampling were employed to address the issue of unbalanced dataset, by increasing weights and samples for minority classes. At last, a local Windows application was developed to visualize the two algorithms and display the recognized chords.

In conclusion, this project successfully designed and implemented several models, resulting in an accuracy of 85% in the test set for guitar chord recognition. Furthermore, an application was developed to facilitate easy utilization of the system.

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Motivation . . . . .	4
1.2	Background . . . . .	4
<b>2</b>	<b>Related work</b>	<b>5</b>
2.1	Note Recognition . . . . .	5
2.1.1	Conventional Methods . . . . .	5
2.1.2	Deep Learning Methods . . . . .	5
2.2	Chord Recognition . . . . .	6
2.2.1	Conventional Methods . . . . .	6
2.2.2	Machine Learning Methods . . . . .	6
<b>3</b>	<b>Design Methodology and Implementation</b>	<b>7</b>
3.1	Description of the Work . . . . .	7
3.2	Note Recognition . . . . .	8
3.2.1	Dataset Creation . . . . .	8
3.2.2	Image and Label Generation . . . . .	9
3.2.3	Image pre-processing . . . . .	10
3.2.4	Data Augmentation . . . . .	11
3.2.5	Model Design . . . . .	13
3.3	Chord Recognition . . . . .	16
3.3.1	Dataset Creation . . . . .	16
3.3.2	Data Pre-processing . . . . .	18
3.3.3	Model Design . . . . .	19
3.3.4	Model Improvement . . . . .	19
3.4	User Interface . . . . .	20
<b>4</b>	<b>Experiments Results and Evaluation</b>	<b>25</b>
4.1	Environments . . . . .	25
4.2	Note Recognition Result . . . . .	25
4.3	Chord Recognition Result . . . . .	27
4.4	Unitest for User Interface . . . . .	31
<b>5</b>	<b>Summary and Reflection</b>	<b>32</b>
5.1	Project Management . . . . .	32
5.2	Contribution . . . . .	33
5.3	Reflection and Future Work . . . . .	33
5.4	Reply to LSEPI . . . . .	34

# 1 Introduction

## 1.1 Motivation

The guitar has always been one of the most popular musical instruments since its inception. Regardless of the time period, many people are engaged in playing it. However, guitar learning has never been an easy task, and various problems plagued beginners, such as chord alteration, rhythm recognition, etc. Among them, chord recognition is perhaps the most difficult one. Because the number of chords is huge and they are even unable to count, guitar learners need tremendous time to practice and familiarise themselves until they can quickly switch between different chords. In this case, being able to recognize chords quickly can speed up chord alteration and facilitate to play of smooth music. Thus, guitarists invented six-line tab scores to provide a visual representation of the chords used in a song since standard notation in traditional five-line scores is too difficult. As a result, most people would like to learn guitars from tab, and most of them, even the advanced guitarists could only read this kind of score.

However, when guitarists play complex songs, especially with other instruments in a band, there is usually just a standard five-line score. They may struggle with chords in standard notation consisting of at least three or more notes, and all placed in the same column [1]. Due to the high note density, many chords look quite similar, making it challenging for unfamiliar guitarists to quickly recognize them. This can result in delays during performances, as guitarists may take several seconds to identify the notes on the sheet music. Moreover, after recognizing notes, guitarists must then match them with the correct chord before playing, so many learners give up at this stage. Therefore, guitarists, especially beginners, need a tool for automatically identifying the input scores in standard notation.

## 1.2 Background

With the development of computer science, more and more people have been paying attention to a new area called music technology. They tried to use the computer to deal with the above difficult challenges, including note recognition. This area is called OMR (Optical Music Recognition), which was defined by Bainbridge and Bell in 2001 based on the concept of OCR (Optical Character Recognition) [2]. In essence, research in this field is focused on using computer vision technology to analyze the images of music scores and transcribe them into electronic versions, which can be used to guide guitarists in playing chords and other musical elements.

This field was proposed more than twenty years ago, and there were only a few papers to discuss how to solve it initially. Not only was OMR too small and niche, but scientists failed to use traditional computer vision skills to develop robust and accurate methods in the early years. Until AlexNet was introduced in 2012, its accuracy in image recognition and classification surpassed other algorithms of its generation, bringing computer vision into a CNN (Convolutional Neural Network) era [3]. OMR problem has also made great progress with the CNN algorithm represented by AlexNet, and many subsequent networks like RNN (Recurrent Neural Network) have also been proven to be a good solution. Nevertheless, in order to make it suitable for real scores, specific environmental issues also need to be considered, including lighting, font of notes and so on. In addition, OMR is a general question, so the previous scientists used not only guitar chord images to train their models. Consequently, their model might not be accurate for this project since guitar chords in standard notation have high note density, but most other music scores contained many single notes.

Plus, note recognition is only the first step of the guitar chord recognition problem, and another model is needed to translate the notes into chords. Specifically, a machine learning model like linear regression, random forest and so on is quite suitable given that the notes recognised in OMR could be treated as machine learning input. But again, some traditional algorithms may fit this problem, like calculating the string distance between each chord. However, the machine learning model is proved to be more robust to a certain degree of noise after enough training with data [4]. Meanwhile, the machine learning model is good for handling a specific problem in chord recognition. For instance, it is not easy to ensure that there are equal numbers of chords for each category. That means there is a problem of unbalanced dataset in this project, and it could be solved with a special loss function in machine learning to give higher weights to classes with fewer examples.

As a result, the chord recognition system was divided into two parts in this study. Firstly, a deep learning model has been developed to recognize notes from sheet music based on the current OMR research. Secondly, a machine learning model was used to classify the recognized notes into chords. Additionally, a local Windows application was developed to visualize the two algorithms and display the recognized chords.

## 2 Related work

### 2.1 Note Recognition

#### 2.1.1 Conventional Methods

The first task of the project is note recognition, and this is also called the OMR problem. The goal of OMR is to recognize musical symbols from scores, and this topic is actually a sub-area of OCR in the early days. But compared to recognizing black letters on white paper, OMR was much more difficult. For example, the Canny operator was proposed in 1986 to detect the edge in the image by gradient change [5]. In 1991, some scholars from UBC had already integrated it into the OCR system, which proved to be a considerable improvement in that area. However, the same algorithm did not work well in OMR since the background of it is not white paper but five black staff lines. Although Rossant and Bloch found that horizontal filtering and other line detection methods could be used to remove the lines, parts of the notes would be lost as they are mixed with the staff [6].

Furthermore, score lines are not the only factors that make OMR such difficult, and so is the shape of the notes. It differs from the alphabet in OCR, which has obvious distinct shapes. Notes are almost identical, and the normal way to distinguish them is their position in the score, specifically the height. Therefore, template matching based on different shapes from Rossant and Bloch's paper didn't work well either [6]. Customising templates and rules for notes may not be effective when the height is indeterminable due to the fact that input images are sometimes rotated and the notes' fonts are different.

Rebello et al. raised an approach called 'reference length' to handle the above problem based on previous image processing methods, including binarization, line detection, symbol segmentation and etc. [7]. Once they had detected the black lines, they recorded the distance between the lines in each image and calculated the pitch by combining the relative position of the notes within them. However, this system still had a low accuracy since they wanted to detect not only notes but also key signatures, time signatures and other music symbols, and even the authors admitted in the paper that it was not a reliable system [7].

Thus, some scholars only focused on the shape and characteristics of the notes specifically rather than using general image processing. For instance, Raphael et al. analysed the morphological attributes of notes and divided them into several small blocks for detection respectively [8]. They got a good accuracy for a standard style of notes through complex graphic analysis. However, this method was obviously not that robust either, as the notes' font might change.

#### 2.1.2 Deep Learning Methods

The real breakthrough for OMR actually came with the introduction of deep learning methods since Lecun et al. proved that deep learning could dramatically improve the accuracy of OMR [9]. For instance, Pacha et al. designed a network based on VGG to recognize single notes and other music symbols in 2018, and they achieved a 98% accuracy on their test set [10]. VGG they used is a classic CNN algorithm developed by Oxford University, and it contained up to 19 convolutional layers and a max pooling layer [11]. With the very deep layers, VGG gained the best learning ability at that time.

Although Pacha et al.'s model has successfully learned the features of a single music symbol with such a deep network [10], the notes in the score often appear consecutively, and it would be inefficient to identify them one by one. Moreover, one of the most significant characteristics of scores is that the preceding symbols may affect the meaning of the subsequent symbols, including sharp and flat symbols, key signatures, etc. Therefore, a new neural network is used for the OMR problem, and it is called RNN (Recurrent Neural Network).

RNN differs from CNN in that the nodes of this network can be connected in a loop, which means that it allows the output of some nodes to affect subsequent nodes directly [12], and this

gives it the ability to learn the relationship between different music symbols. Furthermore, if a neuron in RNN's hidden layers is replaced by a memory block structure, then this network is called LSTM (Long short-term memory). LSTM's memory block holds information from previous neurons longer and provides algorithms to filter out unused parts [13]. Chen used the features of LSTM to design an OMR model that can accept one-line input from a score [14]. Although his model could handle a line of scores, it could only accept monophonic scores, which means that complex chords in the polyphonic score were unrecognisable (Figure 1 shows the sample of monophonic score and polyphonic score separately).

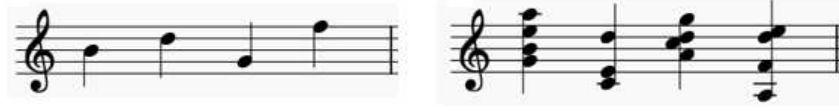


Figure 1: Monophonic vs. Polyphonic

After that, in order to handle the above problem, a group of researchers from UCSD raised an auto-encoder based on RNN and LSTM. They increased the complexity of the network and made it more capable of learning. Thus, it had an appreciable performance in the complex polyphonic spectrum (with high note density) from MuseScore Forum [15]. However, since the scores shared on the above forum are created by MuseScore, a professional music software [16], their model lacked the ability to handle the images captured by the camera in the real world. To address this issue, Ng and Nguyen designed a new pipeline with special pre-processing techniques, including erosion, dilation, bilateral filter, etc. Their model could more accurately deal with images with noise, distortion, uneven lighting, and so on [17]. Meanwhile, Na Li also designed a GAN (Generative adversarial network) to denoise the images, and his model achieved a good result in real scores [18]. This study would develop a deep learning model based on RNN and LSTM and also combine some pre-processing techniques to enhance the model's generalizability to handle real-world photos.

## 2.2 Chord Recognition

### 2.2.1 Conventional Methods

When note recognition outputs the results, they are typically 3-6 notes and a key signature that determines the pitch. For this problem of classification by string, Ristad gave a possible method in 1998 using edit distance. It calculated the number of insertions, Deletions and substitutions needed to edit the string to truth ground [19]. Chords recognition could be solved by getting the edit distance of each output and chords, then selecting the smallest one as the predicted result. However, edit distance has a serious drawback because it can not correctly reflect the similarity of different notes. The three notes in Figure 2 are B4, C5, and B5, respectively. Obviously, the difference between B4 and C5 is smaller, but their edit distance is 2, larger than 1 between B4 and B5. This incorrect result represents the inevitable failure of this simple approach in chord recognition. Thus, complicating the classification rules might be a good idea to make the prediction more robust. Ishibuchi et al. proposed a classification model using fuzzy logic, and they got a high accuracy in a particular binary classification problem [20]. But their model contained more than 90 fuzzy rules, all of which apply only to that problem. It is conceivable that fuzzy logic for the classification of hundreds of chords is a massive workload that is even unrealistic.



Figure 2: Problem of Edit Distance

### 2.2.2 Machine Learning Methods

Machine learning is more of a good solution here for chord recognition because it is not as simple as edit distance and does not require as many specific rules as fuzzy logic. The first idea

of machine learning is KNN (K Nearest Neighbours), representing the predicted data using its k nearest neighbours [21]. For example, the edit distance mentioned above can be combined with it, considering k minimal edit distances would have a better result instead of only one. But in fact, the increase is limited because its rule is still too simple. To add more complexity, logistic regression could be chosen. Mizutani et al. proposed a logistic regression model to predict the emotional expression from scores [22]. However, the accuracy was not that good since logistic regression is a linear model that assumes data is linearly divisible, which is not always true in real situations.

Therefore, some scholars tried to build a non-linear model random forest to handle music information. Random forest was designed by Breiman, and it got the most likely classification by generating a large number of decision trees and letting each tree vote separately [23]. But the random forest model is hard to get a good result by parameter tuning because it has a few optional parameters except the number of trees and the depth, so it is impossible to find a particularly important parameter for the chord problem. After that, Qing et al. created an SVM (Support Vector Machine) to predict the Chinese zither's duration of each music note based on 100 pieces of special music symbols [24]. SVM model they used aimed to find a line with the largest interval in the feature space for classification [25]. Compared to the random forest, it has more parameters, such as regularisation, which can be used for adjusting the loss function, so it is more likely to fit a problem with many details. However, some studies showed that SVM is a costly algorithm, and it also lacks the ability to handle complex datasets [26].

Then ANN (Artificial Neural Network) that is able to deal with complex calculations has been explored for music-related recognition. Gillick et al. analysed the performance of different machine learning models, including KNN, SVM and ANN for solving jazz grammar recognition problems, and they found the ANN model had the highest accuracy among all [27]. This benefits from the fact that ANN can customise the depth of layers, the number of neurons and the loss function to fit complex problems, and also provides dropout modules to prevent over-fitting caused by complex model [28]. However, ANN was considered to be less interpretive because it might perform well in some complex cases but poor on some simple ones [29].

Hence, many scholars currently believe that the best way to choose a machine learning model is to test multiple choices with specific problems and datasets [30]. This was exactly the idea behind this study. Multiple models containing an SVM, a random forest, and an ANN were created and tested on the chord dataset in subsequent sections.

## 3 Design Methodology and Implementation

### 3.1 Description of the Work

This project aims to build a system capable of accurately recognising chords from input polyphonic scores in standard notation. The implementation of the algorithm is divided into two parts. The first part focus on predicting notes, key signatures and other useful musical components from input scores. And the second part focus on predicting chords with notes and key signatures from the first part. Although this is a research-based project, there is also an additional goal of creating an App to visualise the results produced by this system.

The key objectives this project achieved were:

1. Investigated current deep learning approaches for optical musical recognition. Plus investigated machine learning-based classification models to identify chords from those musical symbols.
2. Created a guitar chord dataset containing score images and labels for chord recognition.
3. Enhanced the quality of images to fit the subsequent model by pre-processing.
4. Added data augmentation to increase the amount of the data and mimic the camera-based photos.
5. Designed and developed a deep learning model to recognise musical components from polyphonic scores and tested the different parameters tuning.
6. Translated and numericalized the predicted symbols into a CSV file and filtered out non-chord data for chord recognition.



