Individual Project

University of the Arts London
Creative Computing Institute

Conductor： An exploration of applying machine learning models to hand-controlled music generation

Author:

Ziqing Xu

Supervisor:

Jasper Zheng

Introduction video link https://youtu.be/WQ1uSLL0rSo
Demonstration Video link https://youtu.be/C4xpgRHIve8
GitHub link https://github.com/ZIqinGX/MSc_Advanced_project/tree/main/4.Final_outcome

Abstract

Although there has been some research on artificial intelligence models for music generation and human-AI co-creation, real-time collaborative creation between humans and AI in the music domain remains relatively less explored. In this work, I address this gap by proposing a real-time human-AI collaborative creation system where the rhythm of AI-generated music can be controlled by humans. This research aims to explore the potential of real-time performance between humans and AI. Firstly, A system that combines hand gesture capture with LSTM-based music generation is created by iterative design. Secondly, post-processing rules is applied to the notes generated by the model to prevent note repetition. Lastly, parallel real-time framework that integrates the music generative model with gesture capture to more fluent interaction is introduced. The final outcome is a real-time AI and human co-creation music system named as *Conductor*. The result and middle process outcomes are evaluated by three objectives: 1) The generated notes can be manipulated, 2) The generated musical rhythm had variations and dynamics, 3) Completing the task of computer-generated note prediction and interacting with humans. Achievements made including but not limited to duration of notes can be manipulated by hand and system built is interactive, with limitations such as chord cannot be played, constraints of a controlled method and cause of repetitive notes not found.

There is still much work to be done in the future, and it holds considerable potential for further exploration. I believe that this innovation paves the way for more practical applications of generative models in real-time performances.

Acknowledgements

I would like to express my sincere gratitude to my advisor Jasper Zheng for his patience, responsible attitude, and invaluable advice throughout the completion of my master's project. I am also deeply thankful to Professor Rebecca Fiebrink for introducing me to the world of machine learning and for her inspiring teaching methods. I would also like to thank course leader, Phoenix Perry, for her hard work and dedication in M.Sc. Creative Computing course. The time I spent at the Creative Computing Institute has been an extraordinary highlight of my academic journey.

Contents

1.Introduction

1.1 Motivation

Generating music by computational techniques has existed for a long time. Computer algorithms have been adopted in shaping the music industry since the 1950s (Bonnici et al., 2021). The adoption of AI methods in live music performances and improvisation has gained notable interest in recent years due to the accessibility of software components, such as Deep Learning, as well as the expansion of extensive datasets (McCormack et al., 2020).

In the field of collaborative creation research, significant creative work has been done by providing live accompaniment for musicians (Kylafi, Ioannidis, 2020) and devising intelligent collaborative music controllers for performance artists (Martin, 2022). However, these artworks of music require musical knowledge or training as the basis for use. My interest in creating music pieces is limited by the lack of a music composition foundation. Therefore, in this article, my research question revolves around how to integrate machine learning models into my artistic practice to enable collaborative real-time music generation and explore ways to make such real-time generation tool more interactive and diverse so that I can use it without music background. The start of exploration is to manipulate the duration of musical notes through gestural control.

1.2 Objective

My objective is to construct a real-time system based on existing related work that uses LSTM models for note prediction. This system would enable the modulation of note duration based on human hand movements, thereby facilitating the real-time creation of music that brings together subjective human hand conditioning and deep learning models to predict compositions of human musicians, forming an action-production loop.

2.Methodology

To deeply explore the collaborative potential of humans and music generative models in real-time creation, I thoroughly reviewed the related work to grasp the current advances in the domain. Subsequently, I replicated and analyzed an LSTM-based note prediction model, highlighting its strengths and potential performance limitations.
These observed limitations and challenges during practical applications prompted my conceptualisation of a novel system design. Utilizing iterative design, I conducted three iterations, with each iteration measured against the extent to which it achieved the

following three objectives: 1. Manipulating the generated notes; 2. Ensuring the generated musical rhythm had variations and dynamics; 3. It was Completing the task of computer-generated note prediction and interacting with humans.
I employed the relevant technical tools and methods to realize concepts, iteratively refining the system through creative practice. Upon the system's completion, I undertook a series of tests to ensure it met anticipated functionalities and performance benchmarks. Finally, I conducted a comprehensive reflection on the exploratory outcome, compared the final system with related work, and analyzed the advantages and shortcomings for future developments.

3.Background

3.1 RNN

Figure 1 shows the structure of a simple neural network. X1, 2, and 3 are inputs of this network. Y1 and 2 are outputs of this network. Inputs will go into hidden layers, and by adjusting the Weight in (Win) and Weight out (Wout), the outcome will be close to the expected answer. The neural network has completed its learning.

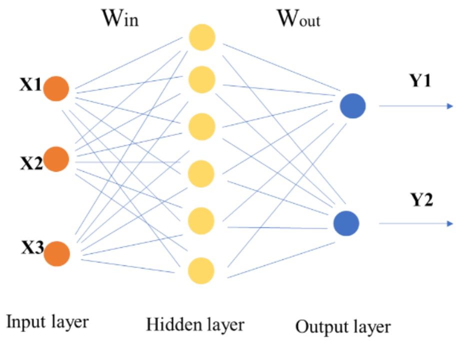


Fig1 Simple neural network

Deep learning neural networks like CNN (convolutional neural network) extend along the horizontal direction; this kind of property determines it cannot handle changes over time. To solve the problem involving the development of time or sequence, RNN(recurrent neural network) was introduced. Unlike CNN, it takes sequential or time series data as input and can deal with speech recognition (Sak, Senior and Beaufays, 2014).
One of the most commonly used RNN units involving sequential tasks is the Elman RNN (ERNN) cell (Hewamalage, Bergmeir and Bandara, 2021). It was established by Elman in 1990. This kind of RNN shares the same activation between the hidden layer and context layer in a one-to-one correspondence (Elman,1990).
If using Ws to show weight matrices between different levels in RNN, it was assumed that the chain of repeated modules in RNN shares the same Ws (*Understanding LSTM Networks,* 2015).

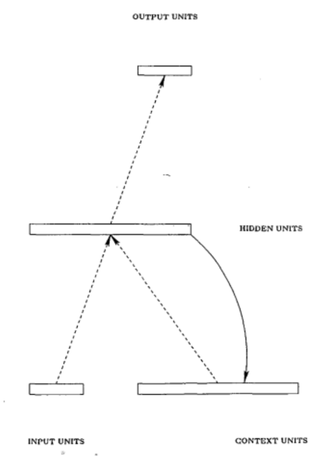


Fig2 Simple recurrent network structure

(Elman, 1990)

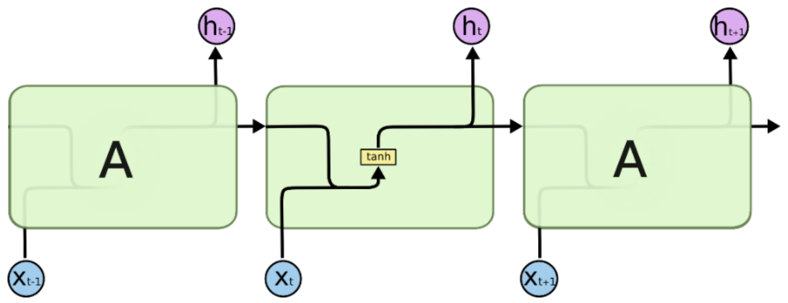


Fig3 Standard RNN. The defined formula can be found in https://colah.github.io/posts/2015-08-Understanding-LSTMs/

(*Understanding LSTM Networks,* 2015)

3.2 LSTM

LSTM is Long short-term memory. It is a type of RNN invented to avoid the vanishing gradient problem. The first paper introducing LSTM appeared in 1997, written by Sepp Hochreiter and Jürgen Schmidhuber (Sherstinsky, 2020). It can keep long-term memory compared with simple RNN.
The picture below shows what a LSTM is like.

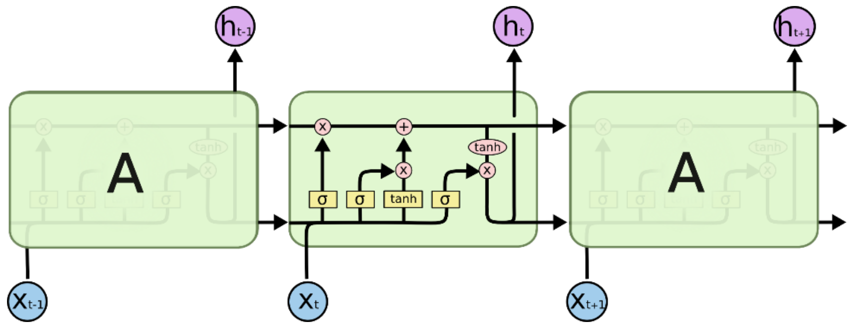


Fig4 LSTM

(*Understanding LSTM Networks,* 2015)

The typical structure of LSTM consists of an input layer, a recurrent LSTM layer, and an output layer (Sak, Senior and Beaufays, 2014). In the recurrent LSTM layer, there are forget gates to decide what to ignore.

4.Related work

4.1 Radio Baton

Radio Baton is an electronic instrument created by Max Mathews. It consists of two batons with a radio transmitting antenna, and an antenna board as base, which can track the location of 2 rods in 3D space by receiving a signal from the transmitter in wands (*Max Mathews Radio Baton controller and the conductor program demonstration*, 2010).

Each baton has x, y, and z coordinates as three information variables, so moving two batons can control six variables, making conducting flexible. Users can assign these factors to whatever attributes they want to control.

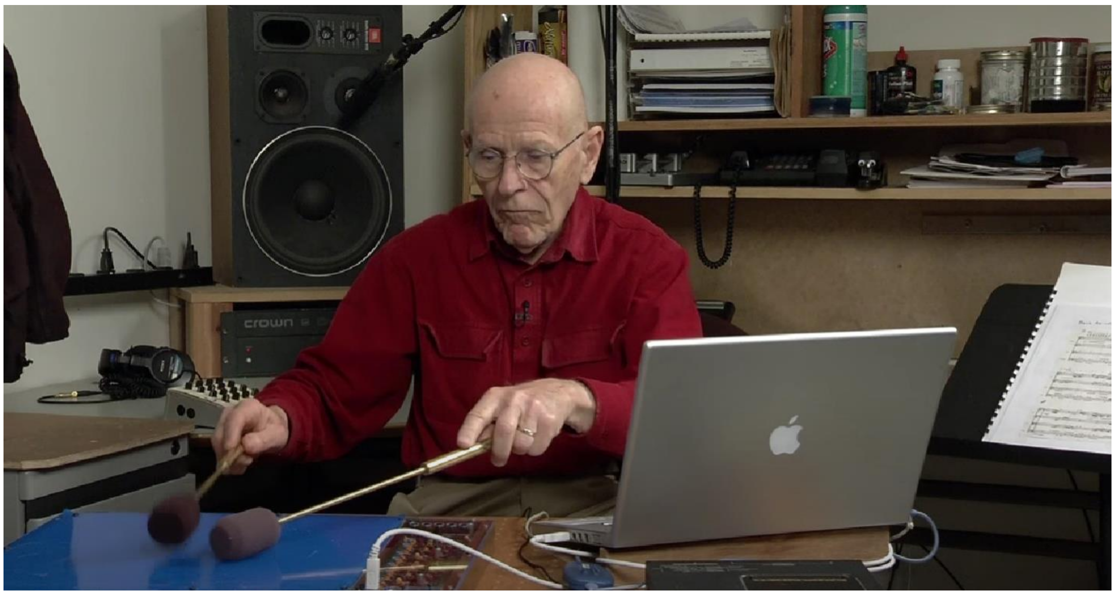


Fig5 Max Mathews Radio Baton Demonstration

(*Max Mathews Radio Baton Demonstration*, 2010)

Radio Baton can work on a conductor program and score built into the computer memory. Batons and baseboard transmit information in the conductor program to control the performance, such as tempo. The score stored in the computer will be read and played when needed.

In his video recording(2010), Dr. Max Mathews mentioned that his model of control is not the same as a violinist or pianist playing music; instead, it is a figurine resembling an orchestra conductor conducting music. Just like a conductor, the aim of this instrument is not to play every note but to guide the performance of music.

Inspired by his metaphor of the conductor, the author believed that users could be incorporated into the music generation process in the role of a conductor, controlling the expression of music through conducting gestures.

This system provides a diverse range of control over music and an effective way to focus only on conveying musical feelings instead of knowing much musical knowledge. Separating music score and music playing influences the following design of the real-time system into the music generation part (LSTM model) and conducting portion (hand gesture capture). And gives an idea of allowing the user to control the performance of the music, leaving the generation of musical scores to a machine learning model and playback to the computer program.

4.2 Interactive Musical Prediction System

In forming concepts, projects built by Dr. Charles Patrick Martin give the author much inspiration.
One piece of work is The Intelligent Musical Prediction System (IMPS). It is a physical, electrical controller equipped with eight knobs to support the direct manipulation of the synthesizer program by musicians (Martin, 2022). Fig6 (Martin and Torresen, 2019) illustrates how this system works. When musicians stop playing, IMPS can take control of performance by predicting the playing habit.
According to Dr.Martin(2022), the deep learning neural network gets information including the number of controller movements and the time interval between each movement, so rather than predicting notes, it predicts what the following behavior that performance may take by learning performance patterns from gestural data gained from user during rehearsals and live performances, this is what differentiates this system from other music generative systems.

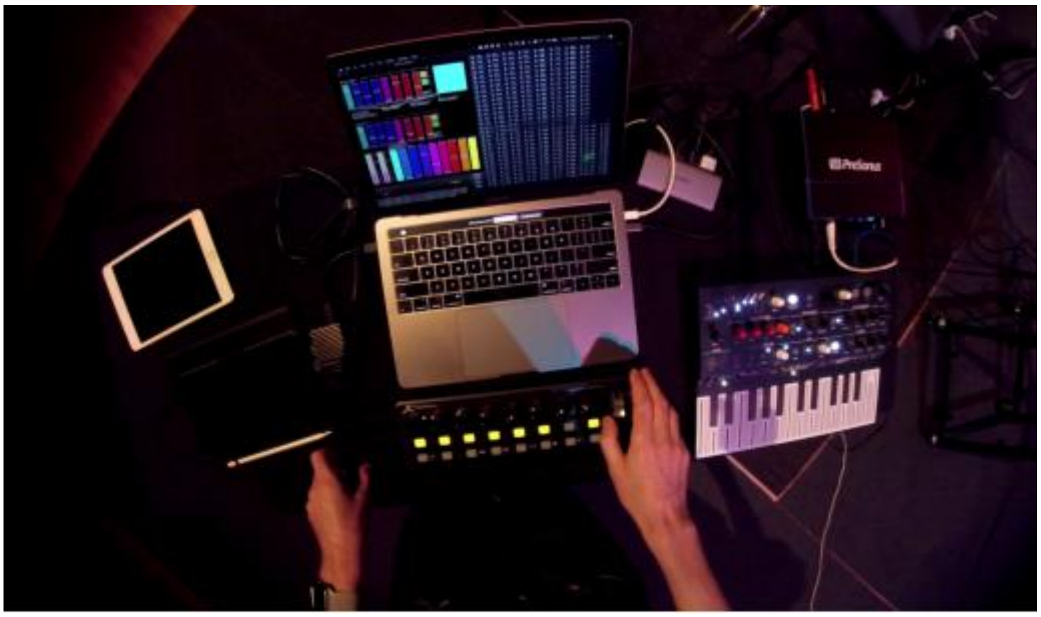


Fig2

Fig6 Performing with the generative electronic music controller

(Martin, 2022)

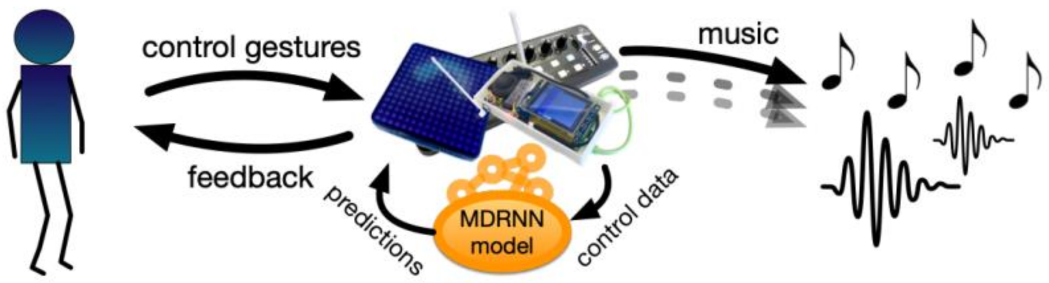


Fig7 Interactive musical prediction system

(Martin and Torresen, 2019)

For the technical part, the deep learning neural network used in this system is a mixture density recurrent neural network (MDRNN), which was introduced in another paper by Dr.Martin and Professor Torresen(2019). MDRNN was formed by employing a mixture density network (MDN) to the outcome of RNN.

