Music Analysis using Computational Methods

A Dissertion

submitted in Partial Fulfilment of the Requirements for the Award of Degree of

INTEGRATED MASTER OF SCIENCE

In

PHYSICS

(Specialization in Computational Physics)

Submitted By

Shashank Sinha

Roll no.- 1810002, Enrolment no. – 180222

Under the supervision of

Dr. Anurag Sahay

Assistant Professor

Department Of Physics

NATIONAL INSTITUTE OF TECHNOLOGY PATNA

May 2023 (2022-23)

CERTIFICATE

The undersigned certifies that **Mr. SHASHANK SINHA**, **Roll No. (1810002)**, **Enrolment No. (180222)** is registered for the Five-Year Integrated Master's Program in the Department of Physics.

I hereby recommend that the Dissertation entitled, **MUSIC ANALYSIS USING COMPUTATIONAL METHODS** be accepted as the partial fulfilment of the requirements for evaluation and award of the five-year Integrated M.Sc. Degree in Physics.

Date……………………….

DECLARATION AND COPYRIGHT TRANSFER

I ……………………………… Roll No …………….. Enrolment No …………………. a registered candidate for Postgraduate Programme (Integrated M.Sc. – Physics) under department of Physics of National Institute of Technology Patna, declare that this is my own original work and does not contain material for which the copyright belongs to a third party and that it has not been presented and will not be presented to any other University/ Institute for a similar or any other Degree award. I further confirm that for all third-party copyright material in my thesis/ dissertation (including any electronic attachments) is "blanked out" third party material from the copies of the thesis/dissertation/book/articles etc.; fully referenced the deleted materials and where possible, provided links (url) to electronic sources of the material. I hereby transfer exclusive copyright for this thesis to NIT Patna. The following rights are reserved by the author: a) The right to use, free of charge, all or part of this article in future work of their own, such as books and lectures, giving reference to the original place of publication and copyright holding. b) The right to reproduce the article or thesis for their own purpose provided the copies are not offered for sale.

ACKNOWLEDGEMENT

I would like to express my gratitude to my supervisor **Dr. Anurag Sahay** for his guidance and supervision in carrying out this project. He has always encouraged me to do something innovative and go beyond the limits. Without his constant motivation, completing this thesis was impossible.

I also express my thanks to my lab mates and classmates for their help during this project. I would also like to sincerely thank **Dr. D.K. Mahto, HOD, Dept. of Physics, NIT Patna** and all faculty members of the Physics Department.

Finally, I would also like to thank my **Parents and Friends** who supported me and helped me a lot in finalizing and completing this project within the limited period.

Thanking You

SHASHANK SINHA

Contents

Abstract

Music is nothing but a sequence of events, whether it is a drum roll or guitar riff or even piano solo, all are basically a sequence of music elements. And the thing our ears like is the correlation between music elements, which comes after each other. For example, a note or chord can create a dissonance, and what follows it is a resolving music element, which resolves that dissonance. And it is this correlation between music elements, which we can use to our advantage in working on music with algorithms as our tools.

Music generation can be formulated as a next music element prediction problem, which would allow us to generate as much music as we wanted by just keep passing the previously generated music element.

For implementation, used GRU (Gated Recurrent Unit) in-place of the vanilla RNN (Recurrent Neural Network), because of its long term dependencies retaining ability, which is much needed in music dataset. Each GRU takes previous layer's output as input and activation, and the output would be the next music element.

Introduction

Machine learning is now being used for many interesting applications in a variety of fields. Music generation is one of the interesting applications of machine learning. As music itself is sequential data, it can be modelled using a sequential machine learning model such as the recurrent neural network.