7. Used musical rules and frequency to reduce the number of predicted chords and adopted a super-sampling method to balance the chord dataset.
8. Designed and compared different machine learning models to predict chords and improved parameter tuning.
9. Designed an algorithm to output multiple results when the chords were hard to distinguish.
10. Developed a local Windows application which used the above algorithms to identify chords in standard notation and then output prediction with alternative guitar fingering and sound of the chord.

### 3.2 Note Recognition

Note recognition part completed point 2 to point 5 from 'Description of the Work'. The overall design was divided into the following five parts: data creation, image and label generation, image pre-processing, data augmentation and deep learning. The subsequent five subsections would explain each part's methodology and implementation in detail.

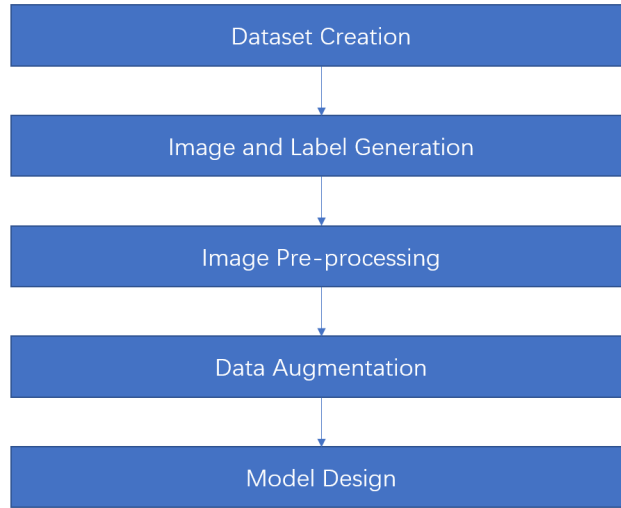


Figure 3: Design Pipeline for Note Recognition

#### 3.2.1 Dataset Creation

Since the study focused on a relatively small area, it was difficult to find a suitable dataset directly. Most previous OMR research has focused on general recognition, which covered different kinds of music, including piano, guitar, etc. And they did not distinguish monophonic and polyphonic scores in the dataset, either. Thus, every dataset in this project was created by myself from MuseScore's music forum [31]. MuseScore is the largest music writing App in the world, and they not only offer a free tool for proposing scores, but they also have a large and active community for people to share their works.

However, even in such a forum, finding a sufficient amount of guitar spectrum in standard notation is not easy. As the Introduction states, most guitar scores are six-line tab, not the five-line staff required for this project. Moreover, many songs do not have enough chords but instead many single notes. Therefore, two datasets were actually generated: One was the guitar chord dataset this project needed, and the other was a general dataset containing all kinds of instruments, such as the piano, the violin etc. The filter in MuseScore's forum helped to make it possible to classify the scores based on the instruments. As a result, the guitar chord and general datasets contained 66 and 360 scores, respectively. For their purposes, the general dataset was used to train the note recognition model, and the guitar dataset was only used as a test set to examine the performance of note recognition, and it would also be used to train and test the chord recognition model later.

The amount of these two datasets were different, while the method to generate them was the same. An open-source API called LibreScore was used to download mscz files from Muse Forum. This API was created by the users from the above forum to make it convenient for everyone to

find scores in posts quickly [32]. Mscz mentioned above is MuseScore’s proprietary file format, and every score written by this App will default to use this format. The biggest attribute of this format is that it contains not only the images of the scores but also the audio sources, labels and other information [33]. Thus, to create the guitar chord dataset, we manually decoded all guitar-based mscz files and checked their score images to ensure they were standard notations in file-line scores. Although there was a considerable amount initially, most guitar scores were tab scores or mixed with too many single notes, and only 66 of them were usable. By contrast, it was much easier to select 360 scores for the general dataset, and the only requirement was to filter out monophonic scores and keep only the polyphonic ones

Furthermore, in order to obey the open-source spirit and according to the data management form, these two datasets are now uploaded to Kaggle for other OMR researchers to use. Please find them using the links below: <https://www.kaggle.com/datasets/celeryq/guitar-chord-scores> and for guitar chord scores and <https://www.kaggle.com/datasets/celeryq/general-notes> for general scores.

### 3.2.2 Image and Label Generation

After that, the mscz files must be decoded into musicxml files and images, stripped of irrelevant audio and other information. Musicxml mentioned above is the most common format in music technology, and it aims to represent a song in code. It is a kind of xml file whose format is quite similar to HTML since its syntax has a number of tags, but the difference is that musicxml uses them to represent components of the music, including pitch, rhythm, volume, time, beats, etc., rather than the website. Due to this format’s simple, comprehensive, and compact nature, it quickly gained popularity when Good released it [34]. Therefore, MuseScore and other major music composing apps have already prepared some built-in plugins that convert their unique formats to musicxml.

However, current musicxml files could not be used directly to generate images and labels because they contained parts in their code that could interfere with the music, such as credits and rights. Therefore, a simple Python script was created to remove these parts from the musicxml code, and because this format was a standard xml file, it could be processed as a parse tree straightly.

Then, clean musicxml files were converted back to mscz format using the same built-in plugin mentioned above since MuseScore could not generate images from musicxml directly. Next, the new mscz files were decoded into musicxml files with score images. In addition, the images here did not represent a whole score because the plugins automatically cropped the score line by line. A single image actually represents a certain line of the score that can be directly used for model training. So to correctly use this system, multiple line inputs are not allowed, and it can only handle a line of the score. For instance, in the sample image below, the above one is illegal since it has two lines, and the next one is a legal input for this model (A few symbols from other lines in the upper or lower part of the image are allowed, and the model will gain the ability to ignore them in subsequent processing and training). Last, since the first part of each score was usually the title, author, etc., the first few images of each score would be irrelevant data. In addition, some images contained only rest notes or even blank lines. So a script was also created to check and filter out this kind of image without correct music symbols based on their musicxml files from the dataset.



Figure 4: Sample Image Input

After that, the above clean musicxml files were used to generate the labels we need to train our model since this format contained every component of the music in its code. So a Python script went through the musicxml files and extracts the required parts based on the tags. In this study, clefs, key signatures, sharp, flat and nature symbols, barlines, and the notes themselves were selected and connected as output. The sample label of the image is shown below, and this format refers to the rules defined by Alfaro et al. [35]. The difference is that label in this study does not focus on the complete expression of music content with string, so time signatures, rhythms, styles and other irrelevant elements in it are removed.



clef-G2 + keySignature-AbM + note-G4 note-C5 note-E5 + note-G4 note-C5 + note-G4 note-C5 + rest-half + note-G5 + barline + note-G5 + note-A5 + note-G5 + note-G5 + note-A5 + note-G4 note-C5 note-G5 + note-G4 note-C5 note-F5 + barline + rest-quarter + note-F4 note-A4 note-C5 note-F5 + note-F4 note-A4 note-C5 note-F5 + barline

Figure 5: Sample Label

It is clear that in the label figure above, each of the different components is connected by a "+". The first symbol is the clef ('celf-G2' in the sample above), which is also the first element of every line of the score, so it can be used to check whether the image input is legal. The subsequent nodes are key signatures and notes (starting from 'KeySignature-AbM' to the end). They are the most significant components of this study and will be used to recognise chords. The middle and ending barlines are used to distinguish sections and can also represent the end of the input. And because this study focused on the polyphonic scores, there was definitely a special treat for chords. Since chords were presented in the form of standard notation in those scores, a group of notes in a chord actually needed to be considered as a single component. Thus, each note in the chord was read from the bottom up and was connected by a space instead of "+". For example, in Figure 5, the "note-G4 note-C5 note-E5" represents the first chord in the image. Moreover, the sharp, flat and natural symbols in the input image were recognised with notes, and their sample labels are shown below. The transitivity of sharp and flat is also taken into account when labels are generated, such that the third note in Figure 6 is still affected by the flat symbol before the second note. It will change to a regular note until it meets the natural note later.

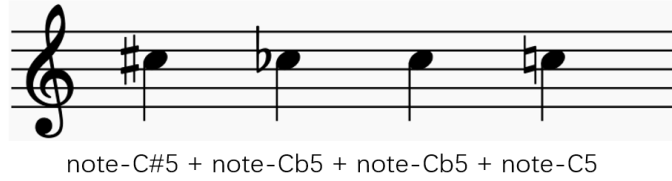


Figure 6: Sharp vs. Flat vs. Natural

Last, some musicxml files were broken, or the author of that file did not follow the standard creation requirement (starting without clef, ending without barlines, etc.), causing the labels were failed to generate from those files. Therefore, a Python script was created to go through and check the whole dataset to remove the images with incorrect or even no labels. After all of the processing, there were 6336 valid data in the general dataset and 584 in the guitar chord dataset.

### 3.2.3 Image pre-processing

When we got the images and labels, we could move to the image pre-processing part to fit the input of the training model and improve the image quality. This study aimed to recognise the chord images from the Musenscore and the camera, i.e. high-quality e-scores and low-quality real

scores. Thus, a robust pre-processing pipeline was needed to improve the quality of camera images and maintain high-quality images simultaneously.

The design in this experiment referred to the work of Ng and Nguyen, one of their OMR works focused on improving the accuracy of the camera input pictures. Their pipeline contained four main steps: close operation (dilation and erosion), Gaussian filtering, bilateral filtering, and median filtering [17]. They compared it with the other 40 methods (using different filters or changing the processing orders) on different camera inputs, and it had the most considerable average improvement for all input categories at about 11.1%. Plus, before Ng and Nguyen’s pipeline, a resize operation was used at first to scale the input image until its height reached 128 to fit the subsequent network. After that, an adaptive thresholding algorithm was added to this experiment to solve the uneven light problem and remove the shadow. This algorithm split the image into multiple 33x33 slices, calculated each area’s applicable local thresholds individually, and translated the image into binary according to the different thresholds [36]. Then the close operation was carried on as a standard morphological operation to smooth object contours and connect narrow discontinuities [37]. The other three filters aimed to remove all kinds of noise from the camera images, including Gaussian noise, salt and pepper noise, etc. [38–40]. For the parameter tuning, a 3x3 filter was chosen for the close operation, a 5x5 Gaussian filter was applied, a 5x5 bilateral filter was followed, and a 3x3 median filter was last. All these processes were implemented using the OpenCV frame in Python, and the parameters were tuned manually according to the test images. The effect of each step on a camera input image and a clean image is shown in the figure below (the left column images are the clean images, and the right column ones are the camera images). It is clear that the pre-processing successfully enhanced the quality of camera-based images to a generally good level, and clean images also did not degrade much because of this series of operations.



Figure 7: Image pre-processing Effect

### 3.2.4 Data Augmentation

Data augmentation was applied in this study because of two reasons. The first was to increase the data because the created dataset in this study was not large enough; the second was to simulate real-world photos and enhance the accuracy of the note recognition model for camera-based images. The data was insufficient because the OMR problem was challenging and requires more than 125,000 images to train the model to finally get a reliable result [15]. It was almost impossible for individual work to get enough data directly since this project needed to check manually whether the scores were polyphonic, and it already cost tremendous time to get 7000 images. Hence, data augmentation played a significant role in this case, increasing the size of the dataset by adding mirroring, rotation, etc. to the images [41].

While camera images could be simulated because of a particular data augmentation method in this study, the approach was based on an OMR experiment conducted by a team of researchers at the University of Alicante. They tried four different data augmentation methods to mimic camera photos and enhance the performance of their deep learning model on a real-world image dataset [42]. The technique they tested included contrast alteration, erosion and dilation operation, rotation transformation and wavy pattern, and the accuracy improvement compared to no augmentation are 1.20%, 2.45%, 4.95% and 3.71%, respectively.

In this study, the last three methods were chosen to implement based on OpenCV as the data augmentation process before deep learning. In erosion and dilation, a 1x2 filter was used, and the particular filter size referred to the shape of the note, which was a tall and narrow rectangle. Moreover, erosion and dilation would not be called simultaneously. Instead, each of them had a 50% chance of being selected. The scope of rotation was limited to  $[-3^\circ, 3^\circ]$ , and a random value in this range was applied to every input image. In addition, the distorted coefficient  $D$  was set to  $D = [0, -0.25, 1, 0]$ , whereas the camera matrix  $K$  was given below, where the  $w$  means the width of the image. The above two parameters determined the degree and method of distortion of the wavy pattern.

$$K = \begin{pmatrix} w/3 & 0 & w/2 \\ 0 & -w/2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (1)$$

There were two reasons why contrast alteration was not chosen in this study. On the one hand, it had a minor overall improvement according to Lopez et al.'s research [42]. On the other hand, contrast alteration did not make sense subjectively. Lopez et al. introduced the physical meaning of contrast alteration in their study because they wanted to simulate different colour distributions produced by image capture devices with varying types of lenses. However, an adaptive thresholding was applied in our pre-processing, so contrast alteration was unsuitable in this study since the colour distribution of every input is the same. By contrast, the other three methods all had their meanings to deal with some real-world issues. For example, erosion and dilation could simulate the notes with different sizes and thicknesses. Meanwhile, a slight rotation could simulate the user's natural photo effect. Because users are not machines, achieving an utterly horizontal picture is unrealistic. Wavy simulated the situation that users are taking photos of a guitar book, which is hard to lay on a table compared to a sheet of A4 paper because they have a curve due to its binding approach. In summary, the effect of data augmentation for each method and a total effect are shown below.