The exploration done by Dr. Charles Patrick provides good examples of exploring interactive AI and human co-creation systems, which gives the author a hint of adopting machine learning technology into music generation. The difference between the author's and IMPS's system is using different training models and facing different user group. The interactive musical prediction system (IMPS) is an instrument that enhances the performance of musicians, while the hand-controlled real-time co-creative system I built is aimed at people without a music base, like me.

4.3 Kapoor: Music Generation: LSTM

Kapoor (2021) applies the LSTM model in generating music by using the first 40 notes to predict the 41st note. The author reproduced and reflected on this case.

The first thing is loading the dataset. Her dataset comes from MIDI files of classical piano music, and she chose Frédéric Chopin's compositions as a training source. Secondly, preprocessing data by extracting notes from the dataset and taking out notes which have shown less than 100 times. This step will produce a corpus that contains all valuable notes. The third step is building mapping dictionaries. It builds correlation with each unique note in the corpus and a number and prepares for encoding and decoding. The next step is dividing the corpus into shorter sequences of the same length for both features and their corresponding targets. After that, features will be resized, and targets will be transformed into one-hot code. Next, build a LSTM model and train. Finally, utilizing a well-trained neural network model, a series of notes are generated through continuously updating seed sequences, which are then transformed into their corresponding musical representation.

Based on reflection, this example has substantially accomplished generating music. It can train a model using a dataset to predict musical notes and subsequently generate a musical piece. However, there are three primary shortcomings:

1. There needs to be more functionality to facilitate further modifications to the already generated music.
2. The rhythm remains constant, devoid of any fluctuations or variations.
3. The system solely undertakes the task of predicting musical notes through computational means, needing more capacity for interactive engagement with users.

Considering these limitations informed by reflection, the author proposes a series of enhancements to address these issues. Subsequently, LSTM model will be implemented, designed to enable modifications to the music's rhythm through human hand movements, facilitating real-time musical generation and providing a more interactive and dynamic user experience.

5.First generation

5.1 Plan

First, the addressed issue is the generated music being non-adjustable and lacking rhythmic variation. The concept of the first-generation design involves controlling the duration of the generated notes through the user's hand movements, akin to a conductor, except that here, the performer was replaced by a machine-learning model. The envisioned system is shown in Figure 8:

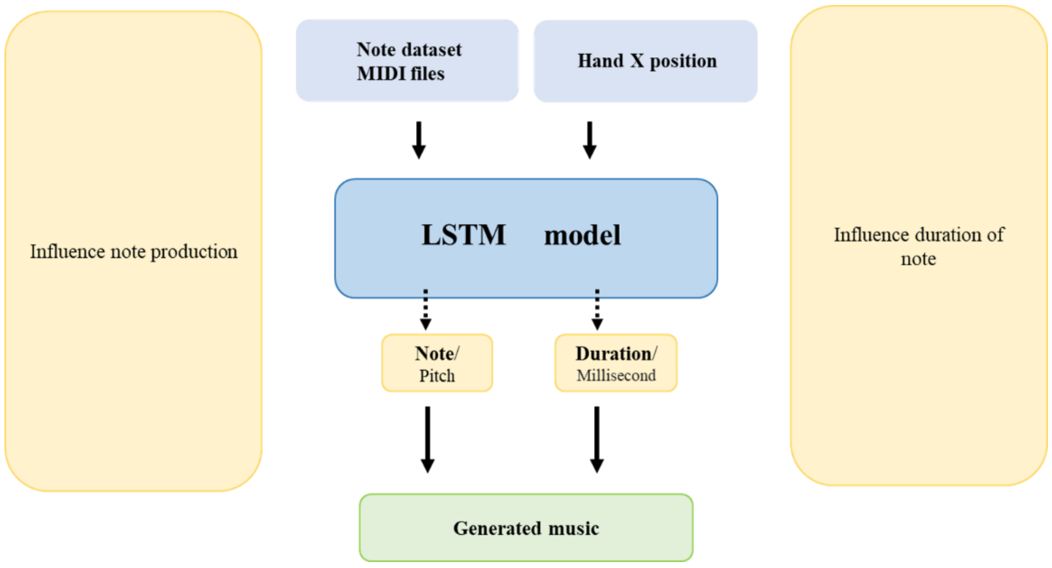


Fig8. First generation system

The position of the hand is captured by the camera, and Mediapipe analyzes the X-coordinates of one key point in the hand; the point I choose is the forefinger. Choosing one finger instead of whole five fingers simplifies the statics and makes work go faster.

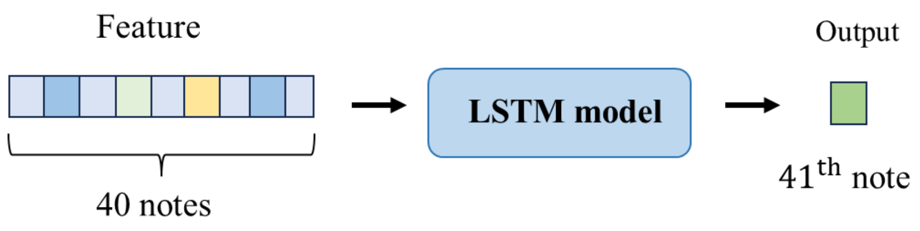


Fig9 Feature Before

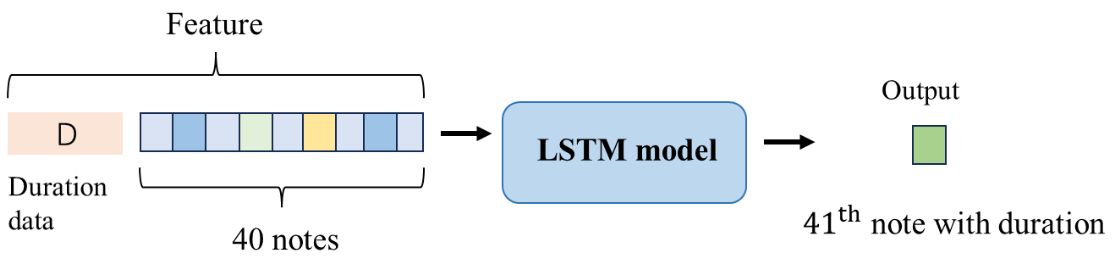


Fig10 Feature New

5.2 Evaluation

This solution might seem ideal at first glance, but it presents significant challenges in practice. Firstly, there is an issue of dataset mismatch. Specifically, the musical notes dataset I have chosen contains over fifty thousand notes. At the same time, there are only over three thousand collected data points for the hand's x-coordinate, which means forming a new training set with the same number of Duration features is difficult as there are note features. Secondly, in this proposal, the generated notes are still produced in one go without the ability to make ongoing adjustments to the music.

6. Second generation Method1： Build new dataset

6.1 Plan

After finding the problem with the first generation, one solution is getting more datasets about X-coordinates to build a mapping relationship between hand movement speed and note duration. By getting data on hand movement speed and mapping relationship, a model for producing duration can be built. The real-time result of music controlled by hand movement will generate new notes by combining the results of the two models. To clarify this idea, the figure below shows how it works.

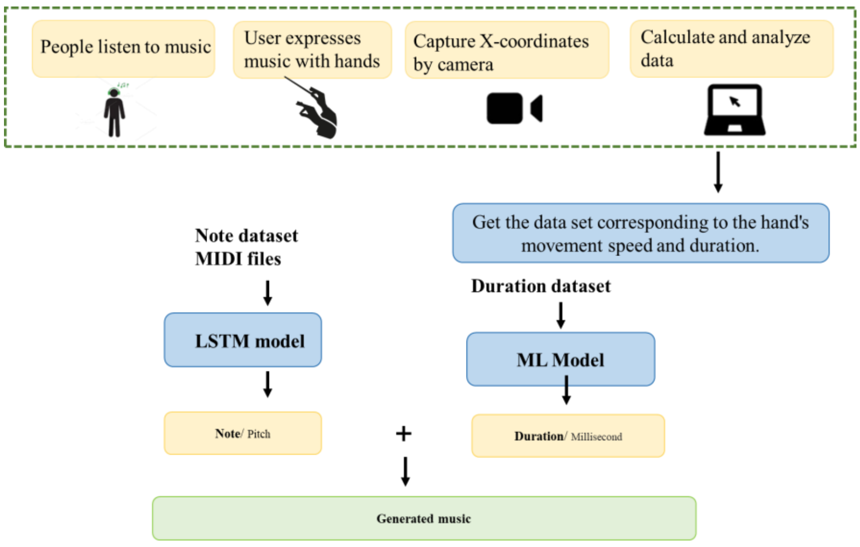

People listen to music
User expresses music with hands
Capture X-coordinates by camera
Calculate and analyze data
Get the data set corresponding to the hand's movement speed and duration.
Note dataset MIDI files
Duration dataset
LSTM model
ML Model
Note/ Pitch
+
Duration/ Millisecond
Generated music

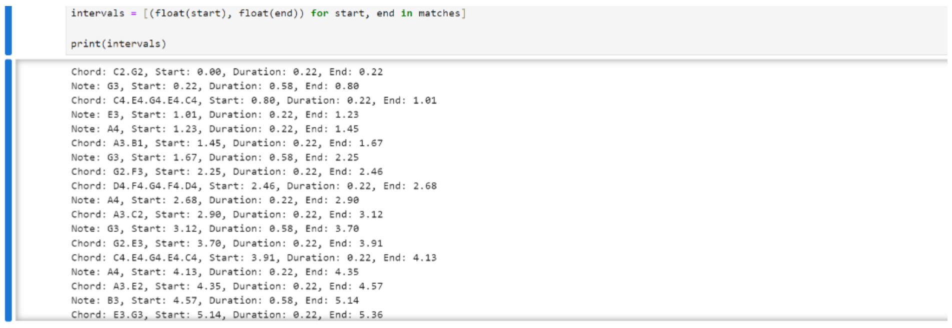
Fig11 Overview of plan

6.2 Implementations

The detailed implementation of getting more data to build a new dataset is as follows:

1. Extract duration information stored in MIDI files by the music 21 library. Fix the BPM of this MIDI file for more straightforward calculation later.
2. Through BPM and duration, the last time of each note and chord can be calculated by the formula:

   Playing time= 60.0 * duration/ bpm.
3. Record the start time and end time for each note and chord.

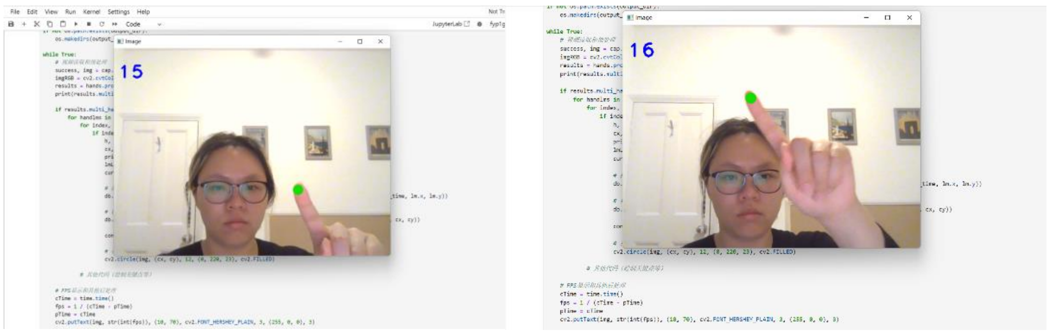


```
intervals = [(float(start), float(end)) for start, end in matches]

print(intervals)
```

```
Chord: C2.G2, Start: 0.00, Duration: 0.22, End: 0.22
Note: G3, Start: 0.22, Duration: 0.58, End: 0.80
Chord: C4.E4.G4.E4.C4, Start: 0.80, Duration: 0.22, End: 1.01
Note: E3, Start: 1.01, Duration: 0.22, End: 1.23
Note: A4, Start: 1.23, Duration: 0.22, End: 1.45
Chord: A3.B1, Start: 1.45, Duration: 0.22, End: 1.67
Note: G3, Start: 1.67, Duration: 0.58, End: 2.25
Chord: G2.F3, Start: 2.25, Duration: 0.22, End: 2.46
Chord: D4.F4.G4.F4.D4, Start: 2.46, Duration: 0.22, End: 2.68
Note: A4, Start: 2.68, Duration: 0.22, End: 2.90
Chord: A3.C2, Start: 2.90, Duration: 0.22, End: 3.12
Note: G3, Start: 3.12, Duration: 0.58, End: 3.70
Chord: G2.E3, Start: 3.70, Duration: 0.22, End: 3.91
Chord: C4.E4.G4.E4.C4, Start: 3.91, Duration: 0.22, End: 4.13
Note: A4, Start: 4.13, Duration: 0.22, End: 4.35
Chord: A3.E2, Start: 4.35, Duration: 0.22, End: 4.57
Note: B3, Start: 4.57, Duration: 0.58, End: 5.14
Chord: E3.G3, Start: 5.14, Duration: 0.22, End: 5.36
```

Fig12 Result of lasting time for notes and chords

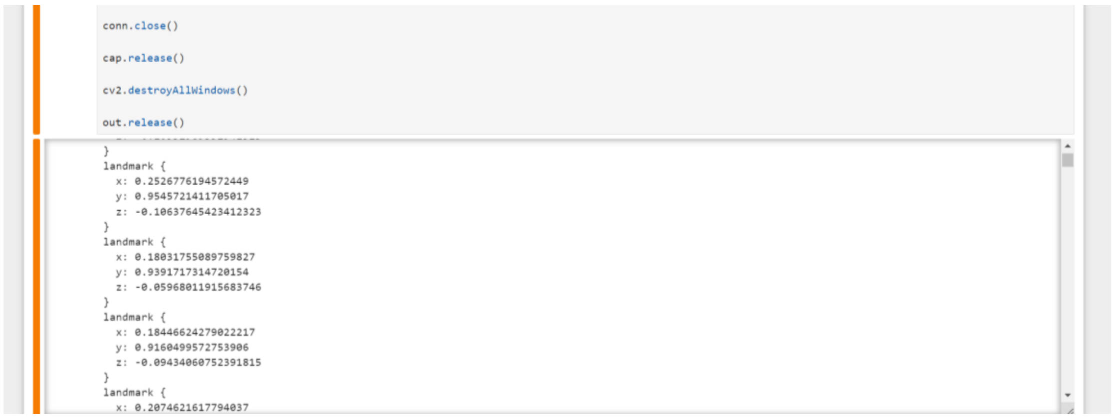
4. The user listens to the same MIDI file and moves their hand according to the rhythm like a conductor. Mediapipe framework will detect the chosen point on hand and recognize its coordinates.



```
while True:
    success, img = cap.
    imgRGB = cv2.cvtCol
    results = hands.pro
    print(results.multi

    if results.multi_ha
        for handlms in
            for index,
                if inde

                cv2.circle(img, (cx, cy), 12, (0, 220, 23), cv2.FILLED)

    cTime = time.time()
    fps = 1 / (cTime - pTime)
    pTime = cTime
    cv2.putText(img, str(int(fps)), (10, 70), cv2.FONT_HERSHEY_PLAIN, 3, (255, 0, 0), 3)
```

Fig13 User move hand when music playing, the code reference of using Mediapipe framework to capture hand is :

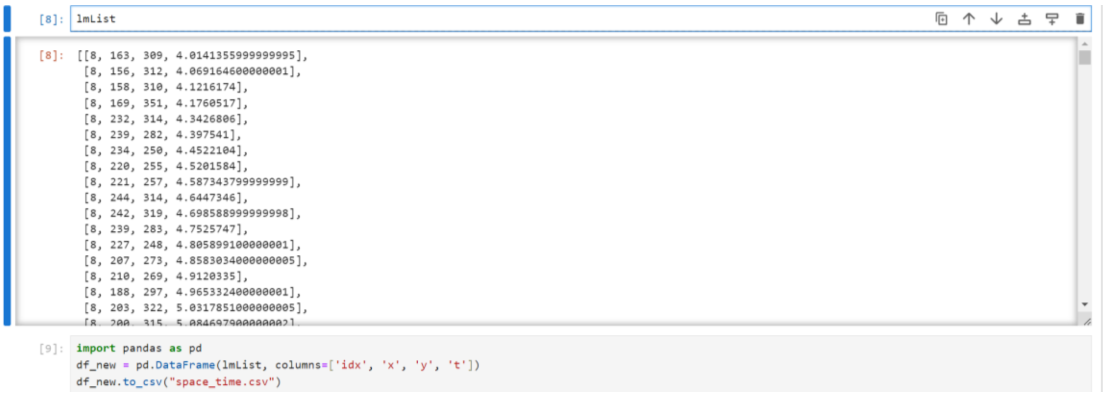
https://blog.csdn.net/dgvv4/article/details/122023047.



```
conn.close()

cap.release()

cv2.destroyAllWindows()

out.release()
```

```
}
landmark {
  x: 0.2526776194572449
  y: 0.9545721411705017
  z: -0.10637645423412323
}
landmark {
  x: 0.18031755089759827
  y: 0.9391717314720154
  z: -0.05968011915683746
}
landmark {
  x: 0.18446624279022217
  y: 0.9160499572753906
  z: -0.09434060752391815
}
landmark {
  x: 0.2074621617794037
```