About the Dataset

- A music element (data-point) is described in terms of following parts :
  1. *Pitch* : The measure of frequency of the sound. It is a theoretical term, but in music, it is specified in terms of notes and chords, along with their octaves.
  2. *Notes* : This represents relatively specified frequencies, which are distributed within an octave. There are 12 different note-names in music, namely C, C# or Db (called as ‘C-sharp’, or ‘D-flat’), D, D#, E, F, F#, G, G#, A, A#, and B.
  3. *Chords* : This represents a group of notes, in which basically frequencies of constituent notes overlap, and a beam of frequencies is what we listen. For example: F-major (F-note + A-note + C-note).
  4. *Octave* : These are specified frequency-ranges, which periodically repeats after spanning a certain frequency range. The higher the octave number, the higher is its frequency range, i.e. 5th octave has higher frequency than that of 4th octave. Taking a note as example, if we try to read music-files as text like “C4”, this denotes C-note of 4th octave.

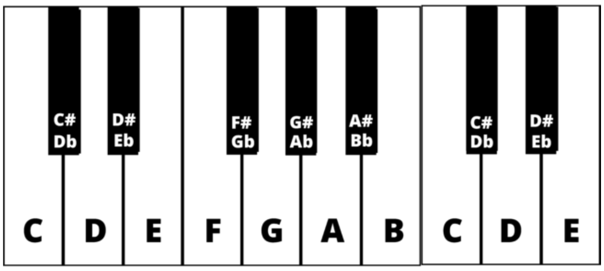


Figure 1: Music Notes (can be better visualized as Piano-keys)

5. *Offset* : It is the instance(timing of occurence) of a music-element. The rhythm of music depends on this, i.e. the timing-gap of 2 music elements.
6. *Instrument* : The instrument on which the music element is played. This alters the timbre of that music element.

- All the audio files are .midi files, collecting only piano music. Chosen such dataset, because it's easy to retrieve data from this kind of music.

Algorithms used (in Neural Network) :

1) *LSTM Layer*

- *Long Short Term Memory* (usually called LSTMs).
- LSTM is a special kind of *RNN (Recurrent Neural Network)*. In RNN, same weights & activation function are used in each iteration to predict next future value.
- In RNN :

$$a_1 = A[(W_{xa})*(x_1) + (W_{aa})*(a_0) + b_a]$$
$$y_1 = A'[(W_{ay})*(a_1) + b_y]$$
$$\ldots\ldots.$$
$$a_n = A[(W_{xa})*(x_n) + (W_{aa})*(a_{n-1}) + b_a]$$
$$y_n = A'[(W_{ay})*(a_n) + b_y]$$

- In calculation of a_n, gradient corresponding to a_0 will be n-times product of (W_{aa}). So if :

W_{aa} > 1 : *Exploding Gradient problem* will emerge. The resulting gradient will be extremely large, & will negatively effect predictions. This problem is solvable by implementing some limitations on each gradient.

W_{aa} < 1 : *Vanishing Gradient problem* will emerge. The resulting gradient will be almost 0, i.e. no effect of old outputs on new predictions. This is also a negative effect, and solving this is a relatively difficult task.

- For solving Vanishing Gradient problem, a different Neural network unit comes in use, i.e. *Gated Recurrent Unit (GRU)*. In this unit, a potential value for prediction is calculated & a sigmoid update-function is there to decide whether value of a_{i-1} is assigned to a_i , or the calculated potential value. There is also a *Reset Gate*, which decides how much is (a_{i-1}) contributing to the potential value. Following are the equations :

  ➔Reset Gate :

  R_{gate} = sigma[(W_r)*(x_i) + (W'_r)*(a_{i-1}) + b_3]

  ➔Potential value :

  $a_{potential\_i}$ = tanh[(W_{aa})*(a_{i-1})*(R_{gate}) + (W_{xa})*(x_i) + b_1]

  ➔Sigmoid update-function :

  u = sigma[(W'_a)*(a_{i-1}) + (W'_x)*(x_i) + b_2]

  ➔Decider equation :

  a_i = (u)*($a_{potential\_i}$) + (1-u)*(a_{i-1})

- In LSTM, the algorithm goes one-step beyond. As there is a Reset Gate in GRU, there are 3-gates in LSTM :- (equation of all gates are similar to that of reset gate, i.e. all are sigma function)

  i.) *Input Gate(in)* -- tells what will be the contribution of potential value (c_{t_cap}) in prediction(c_t) .

  ii.) *Forget Gate(f)* – tells what will be the contribution of past value (c_{t-1}) in prediction(c_t) .

iii.) *Output Gate(g_{out})* – calculates activation function for next layer of LSTM.