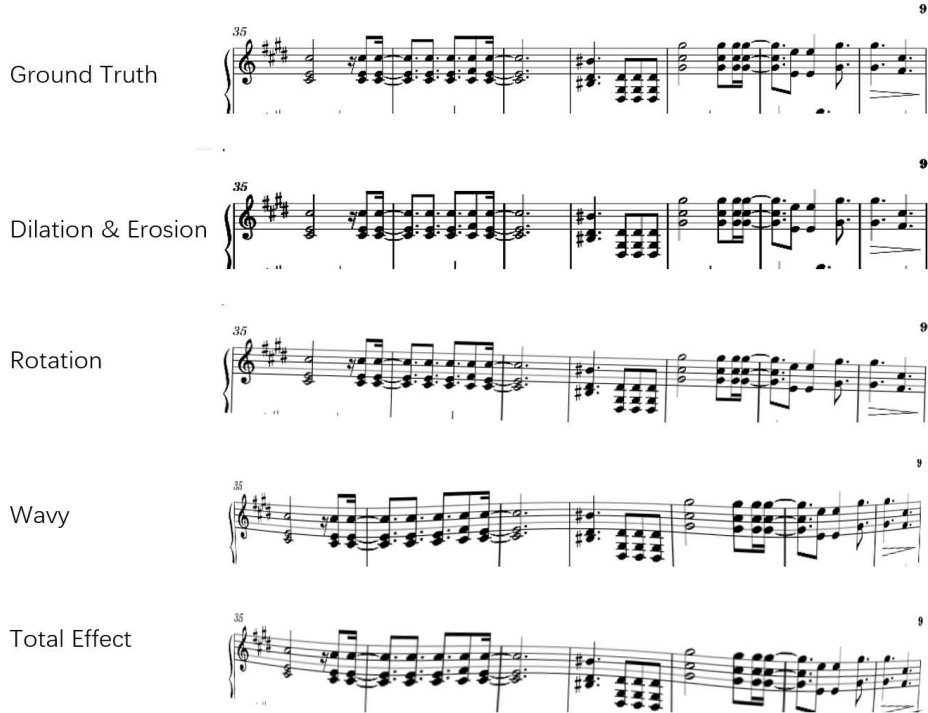


Figure 8: Data Augmentation Effect

### 3.2.5 Model Design

After the above pre-processing, the images and labels were finally ready to go to the training part. In this part, the encoder-decoder was used as the fundamental model structure. This structure was first combined with a neural network by Lecun and Fogelman in 1987. They used Multi-Layer Perception in their auto-encoder to denoise the image and got a generally good result [43]. Encoder-decoder is now a typical model in deep learning research, consisting of an encoder and a decoder structure. The encoder part  $f$  is used to reduce the dimension of the input data and to learn its hidden features  $H$  from input features  $X$ , and the decoder part  $g$  is used to reconstruct the original input data  $X$  from the hidden features  $H$  learned by the encoder with a minimal error like the equation below [43].

$$\begin{aligned} f &: \mathcal{X} \rightarrow \mathcal{H} \\ g &: \mathcal{H} \rightarrow \mathcal{X} \\ f, g &= \arg \min_{f, g} \|X - g[f(X)]\|^2 \end{aligned} \quad (2)$$

One of the critical reasons for encoder-decoder's popularity is their ability to be combined with different network architectures to meet various task requirements. In this study, the encoder was a typical CRNN (Convolutional Recurrent Neural Network) structure containing a CNN, a flattening layer, and an RNN. The design of it referred to Calvo-Zaragoza and Rizo's work but with a different pooling size in the CNN part [44].

The structure of the CNN in this study was similar to the structure of VGG because the width of the convolutional layer doubled after max pooling, which was an excellent configuration to learn hidden features in deep learning they have tested out [11]. The difference was that an additional batch normalization was added between the convolutional layer and the max pooling layer. Because batch normalization could make the input data have the same distribution and unify the scattered data so that accelerating the convergence process and improving the training speed [45]. In detail, the CNN actually comprised four blocks, each consisting of a convolutional layer, a batch normalization layer and a max pooling layer. The parameters of the blocks are shown in the table below. Conv( $n, h \times w$ ) represents the convolutional layer with  $n$  filters and  $h \times w$  filter size. MaxPooling( $h \times w$ ) represents the max pooling layer with  $h \times w$  filter size.

Block 1	Conv(32, $3 \times 3$ ), BatchNorm(), MaxPooling( $2 \times 2$ )
Block 2	Conv(64, $3 \times 3$ ), BatchNorm(), MaxPooling( $2 \times 1$ )
Block 3	Conv(128, $3 \times 3$ ), BatchNorm(), MaxPooling( $2 \times 1$ )
Block 4	Conv(256, $3 \times 3$ ), BatchNorm(), MaxPooling( $2 \times 1$ )

Table 1: Convolutional Blocks

It is clear that the CNN above would change the image's channel to 256 and reduce the height to  $1/16$  and the width by half. Because the input size of the image was  $(1, 128, W)$ , 1 represented the channel number (it is a binary image after thresholding), 128 represented the height, and  $W$  represented the width (see the image pre-processing section, it scales the image to fit the height). The size of the features after the processing of CNN blocks was  $(256, 8, W/2)$ , so there were actually 256 two-dimensional features with a height of 8 and a width of  $W/2$ .

Then the next part of the encoder was a flattening layer, it aimed to flatten all of the two-dimensional features to one dimension to fit the LSTM later. Thus, there would be 2048 (which is  $256 \times 8$ ) vectors with  $W/2$  length after flattening. If we reshaped the vectors to look at them vertically, there were  $W/2$  vectors with a length of 2048. In addition, each of these vectors could be treated as an encoding of an image slice, which was obtained by dividing the input image vertically  $W/2$  times. In summary, the CNN blocks and flattening layer were actually responsible for dividing the input image vertically into image slices with a width of 2 and then encoding each of the image slices to a one-dimensional vector with 2048 length. The sample of image slices is shown below. Although the current slice was relatively narrow, it had more slices than using a larger width. Moreover, Edirisooriya et al. have also proved in their paper that this size was good for learning the characteristics of musical symbols [15].



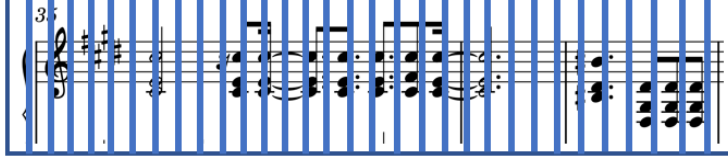


Figure 9: Image Slices

A group of bidirectional LSTM modules was chosen as the RNN part of the encoder because Baro et al. have proved that this structure was the most suitable one in the OMR research to connect each slice [46]. In detail, LSTM introduced a memory block compared to traditional RNN, so it could remember the input of previous slices longer and pass them into future predictions [13]. And bidirectional LSTM added an additional reverse LSTM, so not only did the preceding slices affect the prediction, but subsequent ones also participated in predicting the current image slice [47]. Thus, each prediction of the bidirectional LSTM took into account all other slices, which perfectly addressed the problem of predicting the notes based on all other music symbols in the score. For example, key signatures and natural symbols (see Figure 5 and Figure 6 to check them) were highly related to the prediction of chords, with the key signatures before the notes indicating the pitch on each string and the natural symbol after the notes indicating that the notes were still affected by a sharp or flat before them.

The structure of the model is shown in the figure below (s means input sequence, which is a vector; h means hidden layer; w means weight; o means output layer). The input layer had  $W/2$  neurons, and each neuron accepted a vector with a length of 2048 in sequence ( $W$  represents the width of the input image). The output layer also had  $W/2$  neurons, which means that the bidirectional LSTM modules did not change the size and dimension of the input vectors. However, the value of each output would be directly affected by the two weights from two LSTM chains. Each chain in this study contained 256 neurons (hidden layers), which was also a configuration tested by Calvo-Zaragoza and Rizo in their OMR works [44]. These hidden layers allowed LSTM to learn from other slices because their weights would update each sequence. Thus, although their results were still vectors with a length of 2048 representing the encoding of image slices, the values were changed based on other slices through this model.

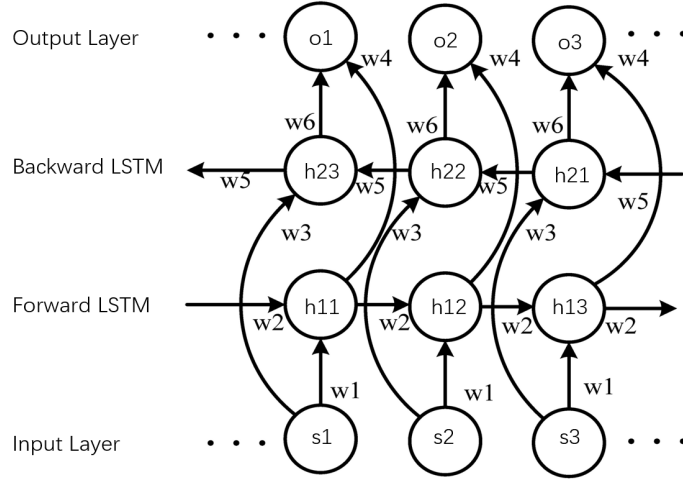


Figure 10: Bidirectional LSTM

The decoder was used to decode the image slices and predict music symbols from them. Due to the fact that the chord in standard notation contained multiple notes, there was more than one input in each image slice (see Figure 5's first component, it is a chord with three notes). Thus, Calvo-Zaragoza and Rizo's decoder design with a fully connected layer and a softmax was unsuitable here since it could only output one prediction in each image slice [44]. In order to solve this problem, an improved RNN was used. The structure of the RNN referred to Edirisooriya et al. [15], and it is shown in the image below. In detail, the RNN's hidden layer had 512 neurons, and its function was the same as the LSTM, which updated the weights so that the model could

generate different predictions more robust. Its input and output layers had ten neurons, but each neuron in the input layer accepted the same vector, which differed from the bidirectional LSTM in Figure 10. The LSTM accepted  $W/2$  image slices in sequence, but the RNN's input was only one image slice, and thus it would use the same slice as its input ten times. Therefore, the output of the RNN for each image slice would be ten vectors with a length of 2048, representing the encoding of just one image slice.

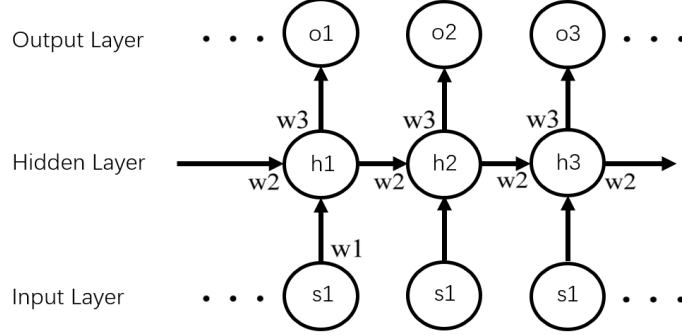


Figure 11: RNN in Decoder

Then all of the 10 vectors would pass to the subsequent softmax layer sequentially to generate the final outputs. The softmax layer here was raised by Boltzmann in 1868, and it is currently a typical output layer in the classification problem. It received a feature and outputs the possibility that it belongs to each class [48]. In this study, the input size of softmax was 2048, and the output size was 296, which was because there were 296 labels that could be predicted (including notes, key signatures, etc.). After that, the class with the highest probability was selected as the output. Consequently, each image slice would be represented by ten predictions after the processing of softmax since it had ten vectors/features. And the reason for 10 was that this study assumed each chord contained at most ten notes, which meant every slice had at most ten musical symbols. If the current slice had less than ten notes, it would remain blank for the remaining positions, and if there was no musical symbol, it would add a 'noNote' symbol like the figure below.



Figure 12: Sample output of an image slice

After the above processing, it is clear that there were  $5W$  predictions (10 predictions for each image slice, including blank, and there are  $W/2$  slices;  $W$  means the width of the input image). This was much larger than the number of labels in the ground truth, so the traditional cross entropy could not be used as the loss function here. Hence, Connectionist Temporal Classification (CTC) loss raised by Kim et al. was chosen in this study [49]. It aimed to maximise the possibility of converting a sequence of image slices to its correct label through a particular alignment method consisting of two main rules. The first rule was that repeated predictions were merged into one. Since the width of each image slice was only 2 pixels, it is normal to have several image slices representing the same musical symbol. Another rule was to delete all 'noNote' symbols because the only function to introduce them was to help the identification of repeated symbols above. The formula of CTC is shown below, where  $P$  represents possibility,  $y$  is the target sequence,  $z$  is the alignment,  $x$  is the sequence of vertical image slices, and  $\theta$  is the model parameters.

$$\max_{\theta} P(y | x; \theta) = \max_{\theta} \sum_z P(y, z | x; \theta) \quad (3)$$

In addition, the above loss was separated for each image slice, and thus the final loss should consider the likelihood of all slices. Plus, the maximisation problem should be changed to a minimisation one. Therefore, the final loss function is shown in the formula below, where  $s$  represents



the number of slices in this image.

$$\mathcal{L}_{Final} = 1 - \frac{1}{s} \sum_i^s \mathcal{L}_{Pitch} \quad (4)$$

### 3.3 Chord Recognition

Chord recognition part completed points 6 to 9 from 'Description of the Work'. The overall design was divided into the following four components: dataset creation, data pre-processing, model Design and algorithm improvement. The subsequent four subsections would explain each part's methodology and implementation in detail.

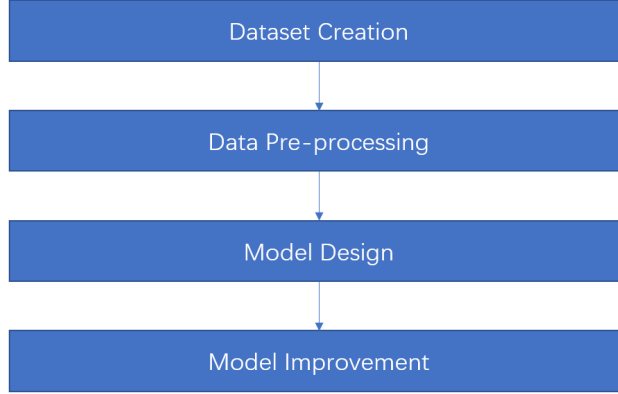


Figure 13: Design Pipeline for Chord Recognition

#### 3.3.1 Dataset Creation

As mentioned in the note recognition part, the output of a chord contained ten notes, including blank (see Figure 12). Plus, each line of the scores shared the same key signature, and this symbol also affected the prediction of the chord (see Figure 5). Hence, the training data for chord recognition combined the ten notes and the key signature as its features and was translated into csv format. The labels for these features were chords, which could not be obtained from the output. They were instead decoded from the musicxml file mentioned in Note Recognition's Image and Label Generation part. After getting these chords, combining them with notes was a serious problem. Because not every note recognised from the score belongs to a chord, it failed to automatically combine them in sequence. Therefore, we chose to manually filter out the non-chord notes from the dataset and then connect the remaining parts with chords to get 3040 samples for this dataset. The figure shows five pieces of the samples in the dataset. The first column represents the label, and the other columns represent training features (blank is treated as a null value).

	chord	keySignature	note1	note2	note3	note4	note5	note6	note7	note8	note9	note10
0	C	CM	note-C4	note-E4	note-G4	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	F	CM	note-F4	note-A4	note-C5	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	Bb	CM	note-Bb3	note-D4	note-F4	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	Eb	CM	note-Eb4	note-G4	note-Bb4	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	Ab	CM	note-Ab3	note-Cb4	note-Eb4	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Figure 14: Initial dataset for chord recognition

However, the above dataset was impossible to be used to train a machine learning model directly since the type of the features was the string. Thus, the first thing need to do was to numericalize the features to floats or integers. Because the chords here were labels, their difference from each other was negligible in this study. So LabelEncoder from the sklearn package was chosen

to convert chords to integers starting from zero in the order in which they appeared [50]. The key signatures could also be processed in the above way. Although they were training features, their differences from each other were not important either. However, an algorithm was designed as a numerical method for note encoding because the difference between notes was highly related to the accuracy of the machine learning model (see Figure 2, the pitch difference between notes on the score should determine the similarity). The algorithm’s logic is shown below: First, replacing the note with its numbered notation according to the dictionary below (based on C major’s rule, but this is also suitable for other keys since the relative difference between each note is not changed). The dictionary will be written as D in the subsequent paper.