Fig14 Data recorded by Mediapipe framework



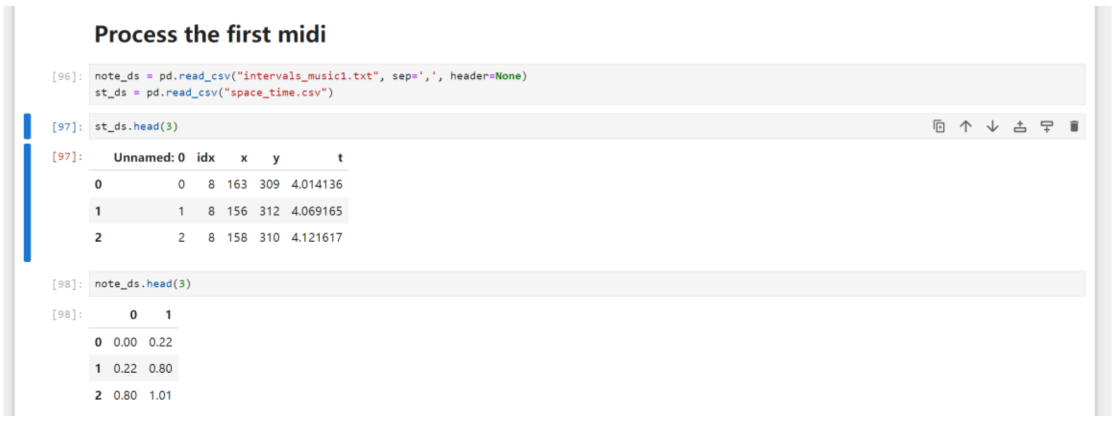
```
[8]: lmList
```

```
[8]: [[8, 163, 309, 4.0141355999999995],
      [8, 156, 312, 4.069164600000001],
      [8, 158, 310, 4.1216174],
      [8, 169, 351, 4.1760517],
      [8, 232, 314, 4.3426806],
      [8, 239, 282, 4.397541],
      [8, 234, 250, 4.4522104],
      [8, 220, 255, 4.5201584],
      [8, 221, 257, 4.587343799999999],
      [8, 244, 314, 4.6447346],
      [8, 242, 319, 4.698588999999998],
      [8, 239, 283, 4.7525747],
      [8, 227, 248, 4.805899100000001],
      [8, 207, 273, 4.858303400000005],
      [8, 210, 269, 4.9120335],
      [8, 188, 297, 4.965332400000001],
      [8, 203, 322, 5.031785100000005],
      [8, 200, 315, 5.084697900000002],
```

```
[9]: import pandas as pd
     df_new = pd.DataFrame(lmList, columns=['idx', 'x', 'y', 't'])
     df_new.to_csv("space_time.csv")
```

Fig15 Dataset with X,Y coordinates after normalized and timestamp

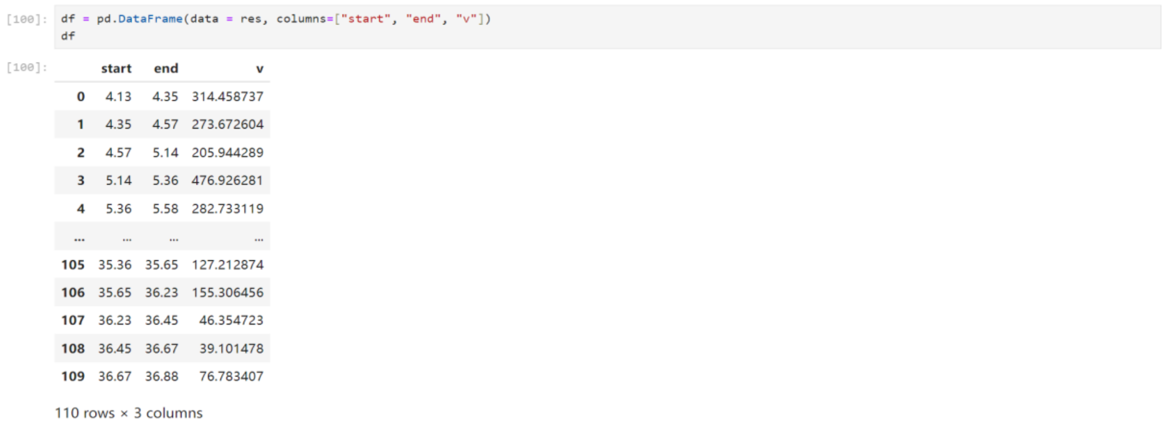
5. Align the start time of the musical notes with the start time of the hand movements and calculate the speed of the hand at the moment each note is played.



Process the first midi

```
[96]: note_ds = pd.read_csv("intervals_music1.txt", sep=',', header=None)
      st_ds = pd.read_csv("space_time.csv")

[97]: st_ds.head(3)

[97]:    Unnamed: 0  idx    x    y         t
      0           0    8  163  309  4.014136
      1           1    8  156  312  4.069165
      2           2    8  158  310  4.121617

[98]: note_ds.head(3)

[98]:       0     1
      0  0.00  0.22
      1  0.22  0.80
      2  0.80  1.01
```

Fig16 Check where Mediapipe start to capture hand position



```
[100]: df = pd.DataFrame(data = res, columns=["start", "end", "v"])
       df

[100]:     start    end           v
       0    4.13   4.35  314.458737
       1    4.35   4.57  273.672604
       2    4.57   5.14  205.944289
       3    5.14   5.36  476.926281
       4    5.36   5.58  282.733119
       ...   ...    ...         ...
       105 35.36  35.65  127.212874
       106 35.65  36.23  155.306456
       107 36.23  36.45   46.354723
       108 36.45  36.67   39.101478
       109 36.67  36.88   76.783407

       110 rows × 3 columns
```

Fig17 The calculated velocity and time duration

6.3 Evaluation

There are two advantages of this method. First, it can solve the problem of the number of features for duration production not being enough. Second, by collecting data on velocity and note duration, their mapping relationship can be built by a machine learning model, and it has personalized conducting style of operator.

However, the shortages of it are also clear:
1. This method is cumbersome and involves numerous steps, and also time-consuming;
2. It requires a high degree of temporal precision at every step. If the time extracted from the notes through the music21 library does not align with the timestamps obtained from MediaPipe framework, the results will be inaccurate. This temporal discrepancy is influenced by several factors, such as the computer's video refresh rate and adjustments to the BPM, meaning the data obtained through this method is very likely to be highly erroneous.
3.This method presupposes to some extent that upon hearing different musical rhythms, a person would alter the speed of their hand movements. However, there is no evidence or verification to substantiate that there is a correlation between hand speed and the rhythm of music. Therefore, the credibility of this method could be better.

At the same time, when comparing the results, there might not be an obvious correlation between hand movement speed and note duration. This outcome could be attributed to several factors:
1. The speed may have been inaccurately measured due to device issues.
2. Adjusting the BPM might have caused changes in the note start and end times.
3. There could be a delay due to the reaction time needed for a person to respond after hearing the music
4. There may not be a significant correlation between hand speed and musical rhythm.
In summary, the method did not accomplish its intended task in this exploration.

7.Second generation Method 2： Assign Duration to note directly

7.1 Plan

The first solution aims to establish a mathematical relationship between hand movement speed and note duration. The second proposed solution involves directly mapping the captured x-coordinates to durations according to their area on the screen and then assigning these durations to the generated musical notes. Fig18 shows this concept:

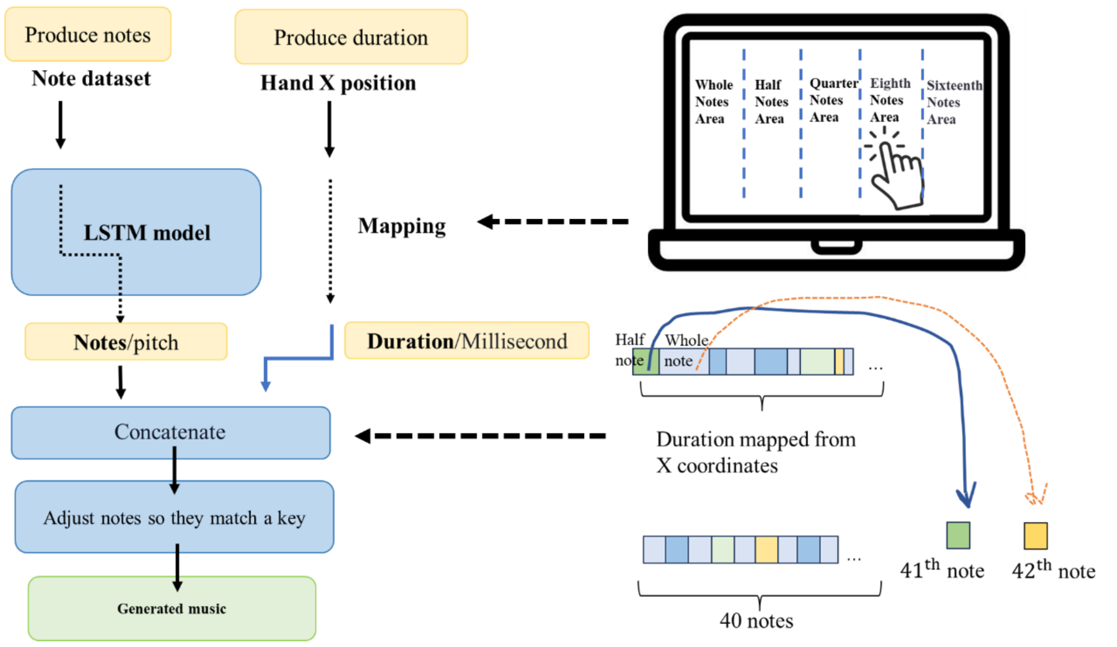


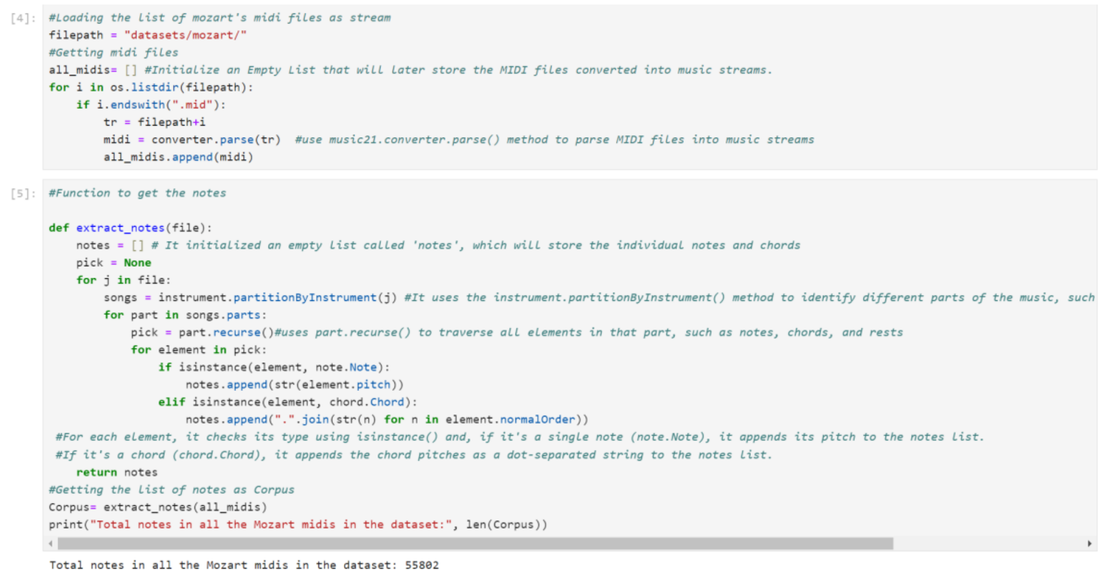
Fig18 Overview of Method2

The detailed method divides the screen into five distinct areas, each associated with a different note duration: whole notes, half notes, quarter notes, eighth notes, and sixteenth notes. The system then determines the segment to which a point belongs based on its x-value and returns the corresponding duration.

7.2 Implementations

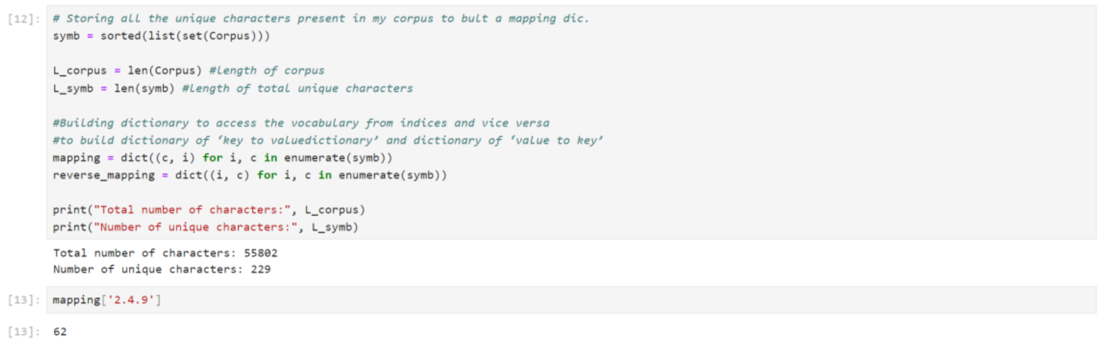
The detailed implementation of Method 2 is as follows:

1. The first step is loading data and pre-processing the notes dataset, making data suitable for training.



```
[4]: #Loading the list of mozart's midi files as stream
     filepath = "datasets/mozart/"
     #Getting midi files
     all_midis= [] #Initialize an Empty List that will later store the MIDI files converted into music streams.
     for i in os.listdir(filepath):
         if i.endswith(".mid"):
             tr = filepath+i
             midi = converter.parse(tr)  #use music21.converter.parse() method to parse MIDI files into music streams
             all_midis.append(midi)

[5]: #Function to get the notes

     def extract_notes(file):
         notes = [] # It initialized an empty list called 'notes', which will store the individual notes and chords
         pick = None
         for j in file:
             songs = instrument.partitionByInstrument(j) #It uses the instrument.partitionByInstrument() method to identify different parts of the music, such
             for part in songs.parts:
                 pick = part.recurse()#uses part.recurse() to traverse all elements in that part, such as notes, chords, and rests
                 for element in pick:
                     if isinstance(element, note.Note):
                         notes.append(str(element.pitch))
                     elif isinstance(element, chord.Chord):
                         notes.append(".".join(str(n) for n in element.normalOrder))
      #For each element, it checks its type using isinstance() and, if it's a single note (note.Note), it appends its pitch to the notes list.
      #If it's a chord (chord.Chord), it appends the chord pitches as a dot-separated string to the notes list.
         return notes
     #Getting the list of notes as Corpus
     Corpus= extract_notes(all_midis)
     print("Total notes in all the Mozart midis in the dataset:", len(Corpus))

     Total notes in all the Mozart midis in the dataset: 55802
```