➔ Prediction :

$c_t = (in)*(c_{t\_cap}) + (f)*(c_{t-1})$

- LSTM is more complex algorithm than RNN & GRU. As they also store more memory for future predictions. They store – 3 gates, many weights, past value(c_{t-1}), and past activation function(a_{t-1}).
- LSTM(units, input_shape, return_sequences)
  Above is the syntax of LSTM layer, where :
  units = dimension of output of this layer, input_shape = dimension of input to this layer. And, return_sequences = boolean value. If true, it returns only the last output of output sequence, else returns the full output sequence.

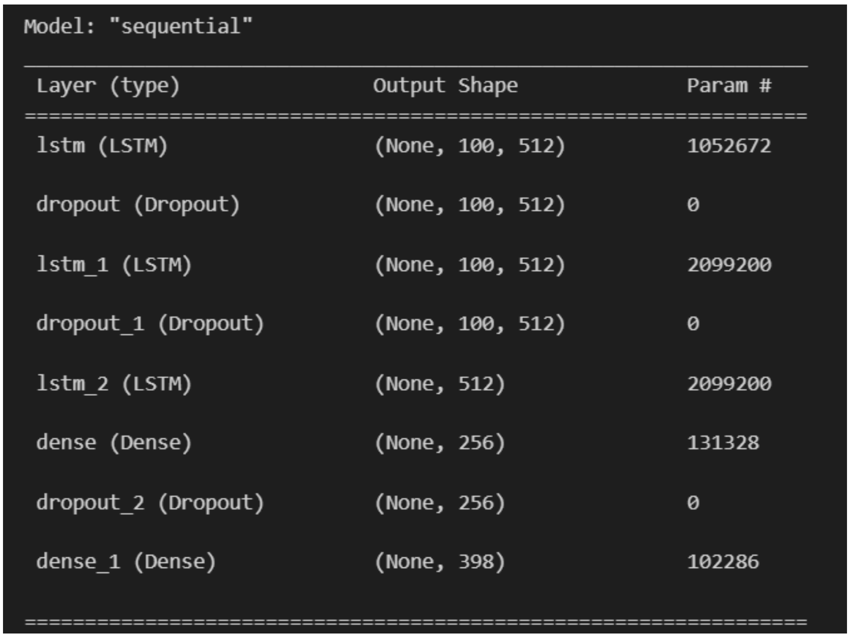
2) Dropout Layer

- Dropout refers to data, or noise, that is dropped intentionally from a neural network to improve processing, to reach to better results.
- Similar to human brain, the neural-network units randomly process countless inputs, and give countless outputs at the same time. So, it might happen that an intermediate output thing gets passed to another neural-network unit even before the end output gets calculated. And, some of these processes result in noise creation.
- Dropout(x) : The syntax of this argument, where x refers to the fraction of previous layer's output to drop. So, value of x lies between 0 and 1.

3) ***Dense Layer***

- Dense layer has a deep connection in neural network, i.e. each neuron in dense layer is receiving inputs from all previous layer neurons. So, it helps in developing a better correlation in neural network.
- This layer performs a matrix-vector multiplication, where values used as correlation-factors are trained with the help of backpropagation.
- Dense(units, activation = <*activation-function*>).

  Above is the syntax of this layer where, units = output-size of this layer. And activation function helps to develop output though complex functions, based on input. By default, it uses linear function (i.e. linear regression model).
- But in final layer of neural network, prediction is needed in terms of probability-distribution, so used “softmax” activation function. In other layers, “linear” activation function is used, as simple linear regression model is needed to predict next note.