Note	C	D	E	F	G	A	B
Numbered Notation	1	2	3	4	5	6	7

Table 2: Note Encoding Dictionary D

After that, the number in the note was considered as a coefficient in this encoding. Flat symbol b and sharp symbol # were constants that decreased or increased the result by 0.49. Thus, the equations to encode the notes are shown below. For instance, ‘note-Ab3’ is encoded as ‘6 + (3 − 1) \* 7 − 0.49’, which is 19.51. And the reason for choosing 0.49 for the flat and sharp symbol was to facilitate the inverse encoding to distinguish notes like ‘note-C#1’ and ‘note-Db1’(encoded as 1.49 and 1.51, respectively). These two notes were technically the same in music, but it is confusing when translating the number back to note. Thus, 0.49 aimed to introduce a slight difference that was not enough to affect subsequent recognition.

$$\begin{aligned}
\text{note-Xn} &= D(X) + (n - 1) * 7 \\
\text{note-Xbn} &= D(X) + (n - 1) * 7 - 0.49 \\
\text{note-X\#n} &= D(X) + (n - 1) * 7 + 0.49
\end{aligned} \tag{5}$$

Another problem in the dataset was that there were null values, and those data could not be treated as input in the machine learning model either. Hence, they need to be removed or replaced by other values beforehand. The missing values could be visualised using the bar chart below, and it is clear that there were nearly no values from note 5 to note 10, so it was reasonable to delete those five columns straightly. Although note 6 also had only 10 data, later experiments found that keeping this feature improved accuracy.

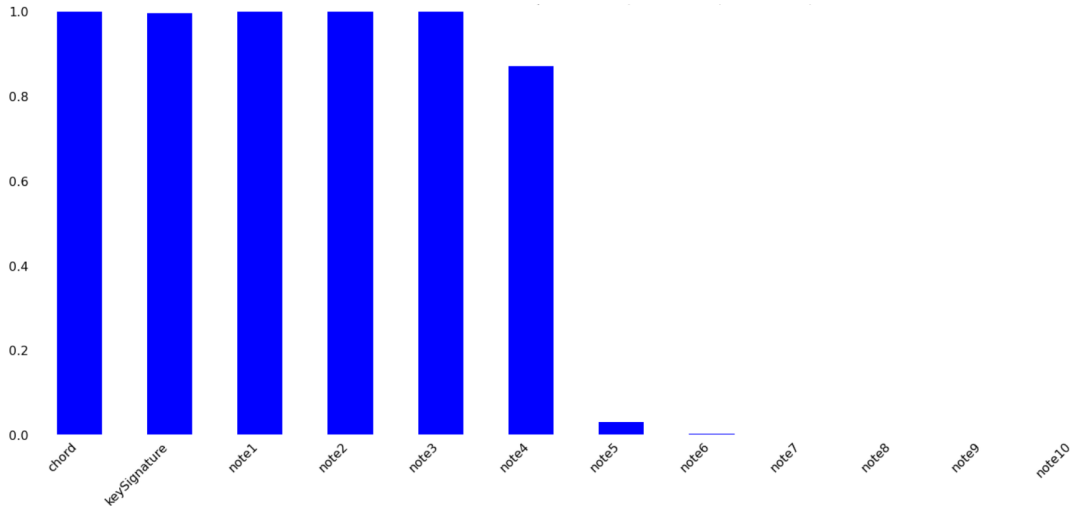


Figure 15: Missing Values

The other missing values were processed by KNN imputation algorithm. This algorithm aimed to find the K nearest neighbours of each data with missing values and used their values to predict the missing ones [21, 51]. Other imputation algorithms, including using mean value, median value and mode, have been tried in this experiment. However, KNN imputation was found to have the overall best effect in terms of both accuracy and meaningfulness. Since the missing values in this

dataset were all notes, the KNN algorithm was able to predict the most likely imputation based on the key of the chord and the remaining notes, which added a sense of musicality to the imputed values. After the above processing, a valid dataset was created for model training, and a sample of the current dataset is shown below.

	chord	keySignature	note1	note2	note3	note4	note5	note6
0	12.0	3.0	22.00	24.00	26.00	29.333333	30.000000	32.000000
1	27.0	3.0	25.00	27.00	29.00	31.006667	29.170000	32.000000
2	7.0	3.0	20.51	23.00	25.00	27.170000	29.333333	29.666667
3	22.0	3.0	23.51	26.00	27.51	30.006667	29.836667	32.000000
4	2.0	3.0	19.51	21.51	23.51	27.006667	26.666667	29.666667

Figure 16: Legal dataset for chord recognition

### 3.3.2 Data Pre-processing

The above part has already created a legal dataset, but if this data was used to train the model directly, it would lead to a poor accuracy. Therefore, this section aimed to improve the dataset to make it more robust. For example, the range in this dataset was distributed too widely under the above encoding method. Thus, the model would be dominated by a few features with large values and ignore others. Zero-mean normalization was implemented to solve this problem. This method used the mean and standard deviation of the input data to scale down them [52], and the formula is shown below, where  $z$  represents the output data,  $x$  represents the input data,  $\mu$  represents the mean, and  $s$  represents the standard deviation.

$$z = (x - \mu) / s \quad (6)$$

Furthermore, since the number of chords was over 400, achieving high accuracy on this small training set was difficult. Some methods are adopted in this study to reduce the number of chords. For instance, Jared’s musical approach called shell voicings aimed to introduce a simple system for playing the chords of any jazz tune using just eight easy chord shapes [53]. In order to achieve this goal, Jared analysed performances from many professional guitarists and summarised some substitution rules in playing chords. Part of the above rules has been applied by this study to simplify chords and to reduce the number of categories. Firstly, the bottom letter of a slash chord was ignored, such as the chord ‘G7/B’ could be simplified as ‘G7’. Then, extensions were translated to 7 or omitted directly. For example, the chord ‘G9’ could be played as ‘G7’ or even ‘G’. Lastly, alterations were deleted, like the chord ‘Galt7’ could be treated as ‘G’. Through these three rules of shell voicings, a large number of the initial chords were merged together, and the total number of chords was reduced to nearly 100.

However, some chords were still unlikely to be detected after the musical method because they contained too few samples compared to the other classes. In particular, the classes artificially augmented by merging, such as the chord ‘C7’, had the most samples at about 250. Hence, this study filtered out some low-frequency samples, and the threshold of this selection was set to 10% of the amount of ‘C7’, which was 25. After that, only 36 chords with high frequency and also obeyed the rules from shell voicings were retained to predict. In addition, this method removed 544 samples from the dataset, which meant there were 3040 remaining data.

Although the above approach filtered out the low frequencies, the classes with the least and the most samples were still ten times different, which meant this dataset was highly unbalanced. The most obvious solution was to increase samples for minority classes, and that was the reason super-sampling was adopted in this study. In detail, ADASYN (Adaptive synthetic sampling) raised by He et al. has been chosen as the main algorithm to implement [54]. This algorithm contained five steps: First, calculated the difference  $D$  between the minority and majority classes to get the number of samples that would be synthesised. Second, KNN was used to detect neighbours for each sample from the minority class [21], and then calculated the percentage  $p$  of the neighbours that did not belong to this class. Third, normalized the percentages to get weight for each sample

using the formula below, where  $p_i$  represents the  $i_{th}$  percentage, and  $w_i$  represents its weight.

$$w_i = \frac{p_i}{\sum_{i=1}^n p_i} \quad (7)$$

Fourth, multiplied the weight  $w_i$  and difference D to get the amount of new data that would be generated next to sample  $x_i$ . Last, used the following formula to create new data for minority classes, where  $s$  represents the new data,  $x_i$  represents  $i_{th}$  sample from this class,  $x_{zi}$  represents a random minority class sample, and  $\lambda \in (0, 1)$ .

$$s = x_i + (x_{zi} - x_i) \times \lambda \quad (8)$$

It is clear that the ADASYN method not only generates mimic data according to each sample in minority classes to make the dataset balanced, but also ensures that different classes are evenly distributed in the sample space after processing.

### 3.3.3 Model Design

As stated in Related Work, machine learning’s model design and selection should refer to the specific problems and be tested practically based on the dataset. Therefore, three machine learning models were created and compared for the processed dataset above in this study, including an SVM [25], a Random Forest [23] and an ANN. To achieve a fair comparison, nested K fold was used on the training set to find the best hyper-parameters for every model here. K fold cross-validation divides the dataset into K subsets for K times of validation. Each time it will be trained on K-1 of these folds and evaluated on the remaining fold. Nested K fold is an extension of K fold cross-validation, which not only divides the dataset into K subsets for validation but also divides the K-1 training folds into K subsets again for parameters tuning [55].

The specific parameter tuning method is based on the idea of grid search. That means the nested k fold has tested all parameters within the input range and returns a parameter set with the best overall validation accuracy. In detail, SVM tested different kernel functions including ‘poly’ and ‘gauss’, and the penalty parameter C from 1 to 100 (Its kernel function maps SVM to a non-linear space using polynomials or Gaussian functions, and penalty parameter C determines the tolerance for error). As a result, the Gaussian kernel function and penalty parameter C of 65 were obtained as the optimal parameter set by nested K fold; Then, Random Forest tested the maximum depth of the trees from 3 to 20, minimum leaf number of each tree from 2 to 20, and maximum features number that each tree used from 2 to 30. And nested K fold gave the best parameters combination at 14, 8 and 22 respectively; Last, ANN tested the batch size of 8, 16 and 32, the number of hidden layers from 3 to 5, and the number of neurons in each hidden layer from 50 to 200. And its final model was a fully connected network with three hidden layers of 80, 200, and 50 neurons, and a typical softmax output layer for the classification problem. Plus, this ANN’s batch size was set to 16, and the activation function was relu for default.

The models were then created with the above optimal parameters sets found in the nested k fold, and their accuracy in the test set was compared. The test set performance would be used together with k fold’s validation accuracy to evaluate the performance of each model. In summary, the ANN model had the best overall performance, so the below ‘Model Improvement’ part was based on the output of ANN. But the detail of the results would be shown in the subsequent ‘Experiments Results and Evaluation’ section.

### 3.3.4 Model Improvement

To further improve the model’s accuracy, a mechanism called ‘ambiguity’ was designed in this study. This mechanism aimed to help the model when it is difficult to distinguish the chord. Hence, the first thing needed to do was define indistinguishable and when this mechanism should be used. Because the above ANN model used softmax as its output layer, it should output an array containing likelihood that it belongs to each class. Moreover, the largest value in the likelihood array should be result of the ANN, and that value also indicated the model’s confidence. In this study, this value was used to determine whether this data is indistinguishable and should be classified as ‘ambiguity’ or not. However, determining the threshold for this value was somehow difficult since there is a trade-off between accuracy improvement and proportion of ‘ambiguity’ data. It is clear that the use of a higher threshold would change more output as ‘ambiguity’, and

no user would like to see most of the output as uncertain. So a set of thresholds starting from 0.5 and ending at 0.95 were tested (increase 0.05 each time). When ANN’s output confidence is below this threshold, the result is considered to be an ‘ambiguity’.

After that, we would like to consider how to handle those ‘ambiguity’ values to increase the accuracy. In this study, the model would output multiple chords instead of just one when it met the ‘ambiguity’ data. And when one of these output chords is the correct answer, it will be considered as a true prediction. Specifically, it would not only output the maximum value from the softmax layer, but also find three alternative possibilities. And the method used to find backup values was an idea that combined musical knowledge and probability. As mentioned in the Dataset Creation section, key signatures have remained in this dataset as a feature, but actually these signatures were critical in determining chords because there were only a limited number of chords that were legal to appear under a certain key except a few tonicized chords [56]. Consequently, the model in this study would scan ‘ambiguity’ data’s likelihood array from softmax layer according to its key signature. The three backup chords would be frequently used chords under that key, and their rank would be determined by their probability in the likelihood array.

The table below shows the performance comparison between different thresholds. It is clear that the proportion of ‘ambiguity’ starts from nearly 5% to 40%, and the accuracy improvement starts from 1.5% to 9.5%. It seems reasonable to choose any number from them as the final threshold since the accuracy improvement and proportion of ‘ambiguity’ are correlated. Therefore, the threshold selection has been implemented in the final App, and the users could choose from no ‘ambiguity’ to 40% ‘ambiguity’ depending on how much ‘ambiguity’ data they could accept.

Threshold	0.50	0.55	0.60	0.65	0.70	0.75	0.80	0.85	0.90	0.95
Accuracy Improvement (%)	1.56	1.73	3.22	3.71	4.21	5.36	6.35	7.01	8.17	9.49
Proportion of ‘Ambiguity’ (%)	4.44	6.25	8.72	10.53	12.50	15.30	19.24	22.37	28.45	38.98

Table 3: Performance of The ‘Ambiguity’ Mechanism Under Different Thresholds

Furthermore, the above table shows that processing indistinguishable data significantly improves the accuracy of the model, which means that simple data may dominate the model training in this study. Thus, the loss function was changed to a particular function called the focal loss to help solve this problem. This function was an improvement of cross entropy loss function and was proposed by Kaiming He in 2017, it aimed to achieve this goal by reducing the proportion of samples with high confidence in the total loss and increasing the weight of indistinguishable data [57]. The formula of the focal loss is shown below, where  $p_t$  represents the confidence from the likelihood array for sample  $t$ ,  $\gamma$  represents the weight ( $\gamma > 1$ ), and  $\log(p_t)$  represents an ordinary cross entropy loss function. Plus,  $\gamma$  was set to 2 in this experiment, which was tested manually, and with the focal loss, ANN’s accuracy improved at about 3% compared to using cross entropy.

$$FL(p_t) = -(1 - p_t)^\gamma \log(p_t) \quad (9)$$

### 3.4 User Interface

A simple user interface was designed and packaged into a local Windows executable for visualizing the results of this study. This application was developed using a Python package called Flet [58], which is a Python version of the well-known open-source UI development framework called Flutter from Google [59]. Flet was chosen because it inherits the advantage of Flutter and supports hot reload for efficient UI construction. Moreover, combining it with the recognition algorithms was more convenient since it was written in Python, the main language used in this study.

The final application consists of four pages, each containing multiple functions. The first page is a welcome page with the name of the App - ‘RecChord’, along with three buttons to start the recognition process, change the settings, and view instructions.