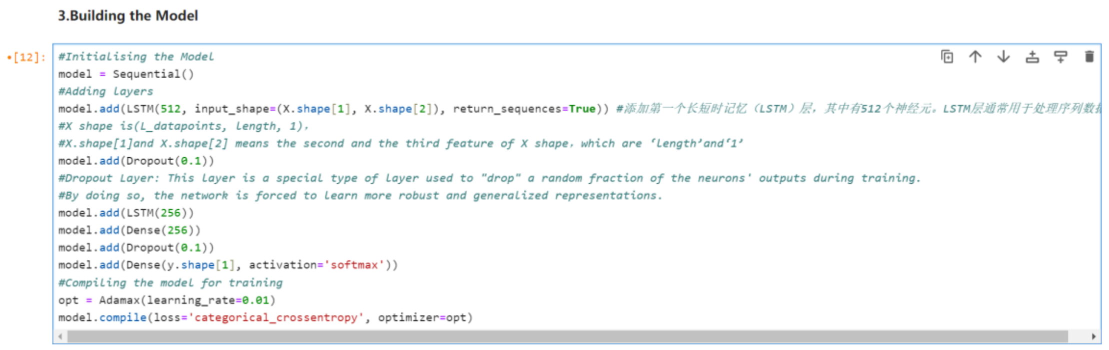
Fig19 Load data



```
[12]: # Storing all the unique characters present in my corpus to bult a mapping dic.
      symb = sorted(list(set(Corpus)))

      L_corpus = len(Corpus) #length of corpus
      L_symb = len(symb) #length of total unique characters

      #Building dictionary to access the vocabulary from indices and vice versa
      #to build dictionary of ‘key to valuedictionary’ and dictionary of ‘value to key’
      mapping = dict((c, i) for i, c in enumerate(symb))
      reverse_mapping = dict((i, c) for i, c in enumerate(symb))

      print("Total number of characters:", L_corpus)
      print("Number of unique characters:", L_symb)

      Total number of characters: 55802
      Number of unique characters: 229

[13]: mapping['2.4.9']

[13]: 62
```

Fig20 Pre-process data

2. Build LSTM model for training.

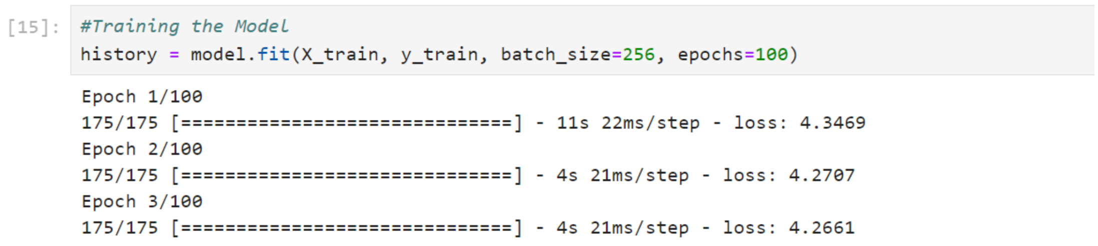


3.Building the Model

```
•[12]: #Initialising the Model
       model = Sequential()
       #Adding layers
       model.add(LSTM(512, input_shape=(X.shape[1], X.shape[2]), return_sequences=True)) #添加第一个长短时记忆（LSTM）层，其中有512个神经元。LSTM层通常用于处理序列数
       #X shape is(L_datapoints, length, 1),
       #X.shape[1]and X.shape[2] means the second and the third feature of X shape, which are ‘length’and‘1’
       model.add(Dropout(0.1))
       #Dropout Layer: This layer is a special type of layer used to "drop" a random fraction of the neurons' outputs during training.
       #By doing so, the network is forced to learn more robust and generalized representations.
       model.add(LSTM(256))
       model.add(Dense(256))
       model.add(Dropout(0.1))
       model.add(Dense(y.shape[1], activation='softmax'))
       #Compiling the model for training
       opt = Adamax(learning_rate=0.01)
       model.compile(loss='categorical_crossentropy', optimizer=opt)
```

Fig21 Model building

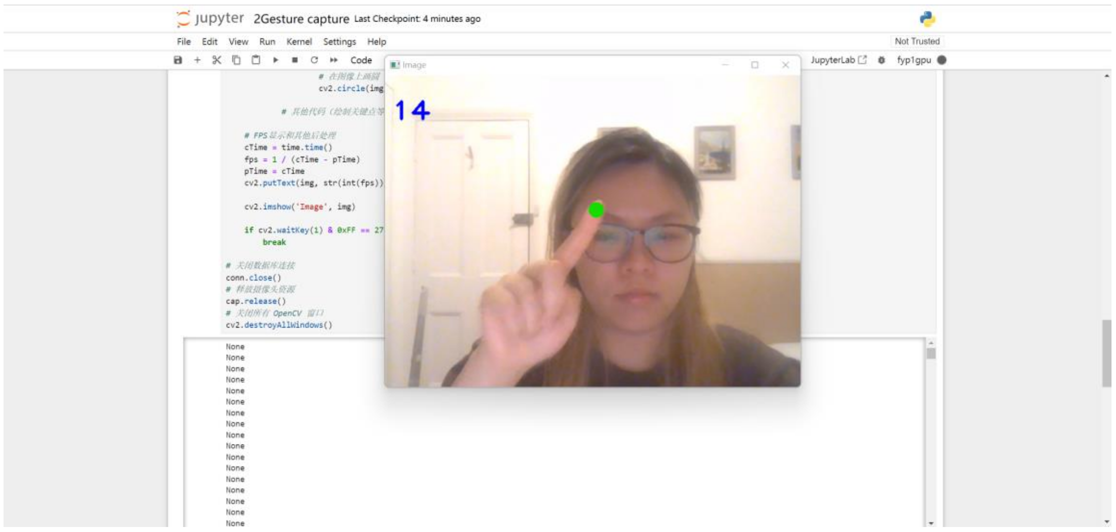
3. Training model for 100 epochs.



```
[15]: #Training the Model
      history = model.fit(X_train, y_train, batch_size=256, epochs=100)

      Epoch 1/100
      175/175 [==============================] - 11s 22ms/step - loss: 4.3469
      Epoch 2/100
      175/175 [==============================] - 4s 21ms/step - loss: 4.2707
      Epoch 3/100
      175/175 [==============================] - 4s 21ms/step - loss: 4.2661
```

Fig22 Traning for 100 epochs

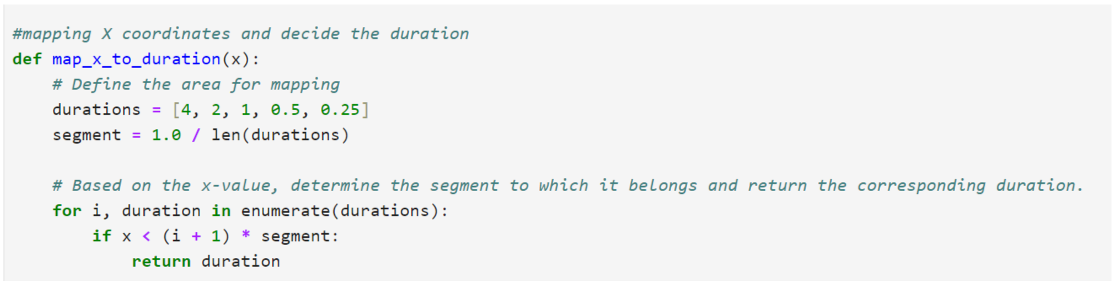
4.Capture hand position by Mediapipe framework.


jupyter 2Gesture capture Last Checkpoint: 4 minutes ago
File Edit View Run Kernel Settings Help
Code
Image
14
Not Trusted
JupyterLab
fyp1gpu

```
            # 在图像上画圆
            cv2.circle(img
        # 其他代码（绘制关键点等
    # FPS显示和其他后处理
    cTime = time.time()
    fps = 1 / (cTime - pTime)
    pTime = cTime
    cv2.putText(img, str(int(fps))
    
    cv2.imshow('Image', img)
    
    if cv2.waitKey(1) & 0xFF == 27
        break

# 关闭数据库连接
conn.close()
# 释放摄像头资源
cap.release()
# 关闭所有 OpenCV 窗口
cv2.destroyAllWindows()

None
None
None
None
None
None
None
None
None
None
None
None
None
None
None
None
None
```

Fig23 Capture hand position

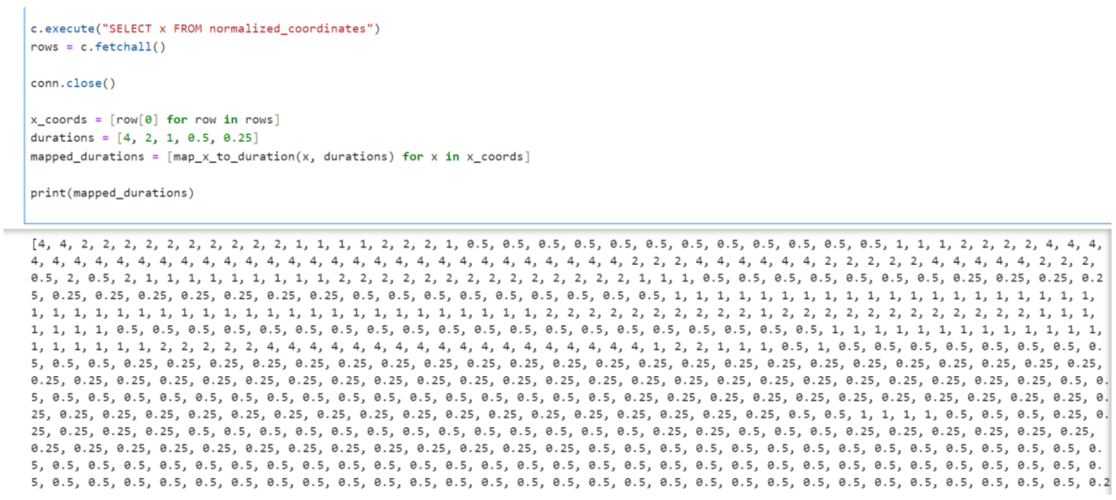
5.Define a function to map X-coordinates into duration. This function operates by comparing the input x value to the range of each segment, and then returning the duration value corresponding to the matched segment. The whole screen is evenly divided into five parts, each part corresponding to a different duration. There are five different types of durations: a value of 4 corresponds to whole notes, 2 represents half notes, 1 means quarter notes, with subsequent values following this pattern.



```
#mapping X coordinates and decide the duration
def map_x_to_duration(x):
    # Define the area for mapping
    durations = [4, 2, 1, 0.5, 0.25]
    segment = 1.0 / len(durations)

    # Based on the x-value, determine the segment to which it belongs and return the corresponding duration.
    for i, duration in enumerate(durations):
        if x < (i + 1) * segment:
            return duration
```

Fig24 Mapping definition

6.Map X-coordinates to durations and save the outcomes.

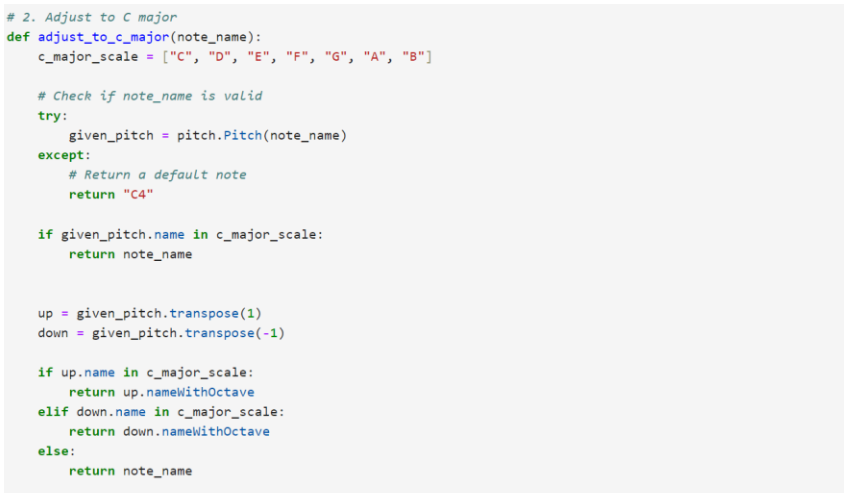


```
c.execute("SELECT x FROM normalized_coordinates")
rows = c.fetchall()

conn.close()

x_coords = [row[0] for row in rows]
durations = [4, 2, 1, 0.5, 0.25]
mapped_durations = [map_x_to_duration(x, durations) for x in x_coords]

print(mapped_durations)
```

```
[4, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 2, 2, 2, 1, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1, 1, 1, 2, 2, 2, 2, 4, 4, 4,
4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 2, 2, 2, 4, 4, 4, 4, 4, 4, 2, 2, 2, 2, 2, 4, 4, 4, 4, 4, 2, 2, 2,
0.5, 2, 0.5, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.25, 0.25, 0.25, 0.2
5, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1,
1, 1, 1, 1, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 1, 2, 2, 1, 1, 1, 0.5, 1, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.
5, 0.5, 0.5, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25,
0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.5, 0.
5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.
25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.5, 0.5, 1, 1, 1, 1, 0.5, 0.5, 0.5, 0.25, 0.
25, 0.25, 0.25, 0.25, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.25, 0.25, 0.5, 0.5, 0.5, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25,
0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.
5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.
5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.2
```

Fig25 Results of X-coordinates mapped into duration

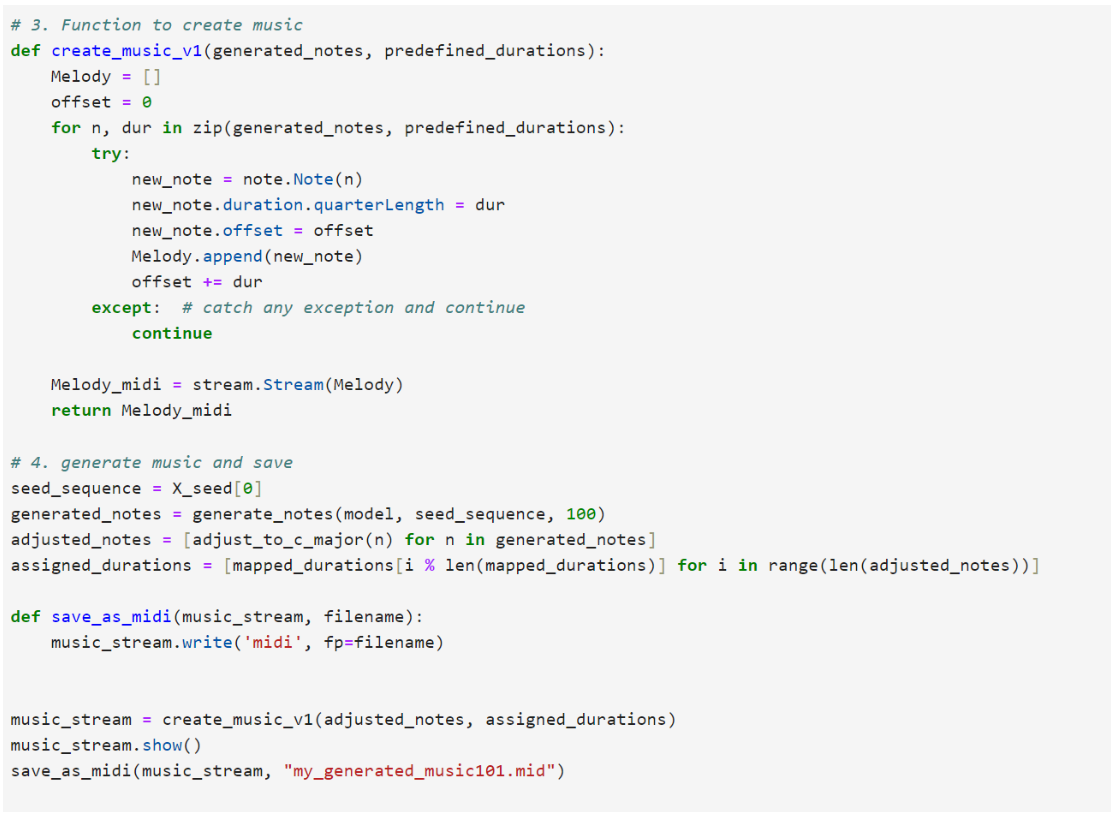
6. Use a half-step increase or decrease to find the closest note in the C major scale.



```
# 2. Adjust to C major
def adjust_to_c_major(note_name):
    c_major_scale = ["C", "D", "E", "F", "G", "A", "B"]

    # Check if note_name is valid
    try:
        given_pitch = pitch.Pitch(note_name)
    except:
        # Return a default note
        return "C4"

    if given_pitch.name in c_major_scale:
        return note_name


    up = given_pitch.transpose(1)
    down = given_pitch.transpose(-1)

    if up.name in c_major_scale:
        return up.nameWithOctave
    elif down.name in c_major_scale:
        return down.nameWithOctave
    else:
        return note_name
```