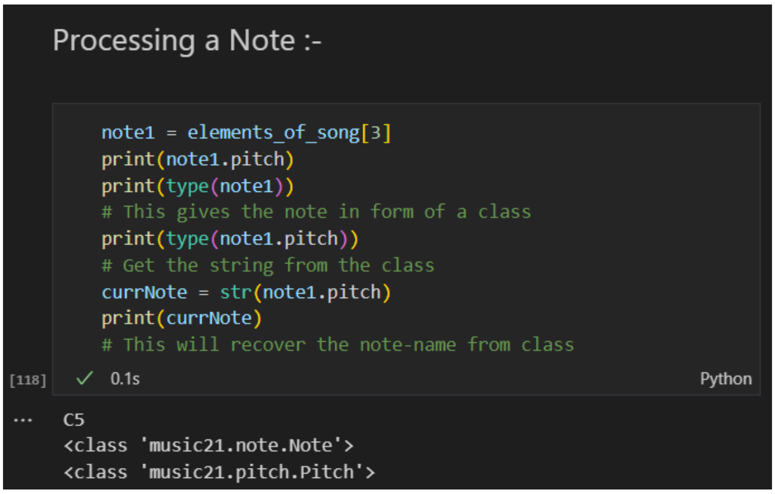


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 100, 512)          1052672

 dropout (Dropout)           (None, 100, 512)          0

 lstm_1 (LSTM)               (None, 100, 512)          2099200

 dropout_1 (Dropout)         (None, 100, 512)          0

 lstm_2 (LSTM)               (None, 512)               2099200

 dense (Dense)               (None, 256)               131328

 dropout_2 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 398)               102286

=================================================================
```

Figure 2 : The Neural Network model (using these layers)

Data Processing :

- In the dataset, there is only 2 types of music elements considered, i.e. notes and chords.
- In case of note, extracted the pitch of music element and converted it to string. And then further mapped the string with an integer, using hashmap data structure.

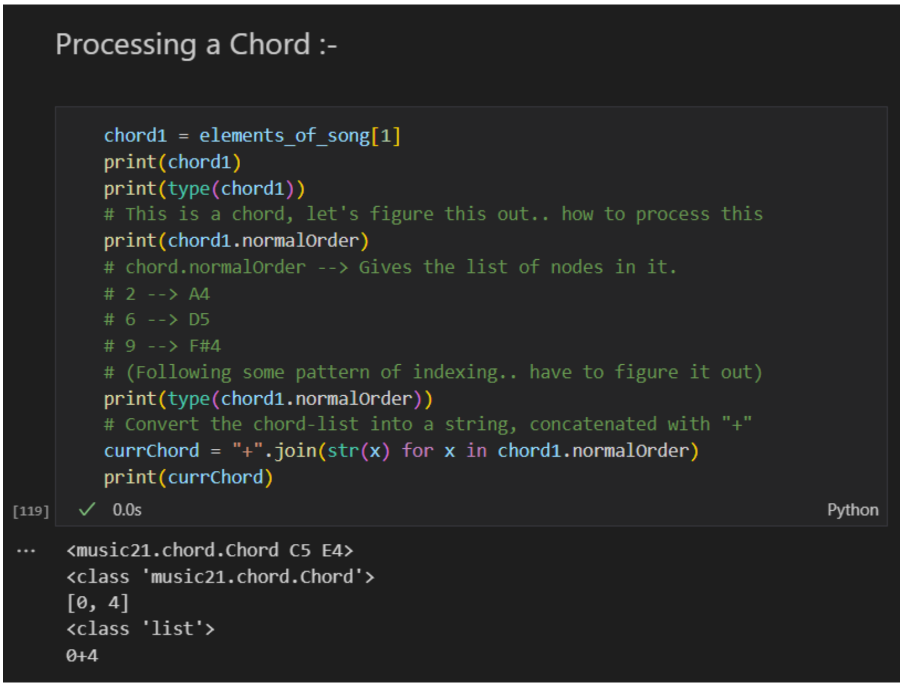


Processing a Note :-

```python
note1 = elements_of_song[3]
print(note1.pitch)
print(type(note1))
# This gives the note in form of a class
print(type(note1.pitch))
# Get the string from the class
currNote = str(note1.pitch)
print(currNote)
# This will recover the note-name from class
```