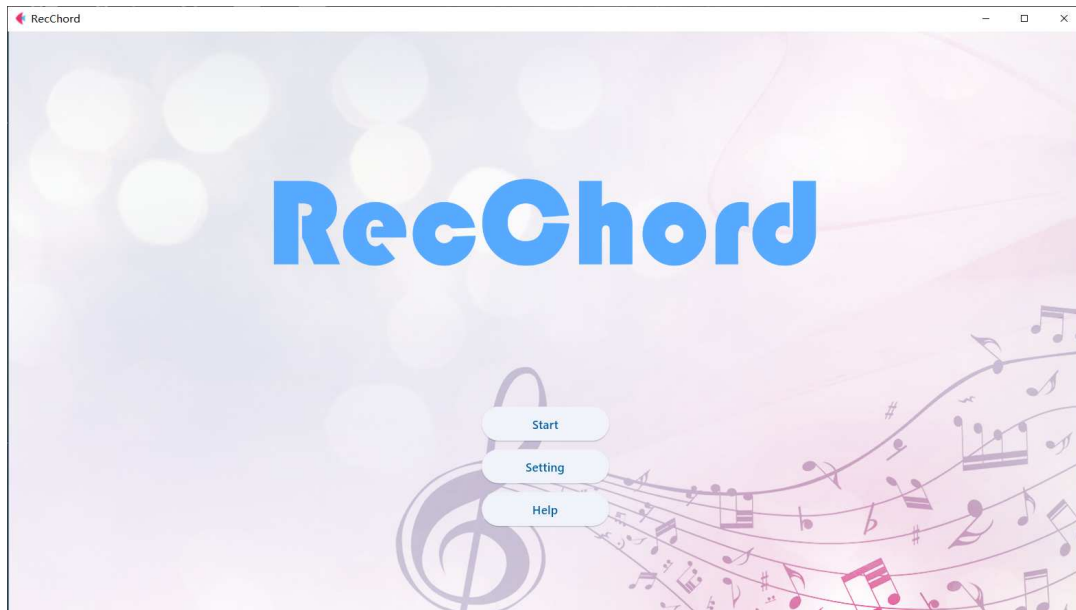


Figure 17: Welcome Page

The Help button provides instructions on how to use the App, which includes eight steps and four interactive buttons on the page. The first figure shows the initial help page, and the second figure shows the interface after pressing the 'sample input' button. The 'shell voicing' button serves as a hyperlink that takes users to the source document. The 'chord' button displays the legal chords in the program, and its effect is similar to pressing the 'sample input' button, so it is not shown here. Finally, the 'back to menu' button allows users to return to the welcome page, as indicated literally.

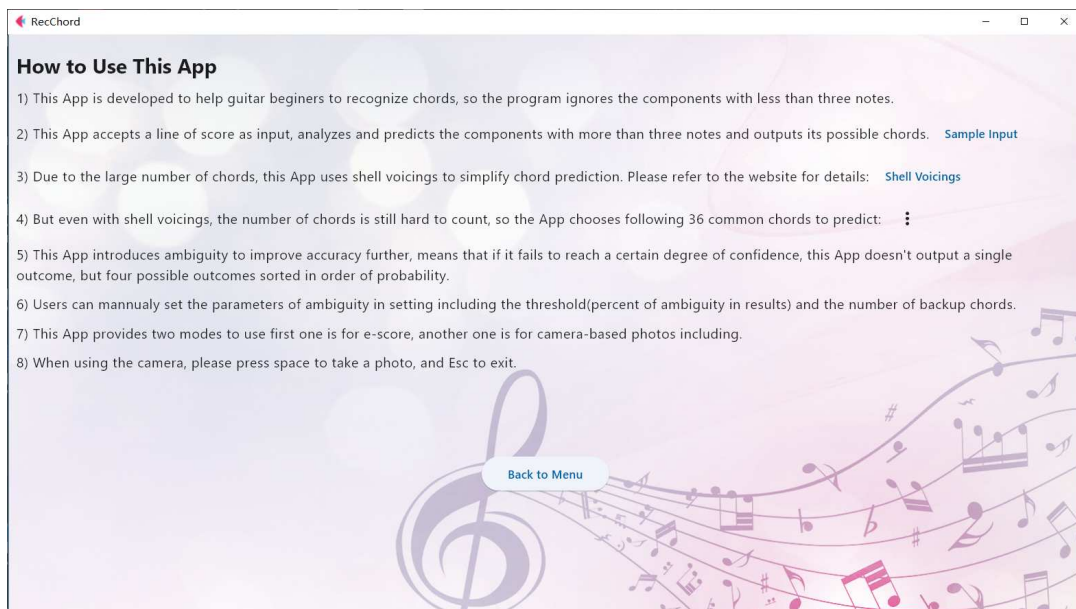


Figure 18: Help Page



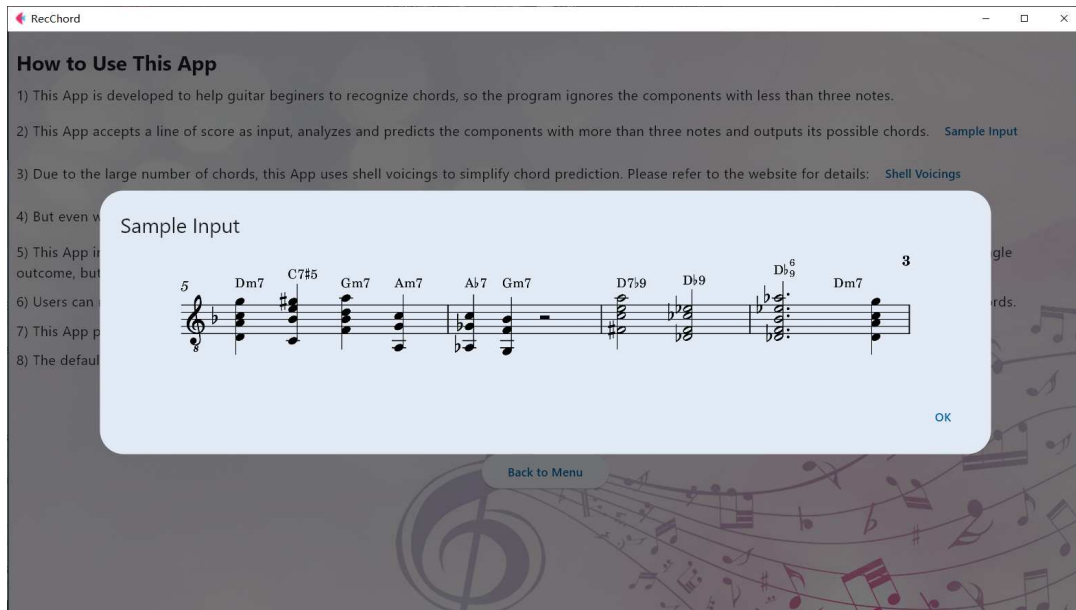


Figure 19: Help Page After Pressing Sample Input Button

The setting page in this App is used to adjust parameters used in recognition and customize the look of the UI. For instance, users can change the threshold for the 'ambiguity' mechanism (as introduced in section 3.3.4), and choose the type of input, either e-score or photo, which determines whether the program adopts the pre-processing pipeline and augmentation model described in sections 3.2.3 and 3.2.4. Additionally, there is a button to switch between dark and light themes. Since this App uses the light theme by default, the sample of the dark theme is also shown below.

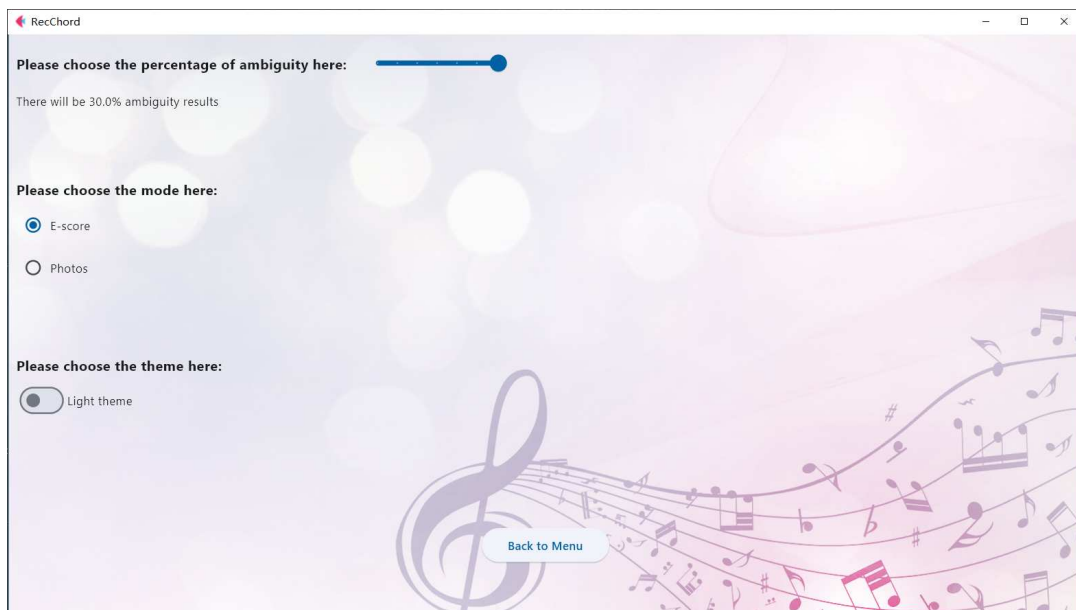


Figure 20: Setting Page

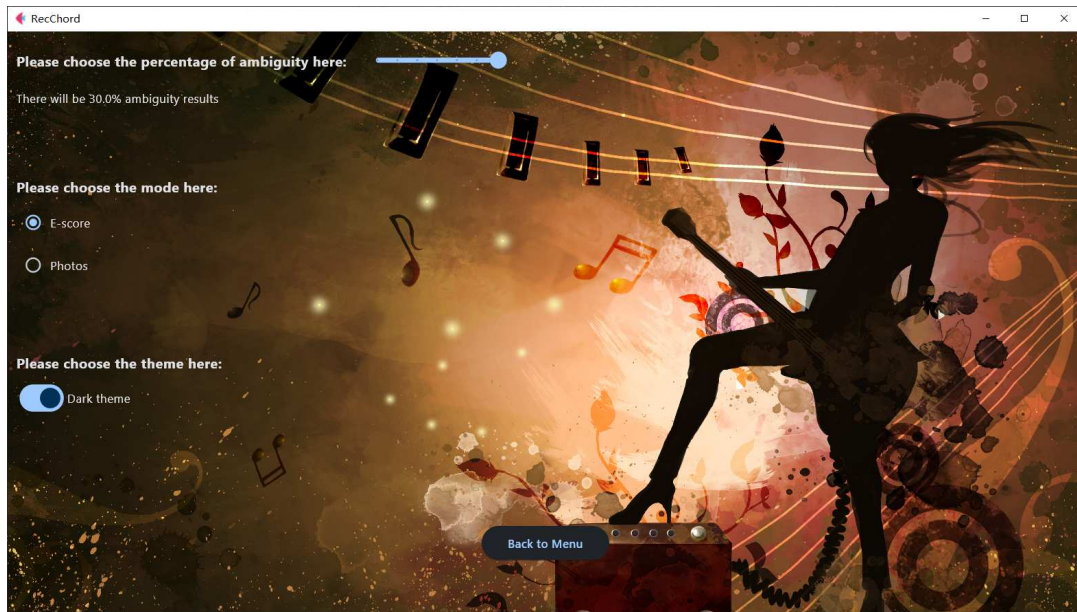


Figure 21: Dark Theme

The start button in the App takes users to the central part of the app, where they can upload pre-stored pictures or take photos with their camera to use as input. Once the user finishes selecting the image, it is displayed in the app. Then the user can press the 'recognition' button to recognize the input image. The figure below shows the output of an ordinary guitar chord score, where chords 1-4 are regular predictions. Chord 5 is the output using the 'ambiguity' mechanism, with the first component being the predicted chord and the other three being the backup chords. In addition, the displayed output here is partial, and users can view the remaining part by scrolling the screen with their mouse.

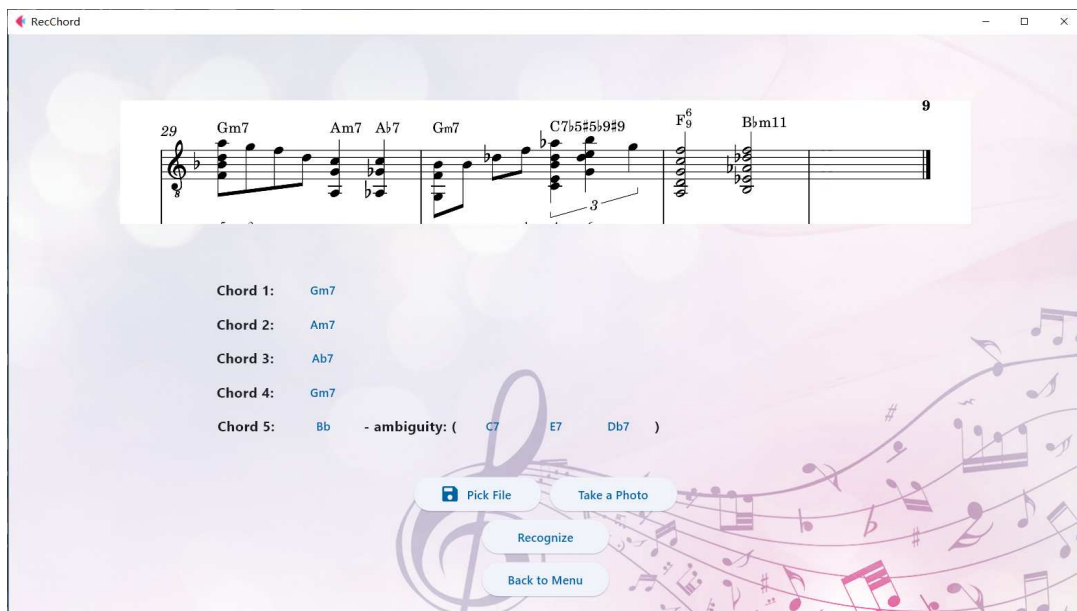


Figure 22: Output of Guitar Chord

Furthermore, every chord, including the backup is interactive. Users can click to check a detailed description for them. The sample is shown below, which includes the current chord's finger alternatives and a button to play the sound of the chord for an immersive experience.