Fig26 Adjust generated notes

7. Combine the predicted note results from the model with the mapped duration values.



```python
# 3. Function to create music
def create_music_v1(generated_notes, predefined_durations):
    Melody = []
    offset = 0
    for n, dur in zip(generated_notes, predefined_durations):
        try:
            new_note = note.Note(n)
            new_note.duration.quarterLength = dur
            new_note.offset = offset
            Melody.append(new_note)
            offset += dur
        except:  # catch any exception and continue
            continue

    Melody_midi = stream.Stream(Melody)
    return Melody_midi

# 4. generate music and save
seed_sequence = X_seed[0]
generated_notes = generate_notes(model, seed_sequence, 100)
adjusted_notes = [adjust_to_c_major(n) for n in generated_notes]
assigned_durations = [mapped_durations[i % len(mapped_durations)] for i in range(len(adjusted_notes))]

def save_as_midi(music_stream, filename):
    music_stream.write('midi', fp=filename)


music_stream = create_music_v1(adjusted_notes, assigned_durations)
music_stream.show()
save_as_midi(music_stream, "my_generated_music101.mid")
```

Fig27 Generate music and save

8. Outcome

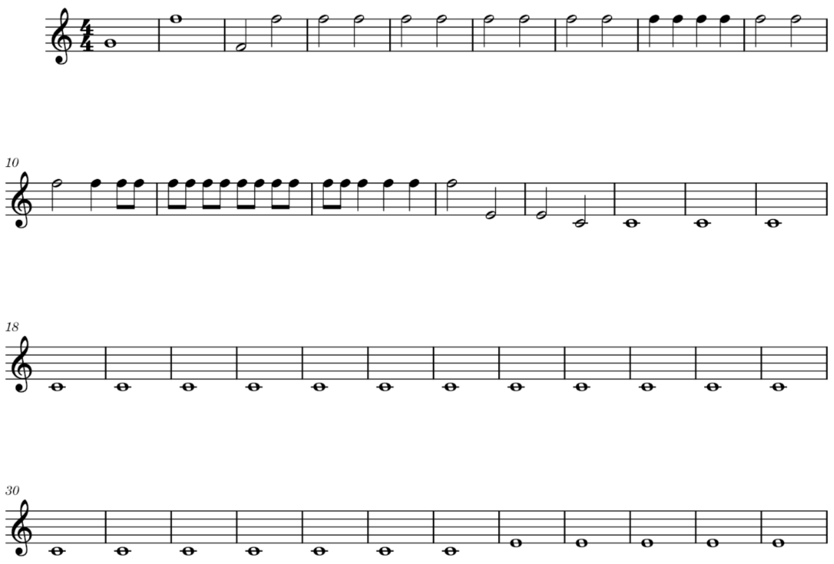


Fig28 Result

7.3 Evaluation

Evaluate from the outcome, this method can complete the task to modify the duration of generated notes. The advantage of this method is quick and direct because of it uses a mathematic way to map X-coordinates to duration. However, the disadvantages of this method are apparent as well. First, using pre-recorded X-coordinates to control music is discordant, and its rhythm is repetitive, arbitrary, and uncontrolled. It is because no feature allowing users to hear the music rhythm in real time and react to it. Second, the whole process requires multiple Jupyter notebooks to collaborate, which is cumbersome and not fluent for a real-time system.

8.Final design: Real-time framework

8.1 Real-time system introduction

Method 2 of the second generation lacks a function that allows people to hear sound and react in real time. Based on previous work, a real-time framework was introduced to form an action-production loop.
This is how whole system works:

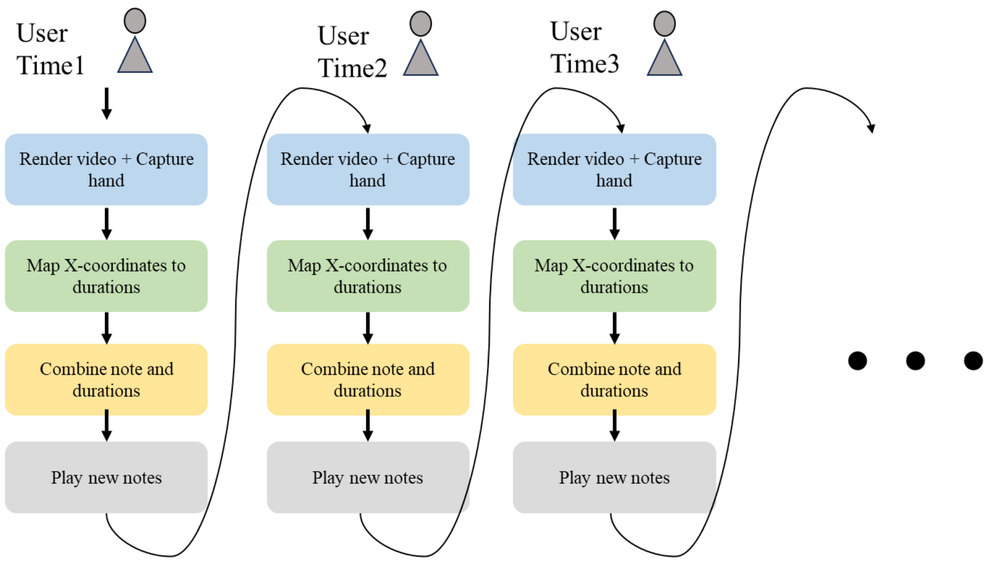

User
Time1
User
Time2
User
Time3
Render video + Capture hand
Map X-coordinates to durations
Combine note and durations
Play new notes
Render video + Capture hand
Map X-coordinates to durations
Combine note and durations
Play new notes
Render video + Capture hand
Map X-coordinates to durations
Combine note and durations
Play new notes

Fig 29 Overview of Real-time system

Before using this real-time system, it is necessary first to train the model and then use the LSTM model to generate notes. In this example, the author chooses to produce 100 notes. Next comes the implementation of the real-time system, a program that executes four tasks in a loop. The first task is to a render video window and capture the X-coordinates of the critical point of hand. After capturing the hand's X-coordinates, the second task is to map these X-coordinates to durations. The third task then involves integrating the notes predicted by the trained model with these durations to form a new note. The final step is to play the music.

8.2 Opensource library used

I will provide a detailed description of the open-source libraries or software utilized in each task step.

Task1: In the first task of the diagram, OpenCV is used to render the video, and then the MediaPipe framework captures the X-coordinates of key points chosen in the video.

Task2: In the second task, `def ` defines a function named ` **`map_x_to_duration`**` that maps X-coordinates to durations and returns durations. This feature is available in Python by default and does not require importing modules from the standard library.

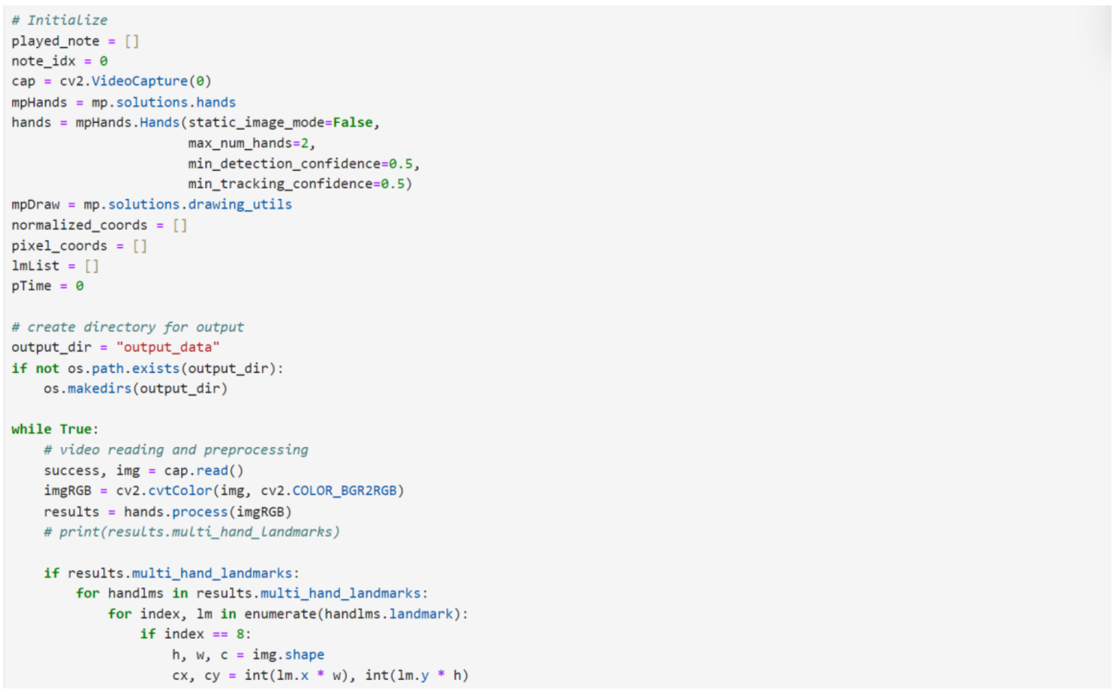
Task3: The third task utilizes the `music21` library, defining a function named `**`create_note`**` for creating and configuring musical notes.

Task4: The fourth task defines a function called `**`play_note`**` by `**`stream`**` and `**`midi.realtime`**` object in `music21`library.

8.3 Technical detail

In this section, I will explain the meaning and functions of important code.

8.3.1 Task 1



```python
# Initialize
played_note = []
note_idx = 0
cap = cv2.VideoCapture(0)
mpHands = mp.solutions.hands
hands = mpHands.Hands(static_image_mode=False,
                      max_num_hands=2,
                      min_detection_confidence=0.5,
                      min_tracking_confidence=0.5)
mpDraw = mp.solutions.drawing_utils
normalized_coords = []
pixel_coords = []
lmList = []
pTime = 0

# create directory for output
output_dir = "output_data"
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

while True:
    # video reading and preprocessing
    success, img = cap.read()
    imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    results = hands.process(imgRGB)
    # print(results.multi_hand_landmarks)

    if results.multi_hand_landmarks:
        for handlms in results.multi_hand_landmarks:
            for index, lm in enumerate(handlms.landmark):
                if index == 8:
                    h, w, c = img.shape
                    cx, cy = int(lm.x * w), int(lm.y * h)
```

Fig30 Task1 code

Python code above uses the ` **`Opencv-python (cv2)`** ` library and Google's `**`MediaPipe`**` library for video capture and gesture tracking. Task1 can be accomplished by 3 steps specifically:

1） Get video stream

Using ` cv2.VideoCapture(0)` from `cv2` library to get video stream from default camera. `mpHands = mp.solutions.hands` initialize `hand` module in `MediaPipe`, it is for detecting hand key point.

```
# Initialize
played_note = []
note_idx = 0
cap = cv2.VideoCapture(0)
mpHands = mp.solutions.hands
hands = mpHands.Hands(static_image_mode=False,
                      max_num_hands=2,
                      min_detection_confidence=0.5,
                      min_tracking_confidence=0.5)
mpDraw = mp.solutions.drawing_utils
normalized_coords = []
pixel_coords = []
lmList = []
pTime = 0
```

Fig31 Initialize setting

2） Read and convert images

In the `while True` loop, frames are read from the video stream using `cap.read()`, and the images are converted from BGR (Blue, Green, Red) format to RGB format using `cv2.cvtColor(img, cv2.COLOR_BGR2RGB)`, because MediaPipe requires RGB format for image input.

```
while True:
    # video reading and preprocessing
    success, img = cap.read()
    imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

Fig32 Convert images

3） Detect hand

` results = hands.process(imgRGB)` processes pictures after converting, finding hand in video stream and return results. `results.multi_hand_landmarks` is an attribute returned by the `MediaPipe` hand recognition model after processing the image. This attribute contains key point information of all hands detected in the current frame of the image.

```python
while True:
    # video reading and preprocessing
    success, img = cap.read()
    imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    results = hands.process(imgRGB)
    # print(results.multi_hand_landmarks)

    if results.multi_hand_landmarks:
        for handlms in results.multi_hand_landmarks:
            for index, lm in enumerate(handlms.landmark):
                if index == 8:
                    h, w, c = img.shape
                    cx, cy = int(lm.x * w), int(lm.y * h)

                    midi_pitch = int(lm.y * 60 + 30)
                    # dur = lm.x * 4 + 0.1
                    dur = map_x_to_duration(lm.x)
```

Fig33 `while True` Loop

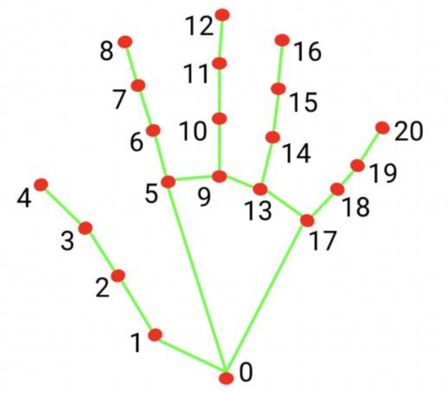
In this process, `MediaPipe` is for hand gesture recognition and tracking, namely processing the images captured and converted by `OpenCV` and returning landmarks of the hands. `OpenCV` is then used to render this processed data back into image frames for display on the screen or further processing.

```python
    if results.multi_hand_landmarks:
        for handlms in results.multi_hand_landmarks:
            for index, lm in enumerate(handlms.landmark):
                if index == 8:
                    h, w, c = img.shape
                    cx, cy = int(lm.x * w), int(lm.y * h)
```

Fig34 Relevant setting

`handlms` represents all the key points of one hand obtained from `results.multi_hand_landmarks`. In MediaPipe hand detection model, each hand is represented by a series of key points (or landmarks), which identify different parts of the hand, such as fingertips, joints.
`handlms.landmark` is a list containing all key points of this hand.
`enumerate()` is a built-in Python function used to convert an iterable list or tuple into an enumeration object. The `enumerate()` function returns two values: the index and the element itself (in this context, a key point object `lm`). This code iterates through all the key points of a hand, providing both the index and the coordinate data for each key point, allowing for the processing of specific key points (In this case is index=8).

The choice of index=8 is based on the MediaPipe Hand Landmarks Detection guide. According to this guide, the number 8 represents the tip of the index finger. The reason for focusing on a single landmark, as mentioned previously, is that capturing just one key point essentially achieves the same result as detecting all key points, but with the added benefits of reduced computational load and increased efficiency in task completion. As for which specific finger to choose, it does not make a significant difference in this context.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Fig35 21 hand-knuckle coordinates in hand landmark model bundle. From https://developers.google.com/mediapipe/solutions/vision/hand_landmarker(*Hand landmarks detection guide | MediaPipe*, 2023)

`h, w, c = img.shape` in above code is to obtain the dimensions and color channel count of an image. `h(height)` is vertical dimension of image, `w(width)` is horizontal dimension of image, `c(channel)` is number of color channels in an image.

In the code, `cx, cy = int(lm.x * w), int(lm.y * h)` is used to convert the normalized coordinates of the hand landmarks detected by MediaPipe into actual pixel coordinates of the image. Since `lm.x` and `lm.y` are normalized (values between 0 and 1), they are multiplied by the width and height of the image, respectively, to obtain their actual positions in the image. The converted coordinates
`cx` and `cy` represent the horizontal and vertical positions of the landmarks in the image, respectively.

8.3.2 Task2

```
#mapping X coordinates and decide the duration
def map_x_to_duration(x):
    # Define the area for mapping
    durations = [4, 2, 1, 0.5, 0.25]
    segment = 1.0 / len(durations)

    # Based on the x-value, determine the segment to which it belongs and return the corresponding duration.
    for i, duration in enumerate(durations):
        if x < (i + 1) * segment:
            return duration
```

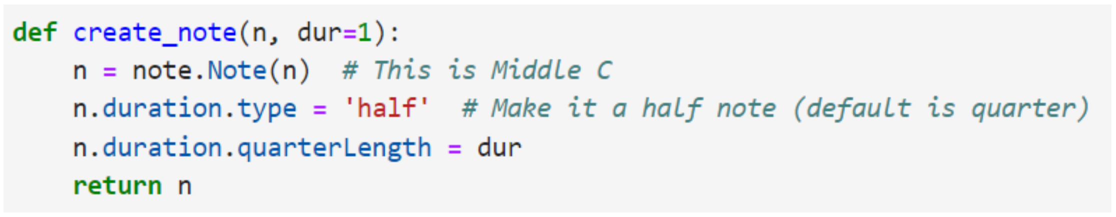
Fig36 Definition of function `map_x_to_duration`