```
C5
<class 'music21.note.Note'>
<class 'music21.pitch.Pitch'>
```

- In case of chord, extracted the constituent notes using “chord.normalOrder” command, which will give an array of random integers corresponding to notes, and converted it to string similar to notes. So that, it can be mapped in the same way as notes, so that all music elements can be treated similarly.

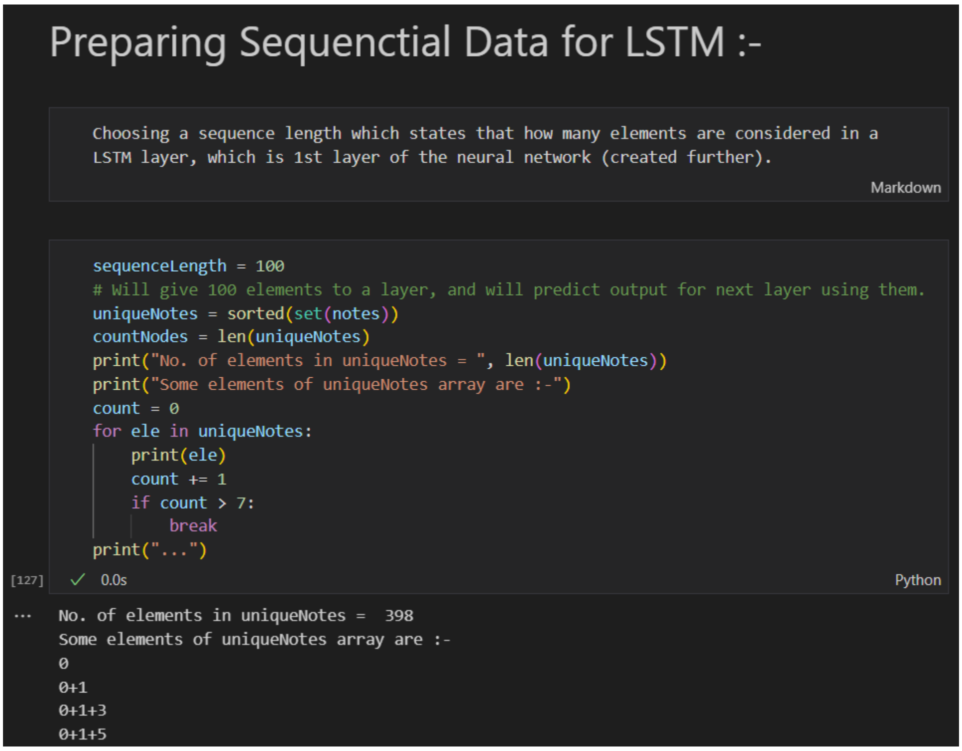


Processing a Chord :-

```python
chord1 = elements_of_song[1]
print(chord1)
print(type(chord1))
# This is a chord, let's figure this out.. how to process this
print(chord1.normalOrder)
# chord.normalOrder --> Gives the list of nodes in it.
# 2 --> A4
# 6 --> D5
# 9 --> F#4
# (Following some pattern of indexing.. have to figure it out)
print(type(chord1.normalOrder))
# Convert the chord-list into a string, concatenated with "+"
currChord = "+".join(str(x) for x in chord1.normalOrder)
print(currChord)
```

```
<music21.chord.Chord C5 E4>
<class 'music21.chord.Chord'>
[0, 4]
<class 'list'>
0+4
```

- After retrieving all music elements from all the music files, created an array comprising of only unique music elements from the collection. And then mapped those elements with their index. This mapping was required because we have some random collection of strings, but the LSTM model works on numerical data, so we can convert our data to numerical using this map.



Preparing Sequenctial Data for LSTM :-

Choosing a sequence length which states that how many elements are considered in a LSTM layer, which is 1st layer of the neural network (created further).