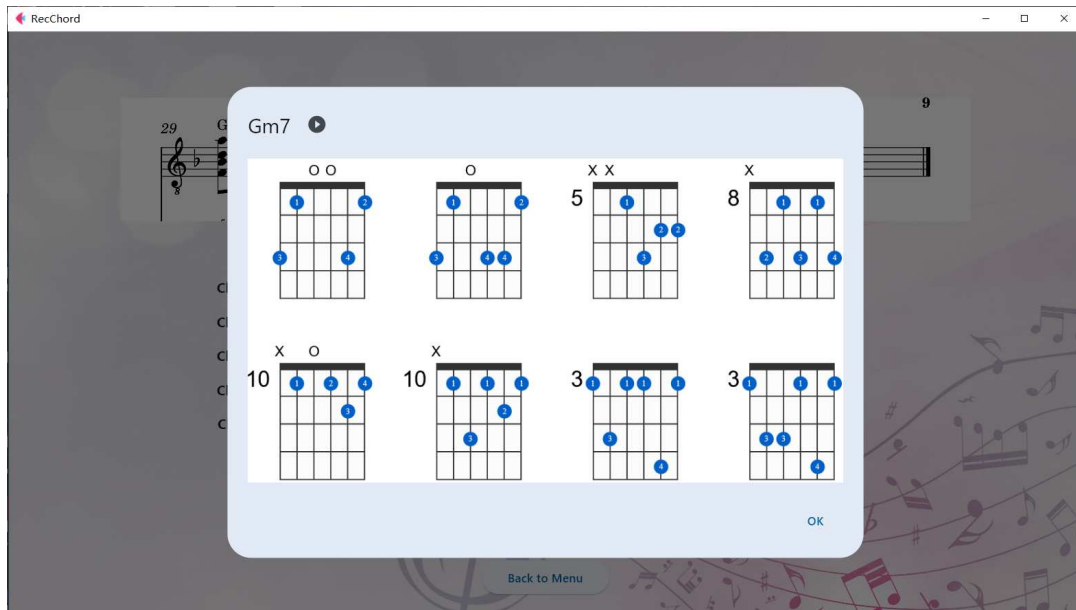


Figure 23: Chord Detail

In order to ensure a user-friendly experience, the App displays appropriate messages or prompts to guide users in handling special cases. For example, the figure below shows samples of messages displayed when no input image or a blank image is provided:

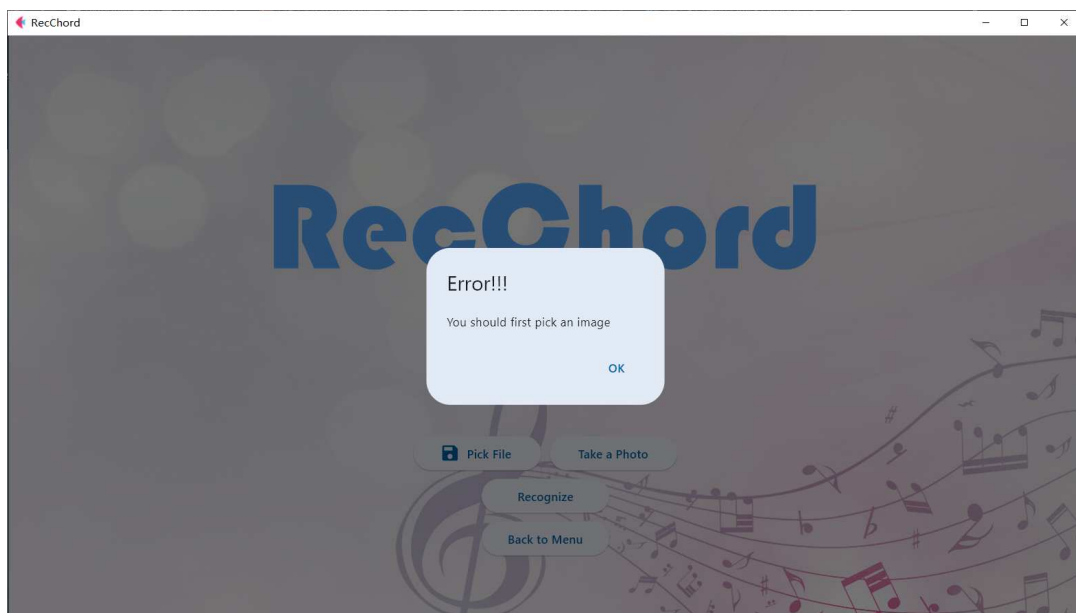


Figure 24: No Input



Figure 25: Blank Image Input

## 4 Experiments Results and Evaluation

### 4.1 Environments

This study relies on several essential tools for development and compilation. Firstly, Python 3.9.12 is chosen as the primary programming language due to its mature deep learning system and other functional modules. Secondly, OpenCV 4.5.4 is utilized for handling image processing and augmentation instructions in the note recognition part. Thirdly, PyTorch 1.13.0 is employed for building the deep learning model since it has high flexibility to change the model's detail. Lastly, scikit-learn 1.1.3, TensorFlow 2.10.0, and Keras 2.10.0 are selected for building the machine learning model, as they provide built-in models that are user-friendly. Furthermore, all the above tools and other necessary packages are already included in the App, and it is important to ensure there are no environmental conflicts before running it.

### 4.2 Note Recognition Result

Some experiment details of the note recognition are shown below. Firstly, The general dataset was divided into a training set and validation set in an 8:2 ratio (the test set is the guitar dataset), which means there were initially 5069 training images, 1267 validation images and 584 test images (each image represents a line of the score, see Figure 5). Secondly, the batch size was set to 12 because each image would not only be divided into many small slices but also predicted ten times in the decoder, and thus it was too time-consuming to have a larger batch size. However, the small batch size made the loss of both the training set and the validation set unstable and unable to evaluate the model's performance accurately. So a small trick was implemented that each epoch contained 96 batches and then calculated the loss together (which means 1152 images in each epoch), and this trick showed the trend of the loss more accurately [60]. Third, the learning rate was set to a small value of  $1e-4$  initially since the batch size was relatively small, and a small learning rate could help to reduce the oscillation of training and validation loss. Last, transform learning was implemented in this study, and a pre-trained encoder using 125,000 images was used to reduce the time consuming [15]. However, their task took into account the recognition of much musical content that was irrelevant in this study, such as rhythm, style and so on. Thus, the loss was simplified to only focus on musical symbols, and the decoder was trained to gain better performance on just note recognition by ignoring other components. Meanwhile, every training image in this study would use data augmentation to mimic real-world photos and increase the model's generalizability.

Due to the GPU and RAM's performance limitations, only 60 epochs have been trained successfully, which means 69,120 images were trained. This number is much larger than the total amount of the training set, but it is legal since data augmentation has been added randomly before the training process. The training loss and validation loss are indicated in the image below. It is clear that both losses decreased rapidly to a low level and stayed in a general good value. Although these two values oscillated in a small range, the overall trend has declined. perhaps the instability of the loss values is due to the introduction of the random data augmentation module, which leads to a certain degree of variation in the training content of each epoch, so that even if the model parameters are optimised, the loss of the next epoch may be increased a little bit.

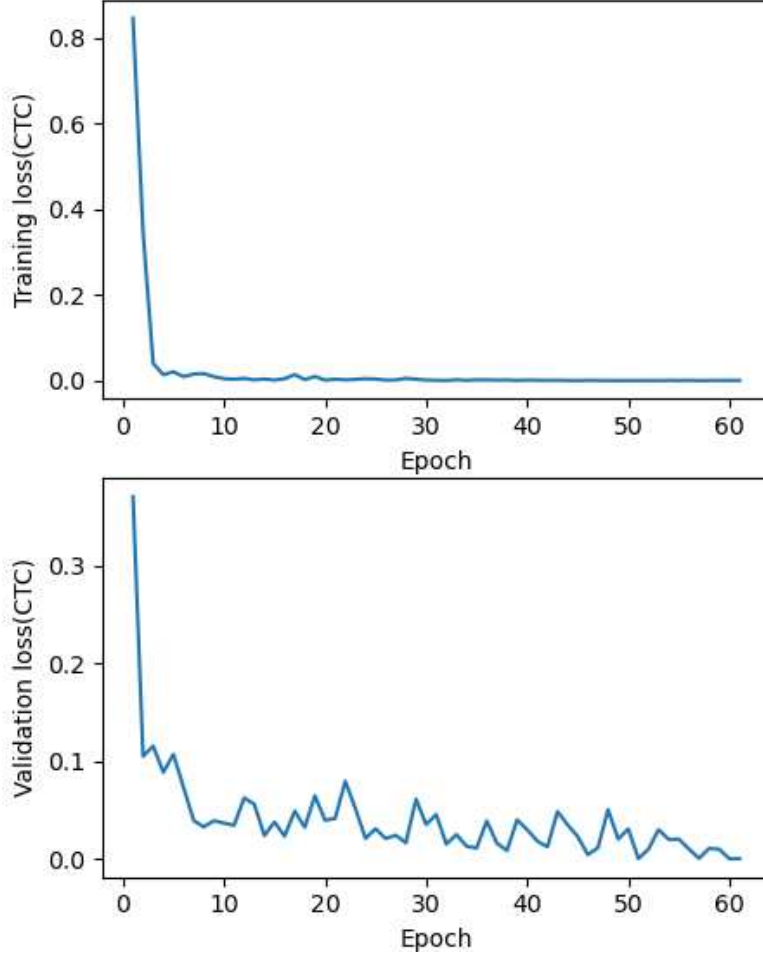


Figure 26: Training and Validation Loss

As mentioned above, the performance of the model was evaluated using images from the guitar chord dataset. The SER (Symbol Error Rate) was used as the evaluation metric, as it was deemed unsuitable to calculate classification accuracy due to the possibility of the model missing some notes or predicting more notes in a specific image. SER is calculated by using the edit distance divided by the length of the sequence(L), and edit distance means the number of insertions(I), deletions(D) and substitutions(S) needed to edit the prediction string to ground truth [19]. That means SER can be calculated using this equation:

$$SER = (I + D + S)/L \quad (10)$$

The table below shows the SER comparison between the pre-trained and trained models after 60 epochs on clean images and real photos (obtained by photographing the clean images in the test set, see the right image in Figure 7). It is clear that compared to the pre-trained model without data augmentation, the trained model has much better performance (over 15% performance increase) in the real photos. Meanwhile, it does not lose much performance in clean images, with only about


a 2% decrease. However, the error rate of the real photos is still twice that of the clean images, which may be because the training and validation dataset’s data augmentation cannot perfectly simulate the real photos. But with more than 69,000 images used for training, photographing them would be an impossible task, so data augmentation was indeed the best method we could come up with for this experiment.

Model	Clean Images	Real Photos
pre-trained	8.29%	36.85%
trained	10.34%	20.40%

Table 4: SER Result

Meanwhile, it is unable to calculate the classification accuracy also means that this study cannot check the precision of each label. While this categorical method is not suitable for the RNN model either since the prediction may be affected by each other in the sequence. Thus, evaluating the precision of each label in isolation may not be meaningful in this context. As a result, we manually compared the prediction from the 584 images with the ground truth and found that the model performed particularly well on most single musical symbols and chords with 3-4 notes. However, it faced challenges in accurately recognizing hard chords with 6-7 notes, as well as sharp and flat symbols. For instance, the chord shown here comprises of 6 notes and 2 flat symbols. Due to the high density of notes, note-F4 is positioned on the left side of the chord, which could potentially result in the chord being mistaken as a single note along with a chord consisting of five notes. The same issue may arise with the missing flat symbol on the left, as it is too far from the chord. We supposed that because the two datasets in this study consist primarily of real songs, such complex chords are hard to find in them. Consequently, the model lacks the ability to handle hard chords properly with extremely high density.

**C13(sus4b9)**



Ground Truth: note-C4 note-F4 note-G4 note-Bb4 note-Db5 note-A5

Prediction: note-F4 + note-C4 note-G4 note-B4 note-Db5 note-A5

Figure 27: Hard Chord

### 4.3 Chord Recognition Result

As stated in section 3.3.1, there were 3040 chords samples in the datasets before the super-sampling algorithm ADASYN generated new samples to balance the dataset [54]. Those original samples were first divided into a training set and a test set in an 8:2 ratio, which means there were initially 2432 training data, and 608 test data. After that, only the data of the training set was handled by ADASYN, and it output a new training set with 2992 samples.

K fold cross validation was adopted on the training set to compare the performance of the three machine learning models. In this study, K was set to 5, and thus their final validation accuracy depended on the average of the five folds. The result of the K fold cross validation is shown in the table below.

Model	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
SVM	70.18%	66.83%	67.44%	70.47%	72.76%	69.54%
Random Forest	66.35%	68.97%	66.30%	67.86%	71.15%	68.13%
ANN	73.75%	67.37%	68.34%	72.34%	74.66%	71.29%

Table 5: Cross Validation Accuracy

The test set performance was also evaluated to compare the three models. It is clear that ANN has the best validation accuracy and test set accuracy compared to SVM and random forest.

Model	SVM	Random Forest	ANN
Test Accuracy	69.38%	71.23%	73.44%

Table 6: Test Set Accuracy

Additionally, normalized confusion matrices were also adopted to further evaluate the performance of the three models, as the accuracy differences mentioned above may not be robust. The confusion matrix is a commonly used evaluation method in multi-class classification, as it indicates which class each input is predicted to belong to, and helps identify classes that are likely to be confused with each other [61]. In the confusion matrix, the columns represent the true labels, while the rows represent the predicted labels. Thus, the values on the diagonal represent the probability of correct classification, and any other values represent the probability of incorrect classification. Since the testset had varying sample sizes for each class, the matrix was normalized to fairly reflect the classification detail.