```
midi_pitch = int(lm.y * 60 + 30)
# dur = lm.x * 4 + 0.1
dur = map_x_to_duration(lm.x)
```

Fig37 Convert `lm.x` to `dur`

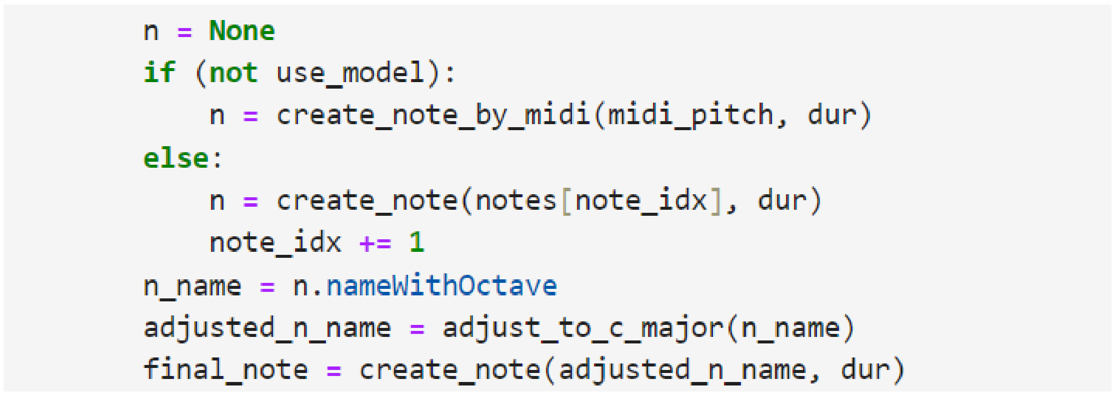
The detailed mapping regulations are introduced in `**7. Second generation Method 2**`. Here, this defined function is implemented to map `lm.x` to `dur`, where `dur` is an abbreviation for duration.

8.3.3 Task3



```
def create_note(n, dur=1):
    n = note.Note(n)  # This is Middle C
    n.duration.type = 'half'   # Make it a half note (default is quarter)
    n.duration.quarterLength = dur
    return n
```

Fig38 Define function `create_note`



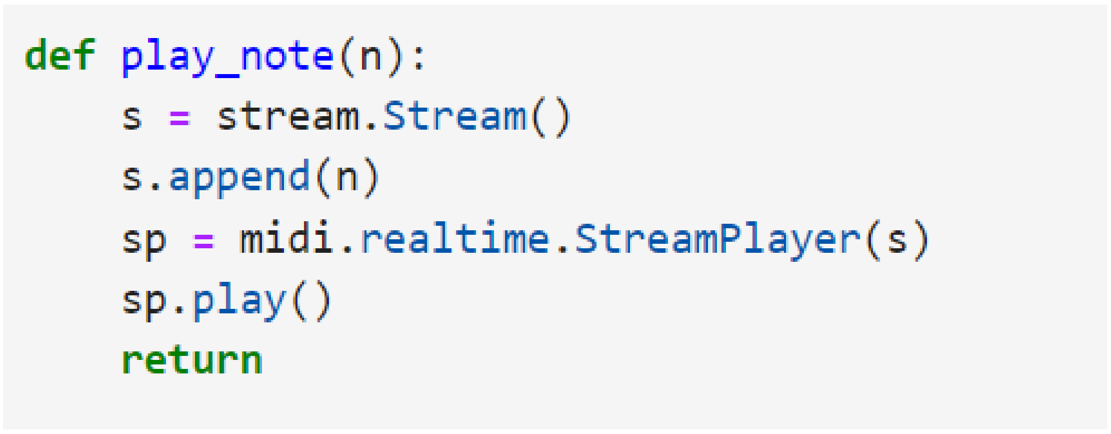
```
        n = None
        if (not use_model):
            n = create_note_by_midi(midi_pitch, dur)
        else:
            n = create_note(notes[note_idx], dur)
            note_idx += 1
        n_name = n.nameWithOctave
        adjusted_n_name = adjust_to_c_major(n_name)
        final_note = create_note(adjusted_n_name, dur)
```

Fig39 Combining note and duration

In Task3, a function called `create_note()` is defined to build and set a new music note. `def create_note(n, dur=1)`accept two parameters: `n` and `dur`. `n` is for specifying the note, `dur` is used to determine the duration of the note. `n = note.Note(n)` using the `Note` class in `music21` library to create a new note object whose duration equals to a half note.

New object called `final_note` will be created by `create_note` function, and it is final note which combines note and mapped duration. It will be played in task4. `final_note` receives `adjusted_n_name` as note and `dur` as duration. `adjusted_n_name` is note given from LSTM model and adjusted into C major.

8.3.4 Task4



```python
def play_note(n):
    s = stream.Stream()
    s.append(n)
    sp = midi.realtime.StreamPlayer(s)
    sp.play()
    return
```

Fig40 Define \`play_note\` function

Task 4 will play generated \` `final_note` \` by \`play_note\` function.
This function realizes on \``stream`\` and \``midi.realtime`\` object in \``music21`\` library to implement music playback functionality. It creates a music stream and add single note into this stream just like a music sheet, then MIDI player will activate music playing.

\``s = stream.Stream()`\`: A new \``Stream`\` object is created using the \``music21.stream`\` module. In \` `music21`\`, \``Stream`\` is a container used for storing and manipulating musical elements such as notes, chords.

\``sp = midi.realtime.StreamPlayer(s)`\`: The \``StreamPlayer`\` class from the \`music21.midi.realtime\` module is used. The \``StreamPlayer`\` class is utilized for real-time playback of music content within a \``Stream`\`.

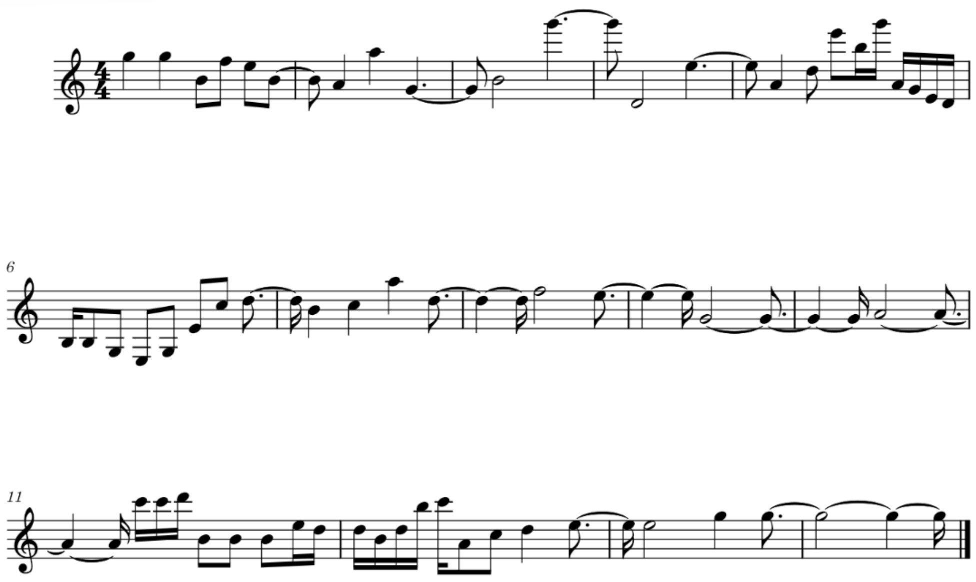


Fig41 Outcome of real-time system

8.4 Real-time framework improvement

After completing the construction of the real-time system, its performance was tested. It was observed that the system tends to lag in areas with longer durations, while it is more responsive in areas with shorter durations. The reason for this phenomenon is that the program operates serially, meaning that in the first loop, if the duration is longer, the music playback time is also extended, requiring more time before proceeding to the first task of the next loop.

To improve the issue of system lag, the author transformed the original serial program into a parallel one, which effectively alleviated the lagging problem.

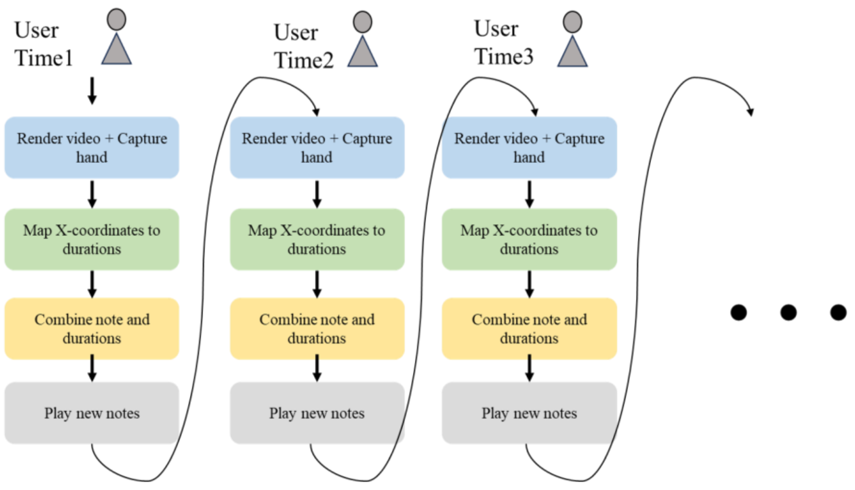

User Time1
User Time2
User Time3
Render video + Capture hand
Map X-coordinates to durations
Combine note and durations
Play new notes

Fig42 Program before improvement is working like this picture shows

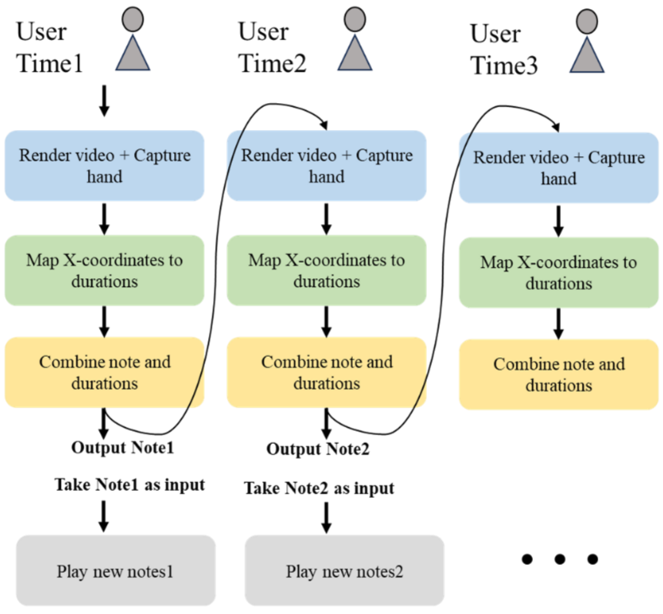
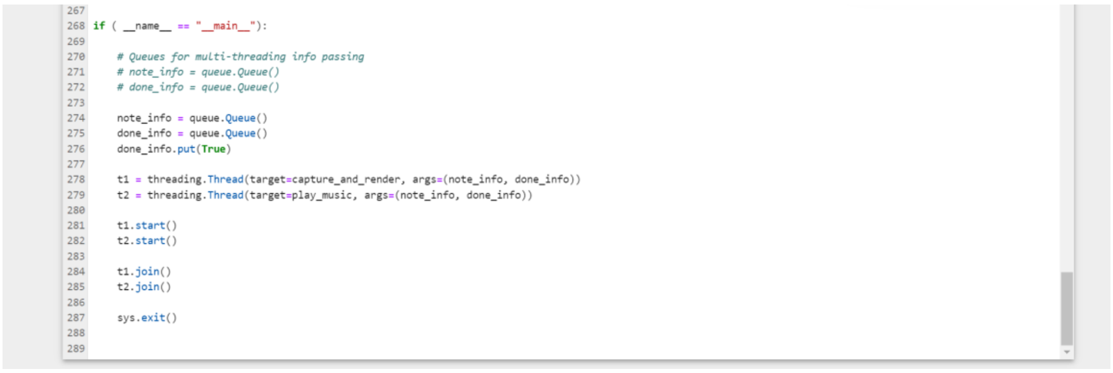


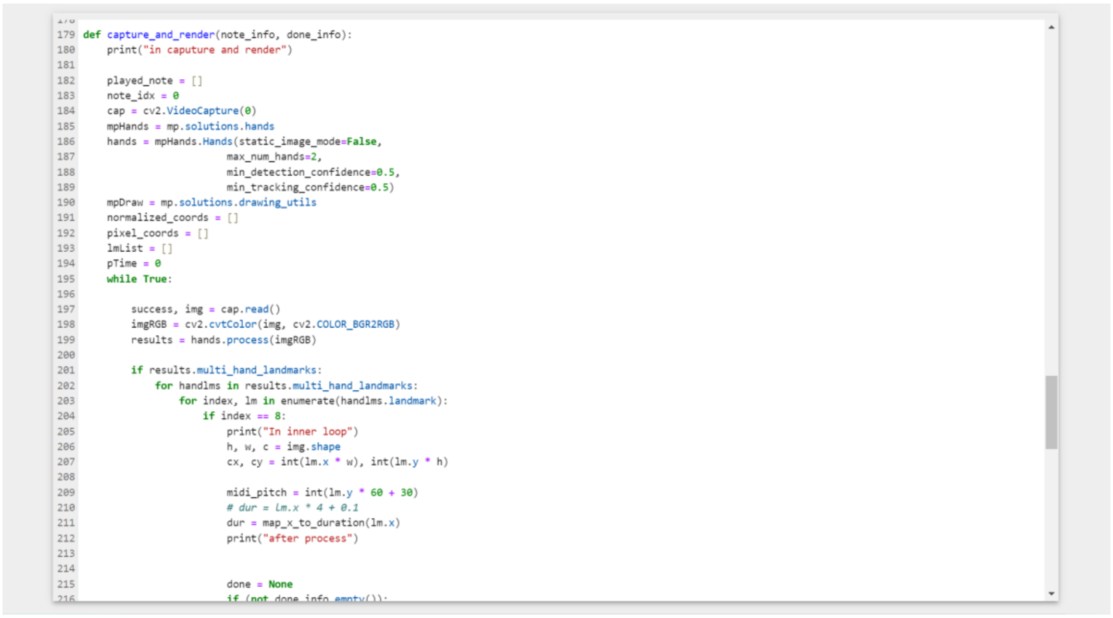
Fig43 Program after development is working like this picture shows

The following code demonstrates this: The original program was divided into two main tasks, t1 and t2, respectively. t1 is responsible for capturing and rendering video, while t2 is used for playing music.



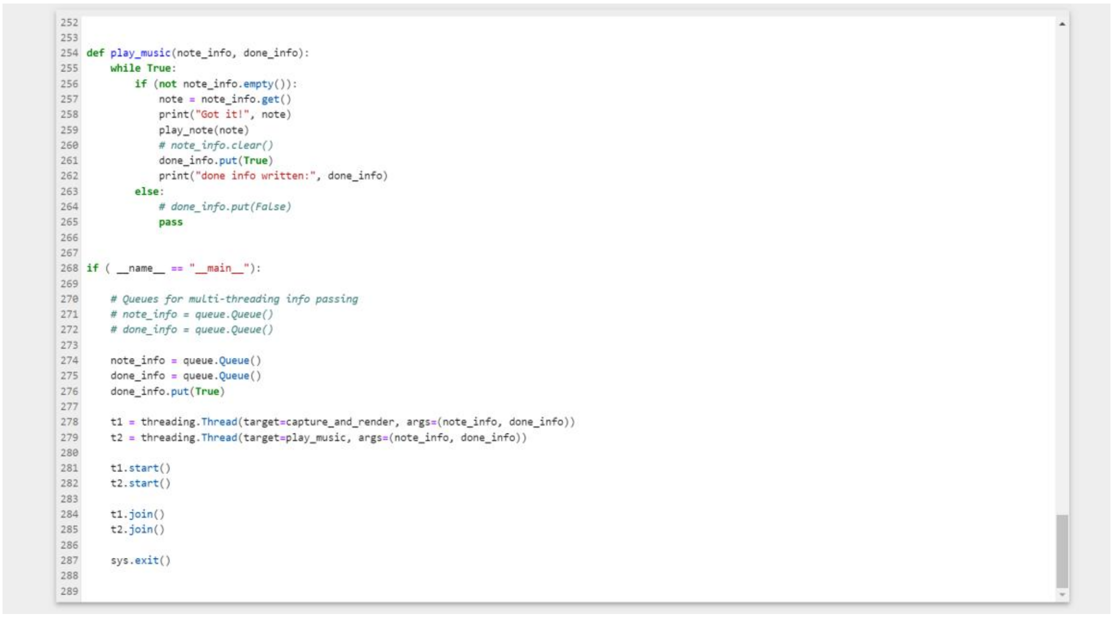
```
if ( __name__ == "__main__"):

    # Queues for multi-threading info passing
    # note_info = queue.Queue()
    # done_info = queue.Queue()

    note_info = queue.Queue()
    done_info = queue.Queue()
    done_info.put(True)

    t1 = threading.Thread(target=capture_and_render, args=(note_info, done_info))
    t2 = threading.Thread(target=play_music, args=(note_info, done_info))

    t1.start()
    t2.start()

    t1.join()
    t2.join()

    sys.exit()

```

Fig44 Separate task1 and task2



```
def capture_and_render(note_info, done_info):
    print("in caputure and render")

    played_note = []
    note_idx = 0
    cap = cv2.VideoCapture(0)
    mpHands = mp.solutions.hands
    hands = mpHands.Hands(static_image_mode=False,
                          max_num_hands=2,
                          min_detection_confidence=0.5,
                          min_tracking_confidence=0.5)
    mpDraw = mp.solutions.drawing_utils
    normalized_coords = []
    pixel_coords = []
    lmList = []
    pTime = 0
    while True:

        success, img = cap.read()
        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        results = hands.process(imgRGB)

        if results.multi_hand_landmarks:
            for handlms in results.multi_hand_landmarks:
                for index, lm in enumerate(handlms.landmark):
                    if index == 8:
                        print("In inner loop")
                        h, w, c = img.shape
                        cx, cy = int(lm.x * w), int(lm.y * h)

                        midi_pitch = int(lm.y * 60 + 30)
                        # dur = lm.x * 4 + 0.1
                        dur = map_x_to_duration(lm.x)
                        print("after process")


                        done = None
                        if (not done_info.empty()):
```

Fig45 Task1: Capture hand and render video



```


def play_music(note_info, done_info):
    while True:
        if (not note_info.empty()):
            note = note_info.get()
            print("Got it!", note)
            play_note(note)
            # note_info.clear()
            done_info.put(True)
            print("done info written:", done_info)
        else:
            # done_info.put(False)
            pass


if ( __name__ == "__main__"):

    # Queues for multi-threading info passing
    # note_info = queue.Queue()
    # done_info = queue.Queue()

    note_info = queue.Queue()
    done_info = queue.Queue()
    done_info.put(True)

    t1 = threading.Thread(target=capture_and_render, args=(note_info, done_info))
    t2 = threading.Thread(target=play_music, args=(note_info, done_info))

    t1.start()
    t2.start()

    t1.join()
    t2.join()

    sys.exit()

```

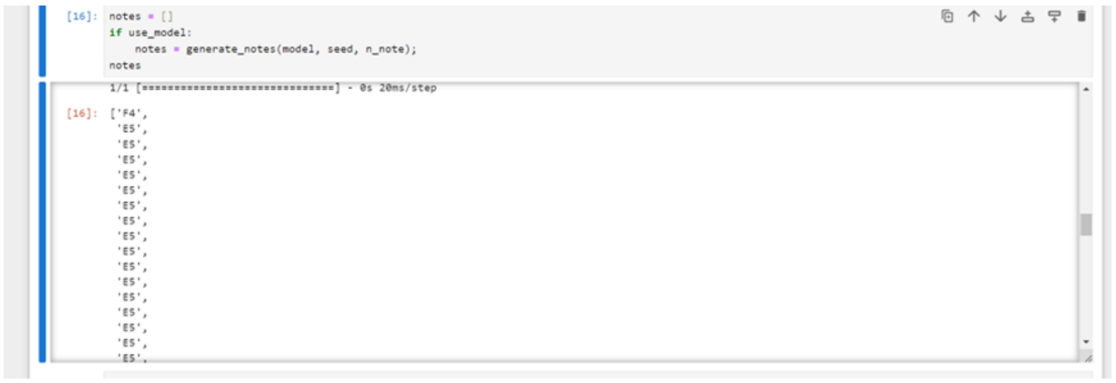
Fig46 Task2: Play music

This development has achieved the effect that video capturing and music playing is smoother than before.
The comparison video can be found from link:
To demonstration before improvement: https://youtu.be/SR9Q436VXAg

To demonstration after development: https://youtu.be/0LWmchE5y5Q

8.5 Resolve issues of note redundancy

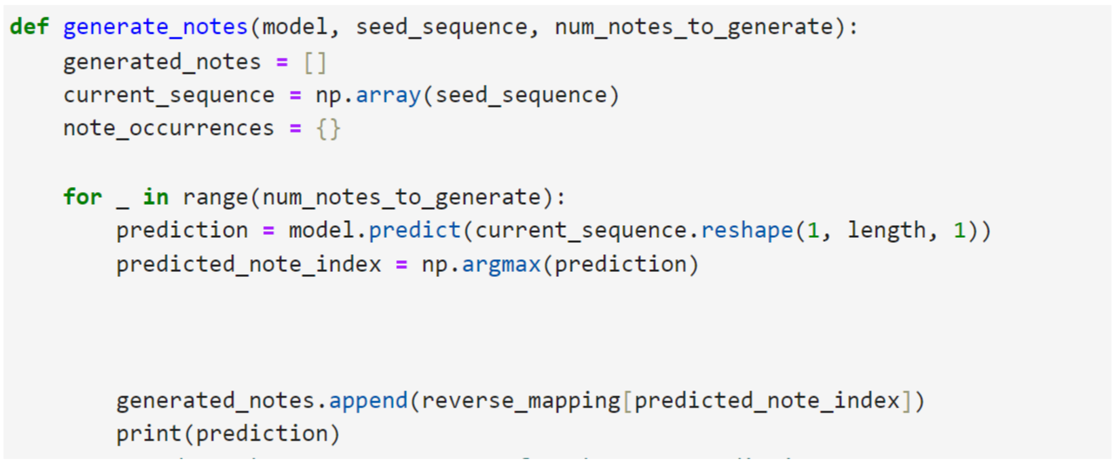
One persistent issue remains through previous iterations and tests: the LSTM model predicts many repeated notes. As illustrated, the same note appears dozens of times, undermining the diversity of the music. Therefore, to avoid such repetition, it is necessary to investigate the cause of its occurrence.



```
[16]: notes = []
      if use_model:
          notes = generate_notes(model, seed, n_note);
      notes
      1/1 [==============================] - 0s 20ms/step
[16]: ['F4',
       'E5',
       'E5',
       'E5',
       'E5',
       'E5',
       'E5',
       'E5',
       'E5',
       'E5',
       'E5',
       'E5',
       'E5',
       'E5',
       'E5',
       'E5',
       'E5',
```

Fig47 Notes repetitively generated by the model

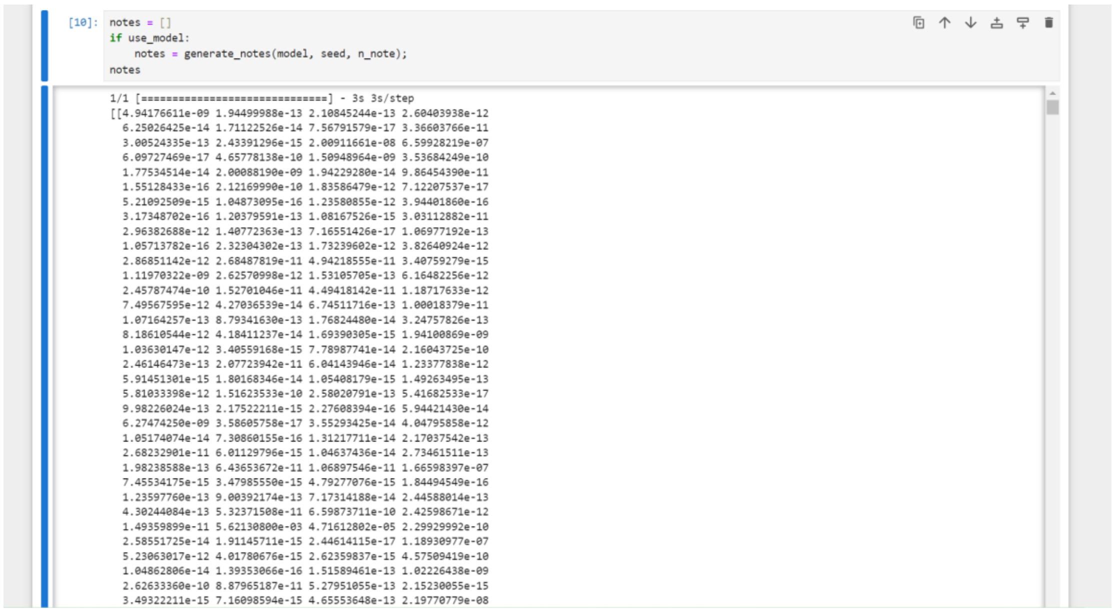
The initial hypothesis was that this occurrence might be related to the model. LSTM provides the probability for each note (result). If a particular note has an exceptionally high probability, for example, close to 1, it will result in the continuous appearance of the same note. If the model functions properly, the probabilities of predicted notes should be diverse and range between 0 and 1.
The approach to test this hypothesis was to display the probability of each note.



```python
def generate_notes(model, seed_sequence, num_notes_to_generate):
    generated_notes = []
    current_sequence = np.array(seed_sequence)
    note_occurrences = {}

    for _ in range(num_notes_to_generate):
        prediction = model.predict(current_sequence.reshape(1, length, 1))
        predicted_note_index = np.argmax(prediction)



        generated_notes.append(reverse_mapping[predicted_note_index])
        print(prediction)
```

Fig48 Print `prediction`

By printing prediction, the probabilities of each note occurring are displayed.



```
[10]: notes = []
      if use_model:
          notes = generate_notes(model, seed, n_note);
      notes
```

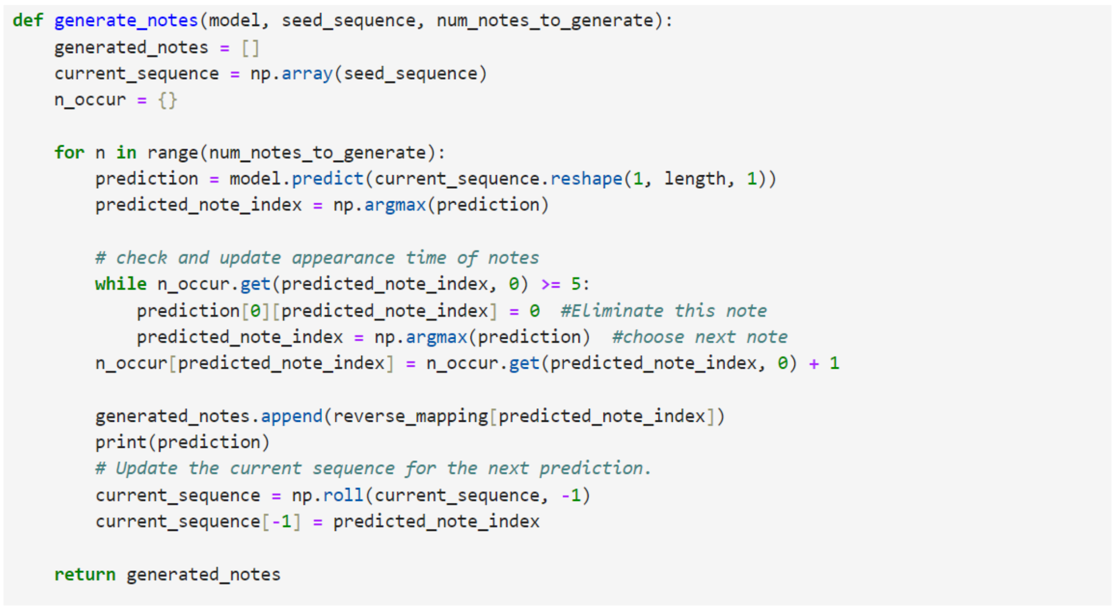
```
1/1 [==============================] - 3s 3s/step
[[4.94176611e-09 1.94499988e-13 2.10845244e-13 2.60403938e-12
  6.25026425e-14 1.71122526e-14 7.56791579e-17 3.36603766e-11
  3.00524335e-13 2.43391296e-15 2.00911661e-08 6.59928219e-07
  6.09727469e-17 4.65778138e-10 1.50948964e-09 3.53684249e-10
  1.77534514e-14 2.00088190e-09 1.94229280e-14 9.86454390e-11
  1.55128433e-16 2.12169990e-10 1.83586479e-12 7.12207537e-17
  5.21092509e-15 1.04873095e-16 1.23580855e-12 3.94401860e-16
  3.17348702e-16 1.20379591e-13 1.08167526e-15 3.03112882e-11
  2.96382688e-12 1.40772363e-13 7.16551426e-17 1.06977192e-13
  1.05713782e-16 2.32304302e-13 1.73239602e-12 3.82640924e-12
  2.86851142e-12 2.68487819e-11 4.94218555e-11 3.40759279e-15
  1.11970322e-09 2.62570998e-12 1.53105705e-13 6.16482256e-12
  2.45787474e-10 1.52701046e-11 4.49418142e-11 1.18717633e-12
  7.49567595e-12 4.27036539e-14 6.74511716e-13 1.00018379e-11
  1.07164257e-13 8.79341630e-13 1.76824480e-14 3.24757826e-13
  8.18610544e-12 4.18411237e-14 1.69390305e-15 1.94100869e-09
  1.03630147e-12 3.40559168e-15 7.78987741e-14 2.16043725e-10
  2.46146473e-13 2.07723942e-11 6.04143946e-14 1.23377838e-12
  5.91451301e-15 1.80168346e-14 1.05408179e-15 1.49263495e-13
  5.81033398e-12 1.51623533e-10 2.58020791e-13 5.41682533e-17
  9.98226024e-13 2.17522211e-15 2.27608394e-16 5.94421430e-14
  6.27474250e-09 3.58605758e-17 3.55293425e-14 4.04795858e-12
  1.05174074e-14 7.30860155e-16 1.31217711e-14 2.17037542e-13
  2.68232901e-11 6.01129796e-15 1.04637436e-14 2.73461511e-13
  1.98238588e-13 6.43653672e-11 1.06897546e-11 1.66598397e-07
  7.45534175e-15 3.47985550e-15 4.79277076e-15 1.84494549e-16
  1.23597760e-13 9.00392174e-13 7.17314188e-14 2.44588014e-13
  4.30244084e-13 5.32371508e-11 6.59873711e-10 2.42598671e-12
  1.49359899e-11 5.62130800e-03 4.71612802e-05 2.29929992e-10
  2.58551725e-14 1.91145711e-15 2.44614115e-17 1.18930977e-07
  5.23063017e-12 4.01780676e-15 2.62359837e-15 4.57509419e-10
  1.04862806e-14 1.39353066e-16 1.51589461e-13 1.02226438e-09
  2.62633360e-10 8.87965187e-11 5.27951055e-13 2.15230055e-15
  3.49322211e-15 7.16098594e-15 4.65553648e-13 2.19770779e-08
```

Fig49 The result shows the possibility of predicted notes

The possibilities of predicted notes are diverse, which means that repetitive notes are not caused by the model.

In order to prevent the repetition of notes produced by the model, post-processing rules are adopted.

The idea is to check how many times the newly predicted most likely note has occurred. If it exceeds five times, lower the possibility to prevent it from appearing in the new sequence again, and then find the following most likely note.