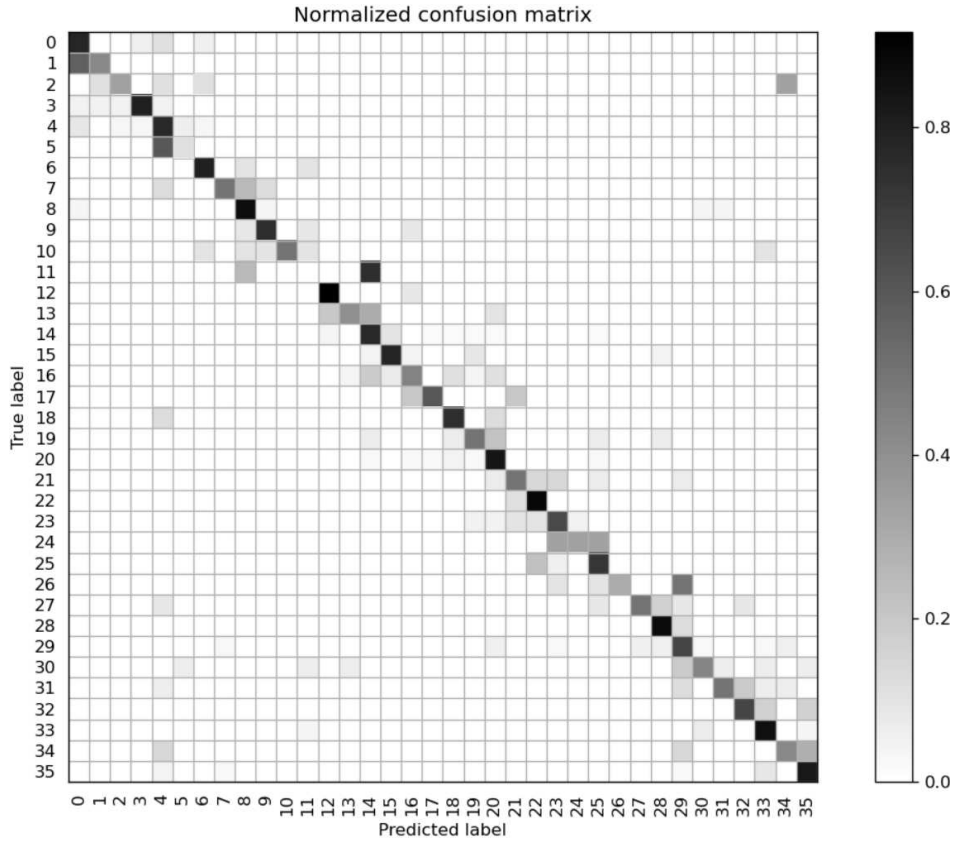


Figure 28: Confusion Matrix for SVM

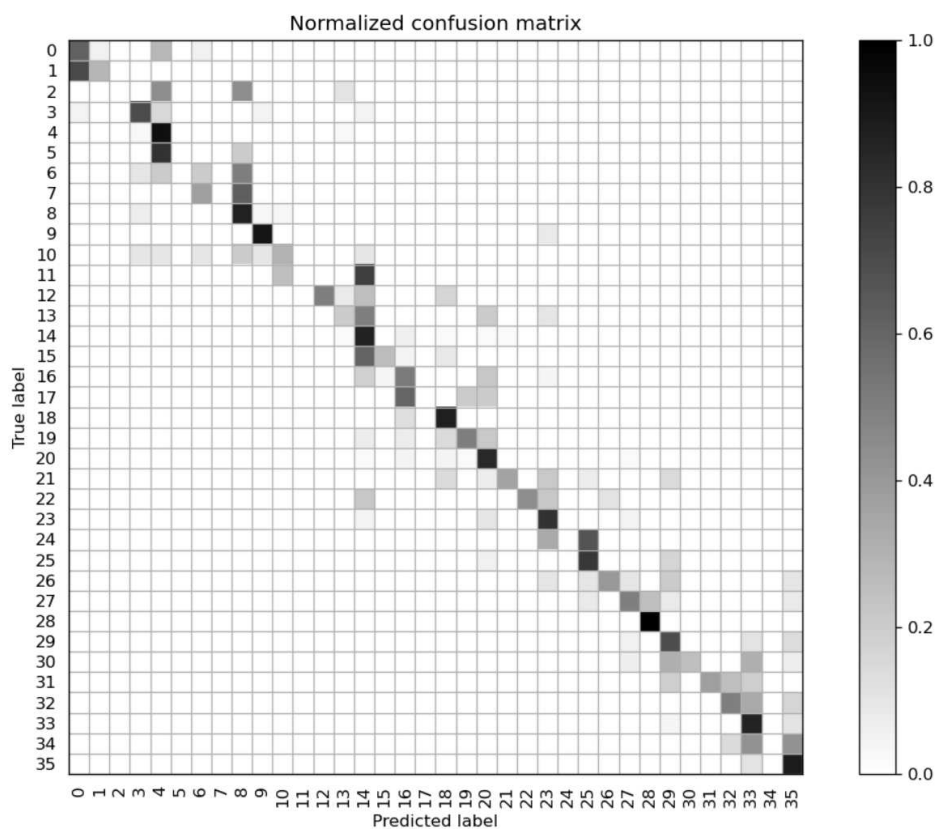


Figure 29: Confusion Matrix for Random Forest

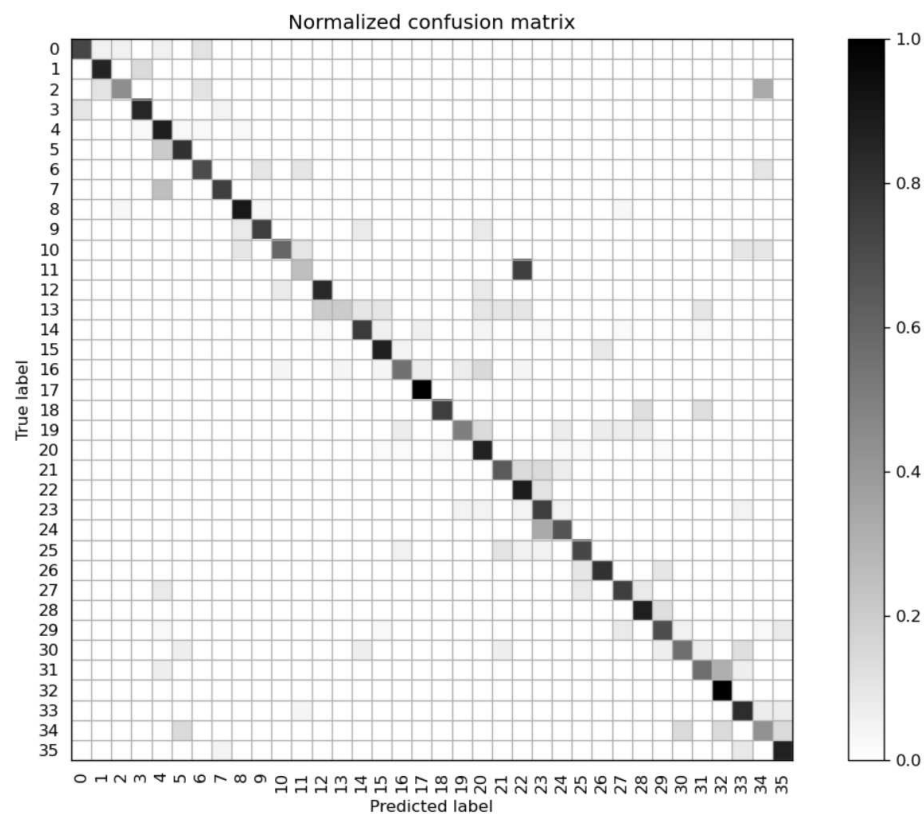


Figure 30: Confusion Matrix for ANN

According to the above confusion matrices, it is evident that the predictions made by SVM and random forest are less reliable. They both have multiple classes with 0% classification correctness, indicating that SVM and random forest do not gain the ability to predict these classes at all. Moreover, the predicted labels in the random forest are heavily concentrated on class 3,7,14, etc., which suggests that the model achieves a seemingly high test accuracy by simply predicting most inputs as the classes with more samples.

In contrast, most of these categories have decent precision in ANN, and thus it has been assumed as the final model to use in the application with the deep learning model. However, there is also an exception in ANN's confusion matrix, where class 11 ('Eb7' chord) is most misrecognized as class 22 ('Eb' chord). The musical explanation could be that the two chords are so close, with just one note difference. Another possible explanation from the point of machine learning is that the samples of 'Eb7' chord in the dataset were limited, so most of this chord was synthesised through super-sampling, which might not accurately represent the true features of 'Eb7' chord. Therefore, the model may struggle to distinguish 'Eb7' chord from other similar, larger classes, leading to misclassification.

While if the 'ambiguity' mechanism and focal loss are adopted, the previous problem can be alleviated (see section 3.3.4). Table 7 shows the final accuracy of the model under different "ambiguity" threshold settings.

Threshold	0.50	0.55	0.60	0.65	0.70	0.75	0.80	0.85	0.90	0.95
Accuracy (%)	76.90	77.72	78.38	79.37	80.36	80.86	81.35	82.51	83.99	84.82
Proportion of 'Ambiguity' (%)	4.44	6.25	8.72	10.53	12.50	15.30	19.24	22.37	28.45	38.98

Table 7: Final Test Set Accuracy

When the threshold of 'ambiguity' is set to 0.95, which means 40% 'ambiguity' data is accepted, the accuracy is at about 85%. This value is also the final result of this study since the dataset of chord recognition is directly obtained from note recognition. In addition, the final confusion matrix of the test set is shown in the figure below. It is clear that the classification of the 'Eb7' chord is improved, and the model generally performs well in each class.

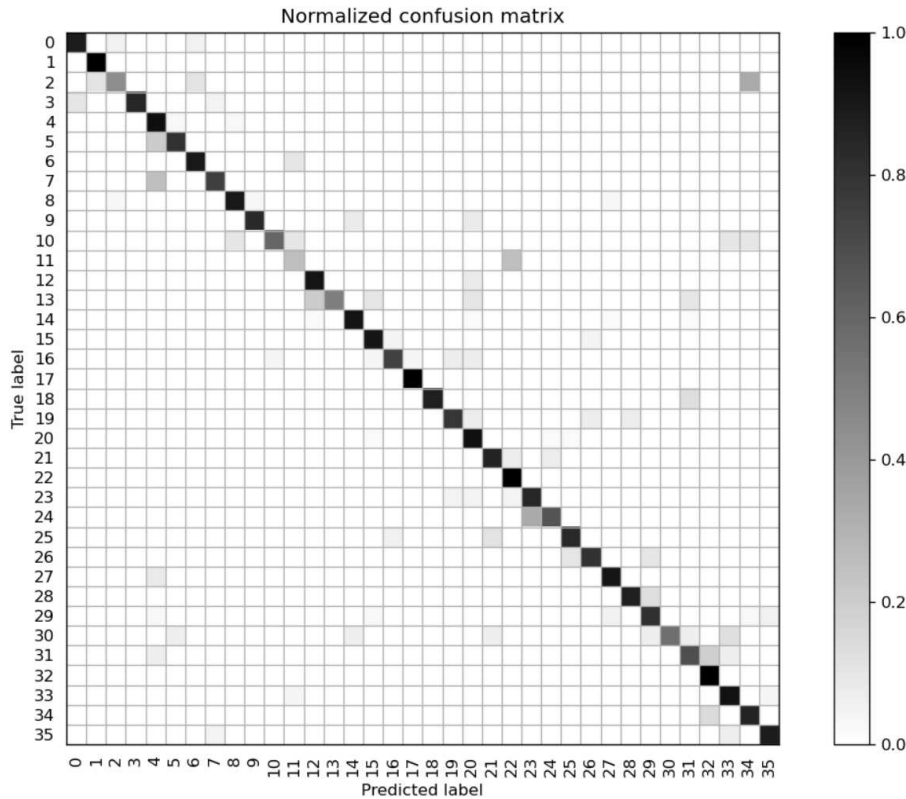


Figure 31: Final Confusion Matrix

#### 4.4 Unittest for User Interface

In order to ensure the reliability of the software, Pytest is used here to evaluate it. Pytest is a commonly used frame for writing unittest, integration test and etc. in Python [62]. In this study, the APIs provided by the two models mentioned above have been tested using the test data shown in the tables below.

Test ID	Test Description	Input	Expected Result	Pass/Fail	Actual Result
1.1	regular e-score image	a line of score with e-score mode	'clef-G2 + keySignature-FM + note-F3 note-Bb3 note-D4 note-A4 '	Pass	'clef-G2 + keySignature-FM + note-F3 note-Bb3 note-D4 note-A4 '
1.2	non exist image	a fake file with e-score mode	AttributeError	Pass	AttributeError (Handled in the App part)
1.3	empty image	a blank image with e-score mode	None	Pass	None
1.4	error image	a blank image with two lines of score and e-score mode	'clef-G2 + keySignature-FM + note-F3 note-Bb3 note-D4 note-A4 ', 'clef-G2 + keySignature-FM + note-F3 note-Bb3 note-D4 note-A4 '	Fail	'clef-G2 + keySignature-FM + note-E5 note-E5 note-E5' (Remind users the correct input in App)
1.5	regular camera-based image	a line of camera image with e-score mode	'clef-G2 + keySignature-FM + note-F3 note-Bb3 note-D4 note-A4 '	Pass	'clef-G2 + keySignature-FM + note-F3 note-Bb3 note-D4 note-A4 '
1.6	non exist mode	a line of e-score image with error mode	'clef-G2 + keySignature-FM + note-F3 note-Bb3 note-D4 note-A4 '	Pass	'clef-G2 + keySignature-FM + note-F3 note-Bb3 note-D4 note-A4 ' (Handled as e-score mode)

Table 8: Test for Note Recognition



Test ID	Test Description	Input	Expected Result	Pass/Fail	Actual Result
2.1	non exist note file	a fake file with thresh-old=30	None	Pass	None
2.2	non exist threshold for 'ambiguity'	a real file with thresh-old=666	None	Fail	UnboundLocalError: local variable referenced before assignment. (Fixed using try...catch)
2.3	regular input	a real file with thresh-old=30	['Gm7']	Pass	['Gm7']
2.4	ambiguity data in the file	a file contains ambiguity data with thresh-old=30	['Bb,C7,E7,Db7']	Pass	['Bb,C7,E7,Db7']
2.5	no data in the file	an empty file with thresh-old=30	symbol '&' indicates no chord	Pass	symbol '&' indicates no chord
2.6	multiple data in the file	a real file with thresh-old=30	['Gm7','Gm7']	Pass	['Gm7','Gm7']
2.7	error data in the file	a file with one column missing and thresh-old=30	None	Fail	KeyError: "'file' not found in axis (Fixed using try...catch)

Table 9: Test for Chord Recognition

## 5 Summary and Reflection

### 5.1 Project Management

The project was mainly managed by myself, and I had weekly meetings with my supervisor Prof. Tony Pridmore to update my current progress and discuss the next step. The time management of the entire project basically followed the Gantt chart raised in the proposal. Except for the App development part, it actually used less than three weeks since the chord recognition part cost more time. There were two reasons for this situation: Firstly, Improving a machine learning model was harder than expected. Although each part of the machine learning model, such as pre-processing and parameter tuning, was relatively straightforward and performed well individually, the combination of these methods resulted in poorer performance. Hence, significant time was invested in designing and testing different approaches. Secondly, there was an unforeseen setback when I had to return to my home country due to a visa issue, which caused a delay of several weeks at the beginning of the second term. But fortunately, the project was successfully completed despite these challenges.

The figure below shows the Gantt chart used in this study. The meanings of the letters in the chart are explained as follows:

a. Proposal writing and literature review

- b. Ethic checklist
- c. Dataset generating
- d. Familiar with basic music knowledge
- e. Familiar with deep learning methods
- f. Implement notes recognition
- g. Write Interim report
- h. Implement chords recognition
- i. Design App and familiar with a UI frame
- j. Coursework catch-up
- k. Develop App
- l. Write final dissertation and demo
- m. Prepare demonstration

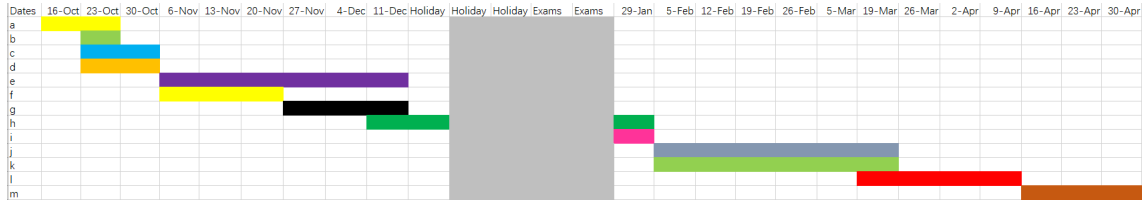


Figure 32: Gantt Chart

## 5.2 Contribution

In conclusion, this project involved many research areas, including musical skills, deep learning, machine learning and software development. I have spent tremendous time on the literature review to explore essential techniques. Through comparing and contrasting various methods, I have acquired diverse knowledge in order to design a novel pipeline for the guitar chord recognition problem.

In the note recognition part, I have created two datasets for guitar chord scores and general scores separately. And I have also integrated different researchers' work to raise a deep learning model capable of recognizing notes from camera-based polyphonic scores with special pre-processing methods and data augmentation skills.

In the chord recognition part, I created an encoding pipeline to transform the note predictions into a usable dataset. I also conducted comparisons among different machine learning models and pre-processing techniques for the musical input, and devised algorithms to enhance the overall performance.

As for the software development aspect of the project, I have designed and tested a user-friendly App that visually showcases the above approaches and provides detailed information about the input chords. Although musical technology was a relatively new area for me at the beginning, this project piqued my interest in it.

## 5.3 Reflection and Future Work

This project met the initial requirements, but some improvements are still needed to make it more accurate and fast.

Firstly, the model could not handle chords with extremely high note density (typically containing 5-6 notes along with several flat, sharp symbols) in the note recognition part of the project. This limitation was primarily due to the special training samples were not enough, and most of the chords in the dataset contained only 3-4 notes (see Figure 27). Even some chords were removed due to the limited samples, which reduced the diversity of the model. Furthermore, as stated in section 4.2, data augmentation could not fully represent camera input, and thus model's accuracy to camera input was relatively poor. To address those data issues, it is crucial to collect a more comprehensive set of training samples that cover sufficient camera-based complex chords.

Secondly, the ANN model used in the chord recognition part could be used as the output layer of the decoder in note recognition. Combining the two models to create an end-to-end system can significantly increase the accuracy and prediction speed. Although the algorithms can be easily

transferred, the subsequent problems after the transfer are hard to solve. Because the current deep learning model was used to identify notes, not chords, its labels contained only single musical symbols. Its polyphonic detection capability was achieved by the RNN in the decoder's output layer to recurrently check each image slice for single notes.

To generate a single chord output, the ANN from chord recognition can be added after the RNN layer. However, this method is not feasible because the current dataset comes from real music, and thus it is impossible to contain only chords in those data. As mentioned in section 3.3.1, a manual check was necessary to filter out the non-chord components before using the ANN model. Another possible solution is to replace the RNN layer with the above ANN. However, this would require adding numerous new labels not only for chords but also for multi-notes, such as the subsequent component next to the 'Bm11' chord in the figure below. This approach is impractical as the existing model already requires significant time and GPU resources to achieve good performance, and increasing the number of new classes may make it difficult to train. Additionally, if we choose to delete the labels for single notes and multi-notes from the model to speed up the process, it may impede the recognition task, as these components resemble chords more than blank lines.



Figure 33: Sample Input with Multi-notes

For the software engineering part, a server can be set up to convert the local App to an online one. Because the deep learning models are pretty extensive and too many packages are related, the size of the App is over 3GB. But unfortunately, the time is limited as stated in the 'Project Management, since the development of two algorithms and an unexpected visa accident cost too much time.

## 5.4 Reply to LSEPI

In this study, all researches are about musical technology, so there is no human or biological data. At the same time, the product will not be used commercially or militarily in the future. Hence, this project does not refer to any serious ethical problem. The possible issue is the data collection that all musical files in this experiment come from the Muse forum. But as stated in the 'Design Methodology and Implementation' section, this forum is a public, open-source forum where users can freely share their work produced by MuseScore [31]. Meanwhile, the API used to collect the data also had an MIT license for private use in GitHub [32], and thus data collection in this study is legal. Another possible issue is the camera use, but this project aims to capture only the photo of scores. Plus, the camera photos will be destroyed after exiting the App, and thus it is also legal in this study. The problem of data leakage is also irrelevant since this experiment did not use any data containing privacy, and even all of them can be found on the Internet initially. The intellectual property of the novel algorithms and designs in this study belongs to me, but it is fully open-source in GitHub now. Everyone can use the link below to utilize the source code and final App: <https://github.com/Celery233333/FYP>.

## References

- [1] E. Brown, “The notation and performance of new music,” *The Musical Quarterly*, vol. 72, no. 2, pp. 180–201, 1986.
- [2] D. Bainbridge and T. Bell, “The challenge of optical music recognition,” *Computers and the Humanities*, vol. 35, no. 2, pp. 95–121, 2001.
- [3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.
- [4] A. N. Bhagoji, D. Cullina, C. Sitawarin, and P. Mittal, “Enhancing robustness of machine learning systems via data transformations,” in *2018 52nd Annual Conference on Information Sciences and Systems (CISS)*, pp. 1–5, IEEE, 2018.
- [5] J. F. Canny, “Canny edge detector,” 1986.
- [6] F. Rossant and I. Bloch, “Robust and adaptive omr system including fuzzy modeling, fusion of musical rules, and possible error detection,” *EURASIP Journal on Advances in Signal Processing*, vol. 2007, pp. 1–25, 2006.
- [7] A. Rebelo, I. Fujinaga, F. Paszkiewicz, A. R. Marcal, C. Guedes, and J. S. Cardoso, “Optical music recognition: state-of-the-art and open issues,” *International Journal of Multimedia Information Retrieval*, vol. 1, no. 3, pp. 173–190, 2012.
- [8] C. Raphael and J. Wang, “New approaches to optical music recognition,” in *ISMIR*, pp. 305–310, 2011.
- [9] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [10] A. Pacha and J. Calvo-Zaragoza, “Optical music recognition in mensural notation with region-based convolutional neural networks,” in *ISMIR*, pp. 240–247, 2018.
- [11] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
- [12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *nature*, vol. 323, no. 6088, pp. 533–536, 1986.
- [13] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [14] L. Chen, R. Jin, S. Zhang, S. Lee, Z. Chen, and D. Crandall, “A hybrid hmm-rnn model for optical music recognition,” in *Extended abstracts for the Late-Breaking Demo Session of the 17th International Society for Music Information Retrieval Conference*, 2016.
- [15] S. Edirisooriya, H.-W. Dong, J. McAuley, and T. Berg-Kirkpatrick, “An empirical evaluation of end-to-end polyphonic optical music recognition,” *arXiv preprint arXiv:2108.01769*, 2021.
- [16] MuseScore, “English forum.” <https://musescore.org/zh-hans/forum1>. Accessed December 4, 2022.
- [17] W. Ng and X. T. Nguyen, “Improving deep-learning-based optical music recognition for camera-based inputs,” in *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 118–121, IEEE, 2022.
- [18] N. Li, “Generative adversarial network for musical notation recognition during music teaching,” *Computational Intelligence and Neuroscience*, vol. 2022, 2022.
- [19] E. S. Ristad and P. N. Yianilos, “Learning string-edit distance,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 5, pp. 522–532, 1998.
- [20] H. Ishibuchi, K. Nozaki, and H. Tanaka, “Distributed representation of fuzzy rules and its application to pattern classification,” *Fuzzy sets and systems*, vol. 52, no. 1, pp. 21–32, 1992.

- [21] T. Cover and P. Hart, “Nearest neighbor pattern classification,” *IEEE transactions on information theory*, vol. 13, no. 1, pp. 21–27, 1967.
- [22] T. Mizutani and S. Sasaki, “A linear regression analysis of musical expressions using the implication-realization model,” in *Proceedings of the 9th International Conference on Computer and Communications Management*, pp. 85–91, 2021.
- [23] L. Breiman, “Random forests,” *Machine learning*, vol. 45, pp. 5–32, 2001.
- [24] Q. Sun, D. Zhang, Y. Fan, K. Zhang, and B. Ma, “Ancient chinese zither (guqin) music recovery with support vector machine,” *Journal on Computing and Cultural Heritage (JOCCH)*, vol. 3, no. 2, pp. 1–12, 2010.
- [25] C. Cortes and V. Vapnik, “Support vector machine,” *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.
- [26] H. Wang and D. Hu, “Comparison of svm and ls-svm for regression,” in *2005 International conference on neural networks and brain*, vol. 1, pp. 279–283, IEEE, 2005.
- [27] J. Gillick, K. Tang, and R. M. Keller, “Machine learning of jazz grammars,” *Computer Music Journal*, vol. 34, no. 3, pp. 56–66, 2010.
- [28] P. Baldi and P. J. Sadowski, “Understanding dropout,” *Advances in neural information processing systems*, vol. 26, 2013.
- [29] J. Zupan, “Introduction to artificial neural network (ann) methods: what they are and how to use them,” *Acta Chimica Slovenica*, vol. 41, pp. 327–327, 1994.
- [30] P.-H. C. Chen, Y. Liu, and L. Peng, “How to develop machine learning models for healthcare,” *Nature materials*, vol. 18, no. 5, pp. 410–414, 2019.
- [31] MuseScore, “Musescore forums.” <https://musescore.org/en/forum>. Accessed April 9th, 2023.
- [32] LibreScore, “app version.” <https://github.com/LibreScore/app-librescore>. Accessed December 5th, 2022.
- [33] MuseScore, “File formats.” <https://musescore.org/en/handbook/3/file-formats#musescore-native-format>. Accessed December 5, 2022.
- [34] M. Good, “Musicxml for notation and analysis,” *The virtual score: representation, retrieval, restoration*, vol. 12, no. 113-124, p. 160, 2001.
- [35] M. Alfaro-Contreras, J. Calvo-Zaragoza, and J. M. Iñesta, “Approaching end-to-end optical music recognition for homophonic scores,” in *Iberian Conference on Pattern Recognition and Image Analysis*, pp. 147–158, Springer, 2019.
- [36] D. Bradley and G. Roth, “Adaptive thresholding using the integral image,” *Journal of graphics tools*, vol. 12, no. 2, pp. 13–21, 2007.
- [37] R. M. Haralick, S. R. Sternberg, and X. Zhuang, “Image analysis using mathematical morphology,” *IEEE transactions on pattern analysis and machine intelligence*, no. 4, pp. 532–550, 1987.
- [38] C. Tomasi and R. Manduchi, “Bilateral filtering for gray and color images,” in *Sixth international conference on computer vision (IEEE Cat. No. 98CH36271)*, pp. 839–846, IEEE, 1998.
- [39] K. Ito and K. Xiong, “Gaussian filters for nonlinear filtering problems,” *IEEE transactions on automatic control*, vol. 45, no. 5, pp. 910–927, 2000.
- [40] N. Gallagher and G. Wise, “A theoretical analysis of the properties of median filters,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 29, no. 6, pp. 1136–1141, 1981.
- [41] C. Shorten and T. M. Khoshgoftaar, “A survey on image data augmentation for deep learning,” *Journal of big data*, vol. 6, no. 1, pp. 1–48, 2019.

- [42] J. C. López-Gutiérrez, J. J. Valero-Mas, F. J. Castellanos, and J. Calvo-Zaragoza, “Data augmentation for end-to-end optical music recognition,” in *International Conference on Document Analysis and Recognition*, pp. 59–73, Springer, 2021.
- [43] Y. Le Cun and F. Fogelman-Soulié, “Modèles connexionnistes de l’apprentissage,” *Intellectica*, vol. 2, no. 1, pp. 114–143, 1987.
- [44] J. Calvo-Zaragoza and D. Rizo, “End-to-end neural optical music recognition of monophonic scores,” *Applied Sciences*, vol. 8, no. 4, p. 606, 2018.
- [45] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *International conference on machine learning*, pp. 448–456, pmlr, 2015.
- [46] A. Baró, P. Riba, J. Calvo-Zaragoza, and A. Fornés, “From optical music recognition to handwritten music recognition: A baseline,” *Pattern Recognition Letters*, vol. 123, pp. 1–8, 2019.
- [47] Z. Huang, W. Xu, and K. Yu, “Bidirectional lstm-crf models for sequence tagging,” *arXiv preprint arXiv:1508.01991*, 2015.
- [48] L. Boltzmann, “Studien uber das gleichgewicht der lebenden kraft,” *Wissenschaftliche Abhandlungen*, vol. 1, pp. 49–96, 1868.
- [49] S. Kim, T. Hori, and S. Watanabe, “Joint ctc-attention based end-to-end speech recognition using multi-task learning,” in *2017 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 4835–4839, IEEE, 2017.
- [50] sklearn.preprocessing, “Labelencoder.” <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html>. Accessed April 15th, 2023.
- [51] L. Beretta and A. Santaniello, “Nearest neighbor imputation algorithms: a critical evaluation,” *BMC medical informatics and decision making*, vol. 16, no. 3, pp. 197–208, 2016.
- [52] T. Jayalakshmi and A. Santhakumaran, “Statistical normalization and back propagation for classification,” *International Journal of Computer Theory and Engineering*, vol. 3, no. 1, pp. 1793–8201, 2011.
- [53] Jared Borkowski, “Shell voicings.” [https://s3.amazonaws.com/kajabi-storefronts-production/sites/86048/downloads/4qnFomATQKrFghiT2f6i\\_Any\\_Jazz\\_Chord\\_by\\_Jared\\_Borkowski\\_V2\\_.pdf](https://s3.amazonaws.com/kajabi-storefronts-production/sites/86048/downloads/4qnFomATQKrFghiT2f6i_Any_Jazz_Chord_by_Jared_Borkowski_V2_.pdf). Accessed April 15th, 2023.
- [54] H. He, Y. Bai, E. A. Garcia, and S. Li, “Adasyn: Adaptive synthetic sampling approach for imbalanced learning,” in *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*, pp. 1322–1328, IEEE, 2008.
- [55] A. Vabalas, E. Gowen, E. Poliakoff, and A. J. Casson, “Machine learning algorithm validation with a limited sample size,” *PLoS one*, vol. 14, no. 11, p. e0224365, 2019.
- [56] Wikipedia, “Key signature.” [https://en.wikipedia.org/wiki/Key\\_signature](https://en.wikipedia.org/wiki/Key_signature). Accessed April 16th, 2023.
- [57] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, “Focal loss for dense object detection,” in *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988, 2017.
- [58] Feodor Fitsner et al., “Flet.” <https://flet.dev/>. Accessed April 18th, 2023.
- [59] Google, “Flutter.” <https://flutter.dev>. Accessed April 21st, 2023.
- [60] B. Schmeiser, “Batch size effects in the analysis of simulation output,” *Operations Research*, vol. 30, no. 3, pp. 556–568, 1982.
- [61] J. T. Townsend, “Theoretical analysis of an alphabetic confusion matrix,” *Perception & Psychophysics*, vol. 9, pp. 40–50, 1971.
- [62] Krekel et al., “Pytest.” <https://docs.pytest.org>. Accessed April 21st, 2023.