```python
def generate_notes(model, seed_sequence, num_notes_to_generate):
    generated_notes = []
    current_sequence = np.array(seed_sequence)
    n_occur = {}

    for n in range(num_notes_to_generate):
        prediction = model.predict(current_sequence.reshape(1, length, 1))
        predicted_note_index = np.argmax(prediction)

        # check and update appearance time of notes
        while n_occur.get(predicted_note_index, 0) >= 5:
            prediction[0][predicted_note_index] = 0  #Eliminate this note
            predicted_note_index = np.argmax(prediction)  #choose next note
        n_occur[predicted_note_index] = n_occur.get(predicted_note_index, 0) + 1

        generated_notes.append(reverse_mapping[predicted_note_index])
        print(prediction)
        # Update the current sequence for the next prediction.
        current_sequence = np.roll(current_sequence, -1)
        current_sequence[-1] = predicted_note_index

    return generated_notes
```

Fig50 Post-processing rules

In this code snippet, `prediction` is the array of probabilities for each note predicted by the model in every prediction.

In `predicted_note_index = np.argmax(prediction), np.argmax` finds the position of the highest probability in the prediction array, and then assigns this index to ` predicted_note_index `.

The purpose of following code is to avoid repetition when generating notes, especially to prevent the same note from appearing multiple times in succession. It achieves this by tracking the occurrence count of each note and selecting a different note when one appears for more than five times.

The whole piece of code can be divided into four parts.

1. First, using a `while` loop to check if current predicted note has appeared for 5 times.

**`note_occurrences.get(predicted_note_index,0)** ` is tracking the number of times each note (identified by ` **predicted_note_index** `) has occurred from the dictionary. If this note never appeared before, 0 is its defaulted number.

**`.get(predicted_note_index)**`method is a way to retrieve `predicted_note_index` which is a key in `**note_occurrence**` dictionary, and this method will return the value associated with ` **predicted_note_index** `. In this case, the value is how many times a note has appeared.

2. Second, if single note has occurred for more than 5 times, `**prediction[0][predicted_note_index] = 0**` will set the possibility as 0 to ignore it.

3. Third thing is finding next most possible note by `**predicted_note_index = np.argmax(prediction)**` after removing repeated note. Besides, updated dictionary ` **n_occur** ` by `**n_occur[predicted_note_index] = n_occur.get(predicted_note_index, 0) + 1**`

4. The final step is updating the current sequence for the next prediction.

`current_sequence = np.roll(current_sequence, -1)`:
`np.roll` is rolling elements in array along the given axis. ‘-1’ here means the first element of array will move to the last place of array.

`current_sequence[-1] = predicted_note_index`:
In python, negative indexing is informing count from the end. So ‘-1’ here indicates assigning predicted_note_index to the last place of **`current_sequence`**. We can sure the most possible and newest note is added to sequence.

8.6 Final System

8.6.1 Final system description

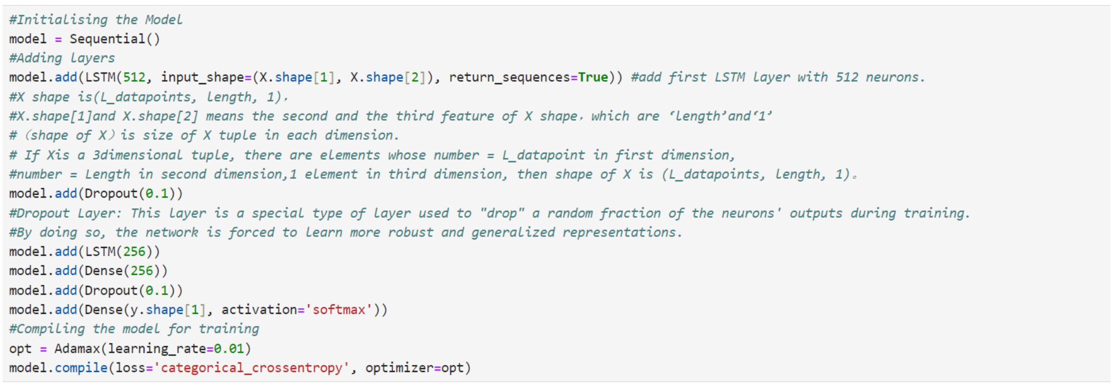
The final system can be seen in two parts: model training and real-time hand capture.
The model training part includes loading the dataset, attracting music information from the MIDI file dataset by the Music21 library, and eliminating rare notes and pre-processing notes and chords by converting them into the shape that the LSTM model expects. Work-related to note predictions is done in this part.
In the real-time hand capture, the task done here are load pre-trained model, setting camera by OpenCV library and activating Mediapipe framework for hand recognition. The captured X-coordinates of the hand will be mapped to durations according to their value. Then by the function provided by the Music21 library, durations generated by hand movement will be assigned to predicted notes. Playing notes and video rendering are two parallel programs to ensure smooth user experience.

8.6.2 Software used and Dataset

Dataset is Classical Music MIDI of Mozart.
Link to dataset is https://www.kaggle.com/datasets/soumikrakshit/classical-music-midi.
The library and software special in this project include : Music21, Mediapipe framework, OpenCV and TensorFlow. Code is written on Jupyter notebook and Vscode.
The rest of useful modules can be found in my GitHub.
Link to my GitHub: https://github.com/ZIqinGX/MSc_Advanced_project

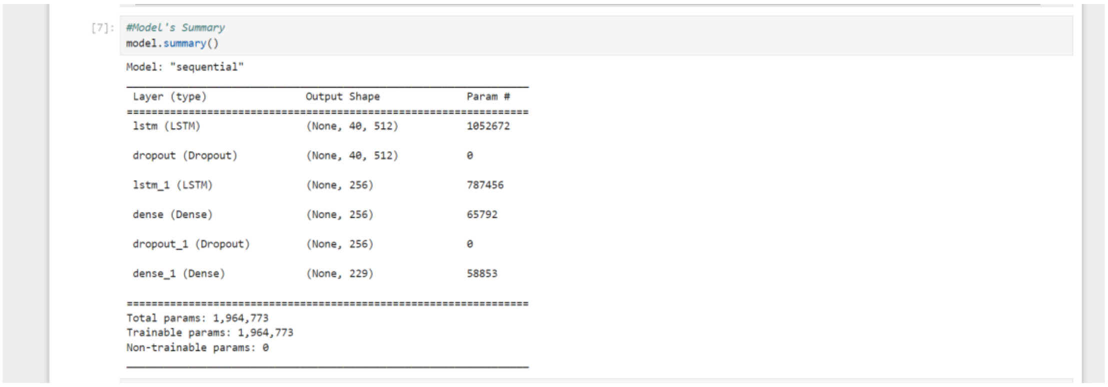
8.6.3 Model structure

Since my work is based on reflect Music Generation: LSTM created by Kapoor (2021). She provides a convenient method to predict music by LSTM model. I leant how to build model structure from her example and changed the dataset in my case.
Original source can be found https://www.kaggle.com/code/karnikakapoor/music-generation-lstm/notebook.



```
#Initialising the Model
model = Sequential()
#Adding layers
model.add(LSTM(512, input_shape=(X.shape[1], X.shape[2]), return_sequences=True)) #add first LSTM layer with 512 neurons.
#X shape is(L_datapoints, length, 1)，
#X.shape[1]and X.shape[2] means the second and the third feature of X shape，which are ‘length’and‘1’
# （shape of X）is size of X tuple in each dimension.
# If Xis a 3dimensional tuple, there are elements whose number = L_datapoint in first dimension,
#number = Length in second dimension,1 element in third dimension, then shape of X is (L_datapoints, length, 1)。
model.add(Dropout(0.1))
#Dropout Layer: This layer is a special type of layer used to "drop" a random fraction of the neurons' outputs during training.
#By doing so, the network is forced to learn more robust and generalized representations.
model.add(LSTM(256))
model.add(Dense(256))
model.add(Dropout(0.1))
model.add(Dense(y.shape[1], activation='softmax'))
#Compiling the model for training
opt = Adamax(learning_rate=0.01)
model.compile(loss='categorical_crossentropy', optimizer=opt)
```

Fig51 model building



```
[7]: #Model's Summary
     model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 40, 512)           1052672

 dropout (Dropout)           (None, 40, 512)           0

 lstm_1 (LSTM)               (None, 256)               787456

 dense (Dense)               (None, 256)               65792

 dropout_1 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 229)               58853

=================================================================
Total params: 1,964,773
Trainable params: 1,964,773
Non-trainable params: 0
_________________________________________________________________
```

Fig52 Model summary

The LSTM model used in this project has six layers. The first layer is a LSTM layer with 512 neurons. The output shape is (None,40,512), which indicates the length of every sequence for input is 40 times long. Next to the LSTM layer is a dropout layer where the dropout rate is set to be 0.1. It means 10% of the output from neurons will be set to 0 in each training epoch to prevent overfitting. The second LSTM layer has 256 neurons. Following the second LSTM layer is a dense layer with 256 units. Another dropout layer follows with a 0.1 dropout rate, providing further regularization. The final layer is a dense layer with the same number as the number of classes in the output. Activation of the final dense layer is a softmax function as it gives probability distribution over the classes.

8.6.4 Training

The final work has been trained for 200 times by GPU.

8.6.5 User interface

To use this project, first choose a dataset and preprocess them. The user opens the Jupyter Notebook with dataset loading. The action in this Jupyter notebook includes eliminating notes which appear very rarely and converting the dataset from MIDI into the proper format for machine learning model to process. Then open the Jupyter notebook with the model building title and run the code there. The training epoch can be adjusted. After training enough times, save the LSTM model.

After preparing work is done, open the file named `3_real_time_parallel.py`. It is where video capture and music playing will happen. The operator chooses `usemodel =true` to adopt the pre-trained model. When the video capture window is invoked, the participant can start to move the hand and make sure the hand is within the frame. After all work is done correctly, notes will be played.

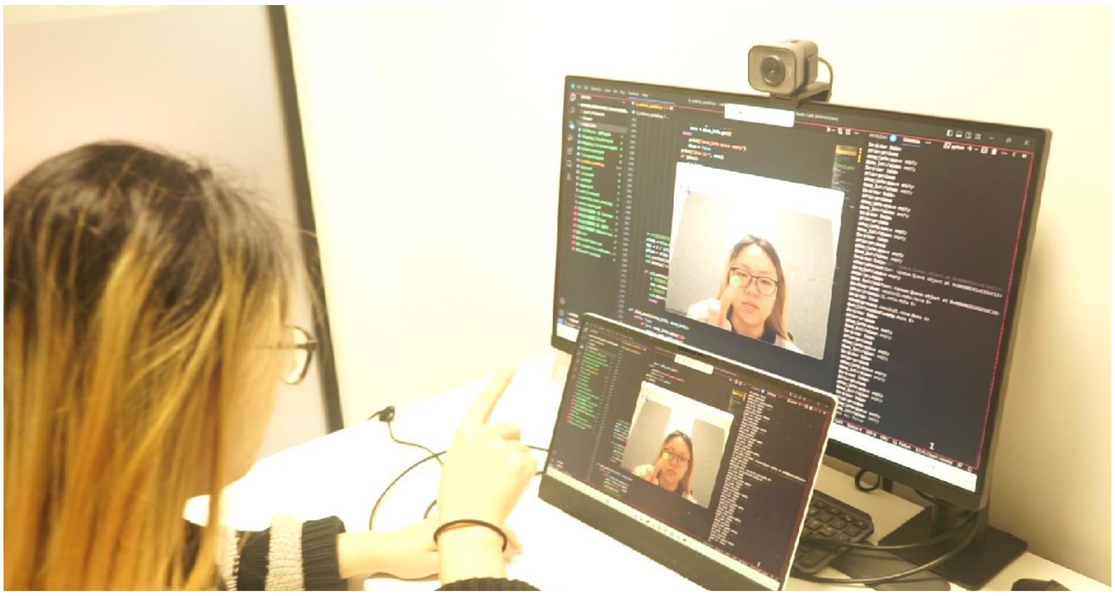


Fig53 Demonstration of Conductor project

Video link**:** https://youtu.be/C4xpgRHIve8

9.Reflection

My reflection will go on by comparing with related work and analyzing the complete extent of three objects mentioned in methodology.

From Radio Baton generated by Dr. Max Mathews, the design of real-time human and machine leaning model co-creation system draws inspiration from treating the user as a conductor rather than a performer. It allows them to focus on controlling the duration of the music. Unlike the Radio Baton, which uses a radio baton to control various music factors, in my project, it is my hand that decides the duration. Radio Baton can control 6 variables, however, in my case, it only achieves control one parameter, which causes a reduction in flexibility. Compared with Radio Baton, there is also positive aspects about my design, that is using camera to track hand movement can save time and resource of making electronic device. it is useful and convenient for quick development. Another difference lies in the application of the machine learning model. In Radio Baton, the musical scores are programmed, however, in my case, predicted notes can be various by changing the dataset, adjusting training epoch, or replacing LSTM by different model to alter the predefined music. There is more space for exploration and potential in musical creativity.

Influenced by the related work IMPS, using machine learning model into music generation is considered. For the reason that these two projects faces different user, the focus between two system is different. IMPS allows AI music generation system to predict interaction between operator and this generation system can take charge of performance when musician stop playing. However, the conductor system presented in this article can not leave user and it predict notes rather than operations.

Compared with applying LSTM model in generating music by Kapoor, final system is interactive, and duration of predicted notes can be manipulated by change of hand position. Besides, real-time system executes tasks in parallel not only provide action-production loop and more possibility to adjust music, but also means smoother using experience. As for the satisfaction of initial inspiration of constructing this project, it is friendly to user without music background to produce music.

One of the advantages is that users can hear in real-time the effect of the duration mapped from their hand gestures being assigned to a note, providing immediate feedback, and allowing for more intuitive control over the subsequent rhythmic changes. This avoids the issue of meaningless music rhythm generation encountered in the previous iteration.

Another advantage is in this generation, the code for predicting notes by the model and the code for real-time capture mapping to music duration and synthesizing music are separated, thus, by pre-training and saving the predicted notes from the model, all tasks can be completed within the same program without the need for repeatedly clicking through several others.

The third advantage worth to mention is the program has been improved from serial processing to synchronous execution, making the synthesis of music smoother.

To evaluate from three objectives: 1) The generated notes can be manipulated, 2) The generated musical rhythm had variations and dynamics, 3) Completing the task of computer-generated note prediction and interacting with humans. The final system completes these three objectives. But it still has a lot of potential to develop.

For the objective 1, the system so far can only adjust duration, not pitch, rhythm. User cannot do enough change on notes because generated of note now are totally dependent on LSTM model.

For the objective 2, what has been built produce a series of single note rather than consistent and fluent music clip. Another shortcoming is current program still struggles to effectively handle and play chords. Now all chords predicted is dealt by choosing the first note in chord to present the whole chord, which is in the lower level of expected.

For the objective 3, the interactive method with human is camera capturing, which is very limited. And in this project, only LSTM model is tested. There is lacking comparing with the performance of other relevant machine learning model.

Another shortcoming is repetitive occurrence of note in the predictions. Post-process rules have been introduced to solve this problem by now, the cause of this issue still remain not truly identified. This system was built based on reflecting related work called 'Music Generation: LSTM', so the structure of the model is consistent with this example without much exploration. In the future, attempts should be made to alter the machine learning model structure and datasets process to see if the issue of note repetition can be genuinely resolved.

In the future, different training epochs should be tested and operations that can be performed on notes should be more diverse, such as changing the pitch. Additionally, more methods will be explored to generate continuous music, rather than separate, isolated notes.

For more music created by the author：

https://github.com/ZIqinGX/MSc_Advanced_project/blob/main/More_music_pieces_developed/Link_to_Youtube.md

Reference

Bonnici, A., Dannenberg, R.B., Kemper, S. & Camilleri, K.P. 2021, "Editorial: Music and AI", *Frontiers in artificial intelligence,* vol. 4, pp. 651446.

Elman, J.L. (1990) ‘Finding Structure in Time’, *Cognitive Science*, 14(2), pp. 179–211. Available at: https://doi.org/10.1207/s15516709cog1402_1.

Hand landmarks detection guide | MediaPipe (2023) *Google for Developers*. Available at: https://developers.google.com/mediapipe/solutions/vision/hand_landmarker (Accessed: 22 November 2023).

Hewamalage, H., Bergmeir, C. and Bandara, K. (2021) ‘Recurrent Neural Networks for Time Series Forecasting: Current status and future directions’, *International Journal of Forecasting*, 37(1), pp. 388–427. Available at: https://doi.org/10.1016/j.ijforecast.2020.06.008.

Kapoor, K. (2021) *Music Generation: LSTM | Kaggle*. Available at: https://www.kaggle.com/code/karnikakapoor/music-generation-lstm/notebook (Accessed: 22 November 2023).

Kylafi, C.E. & Ioannidis, Y. 2020, "iJazzARTIST: Intelligent Jazz Accompanist for Real-Time human-computer Interactive muSic improvisaTion", .

Martin, C.P. 2022, "Performing with a Generative Electronic Music Controller", *Joint proceedings of the ACM IUI Workshops*.

Martin, C.P. and Torresen, J. (2019) ‘An Interactive Musical Prediction System with Mixture Density Recurrent Neural Networks’. arXiv. Available at: http://arxiv.org/abs/1904.05009 (Accessed: 21 November 2023).

Max Mathews Radio Baton Demonstration (2010). Available at: https://www.youtube.com/watch?v=3ZOzUVD4oLg (Accessed: 20 November 2023).

Max Mathews Radio Baton controller and the conductor program demonstration (2010). Center for Computer Research in Music and Acoustics, Stanford University, Stanford, California: Computer History Museum (Oral history collection).

McCormack, J., Hutchings, P., Gifford, T., Yee-King, M., Llano, M.T. & D’inverno, M. 2020, "Design Considerations for Real-Time Collaboration with Creative Artificial Intelligence", *Organised sound : an international journal of music technology,* vol. 25, no. 1, pp. 41-52.

【MediaPipe】(1) AI Visual, hand key-point real-time tracking, CSDN blog (2022). Available at: https://blog.csdn.net/dgvv4/article/details/122023047 (Accessed: 22 November 2023).

Sak, H., Senior, A. and Beaufays, F. (2014) ‘Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition’. arXiv. Available at: http://arxiv.org/abs/1402.1128 (Accessed: 22 November 2023).

Sherstinsky, A. (2020) ‘Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network’, *Physica D: Nonlinear Phenomena*, 404, p. 132306. Available at: https://doi.org/10.1016/j.physd.2019.132306.

Understanding LSTM Networks -- colah’s blog (2015). Available at: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (Accessed: 22 November 2023).