```python
sequenceLength = 100
# Will give 100 elements to a layer, and will predict output for next layer using them.
uniqueNotes = sorted(set(notes))
countNodes = len(uniqueNotes)
print("No. of elements in uniqueNotes = ", len(uniqueNotes))
print("Some elements of uniqueNotes array are :-")
count = 0
for ele in uniqueNotes:
    print(ele)
    count += 1
    if count > 7:
        break
print("...")
```

```
No. of elements in uniqueNotes =  398
Some elements of uniqueNotes array are :-
0
0+1
0+1+3
0+1+5
```

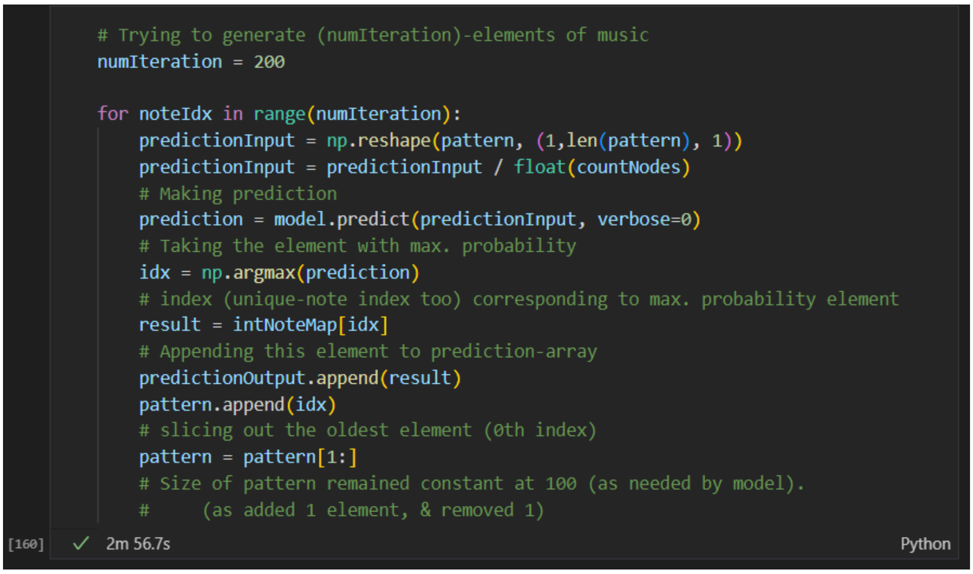
- Then normalized the input-data, i.e. shrinked the values of all the datapoints betweens 0 and 1. Initially they were the indices of the unique-element array, so the values were from 0 to size of array (only integer values). And for algorithms, there were many datapoints possible in between them, as they work in high precision. So, normalization will help in reducing additional noise, produced by varied range of data points, i.e. data having high variance.

Making Prediction :

- As the sequenceLength was specified as 100, so the dimension of the input array must be 100 i.e., we have to give 100 music elements as input.
- For each element of input, first calculated it’s normalized value, and then made prediction using the trained sequential model.
- The sequential model will predict the probability distribution of all unique elements as softmax activation function is used in the

last dense layer. And from that array, the element with maximum probability distribution is considered as the next predicted music element.

- In each step, we will keep incrementing the index with maximum probability value to the input array, which will increase the size to 101. And to check down the size, will keep discarding the oldest music element, i.e. at index 0. As the correlation of newly predicted element or upcoming predictions with the oldest element will be the lowest.
- Now, the music element corresponding to the index having maximum probability is retrieved using reverse mapping, which was created using the same unique notes array, which was used to create unique music element strings to integer mapping.



```python
# Trying to generate (numIteration)-elements of music
numIteration = 200

for noteIdx in range(numIteration):
    predictionInput = np.reshape(pattern, (1,len(pattern), 1))
    predictionInput = predictionInput / float(countNodes)
    # Making prediction
    prediction = model.predict(predictionInput, verbose=0)
    # Taking the element with max. probability
    idx = np.argmax(prediction)
    # index (unique-note index too) corresponding to max. probability element
    result = intNoteMap[idx]
    # Appending this element to prediction-array
    predictionOutput.append(result)
    pattern.append(idx)
    # slicing out the oldest element (0th index)
    pattern = pattern[1:]
    # Size of pattern remained constant at 100 (as needed by model).
    #     (as added 1 element, & removed 1)
```

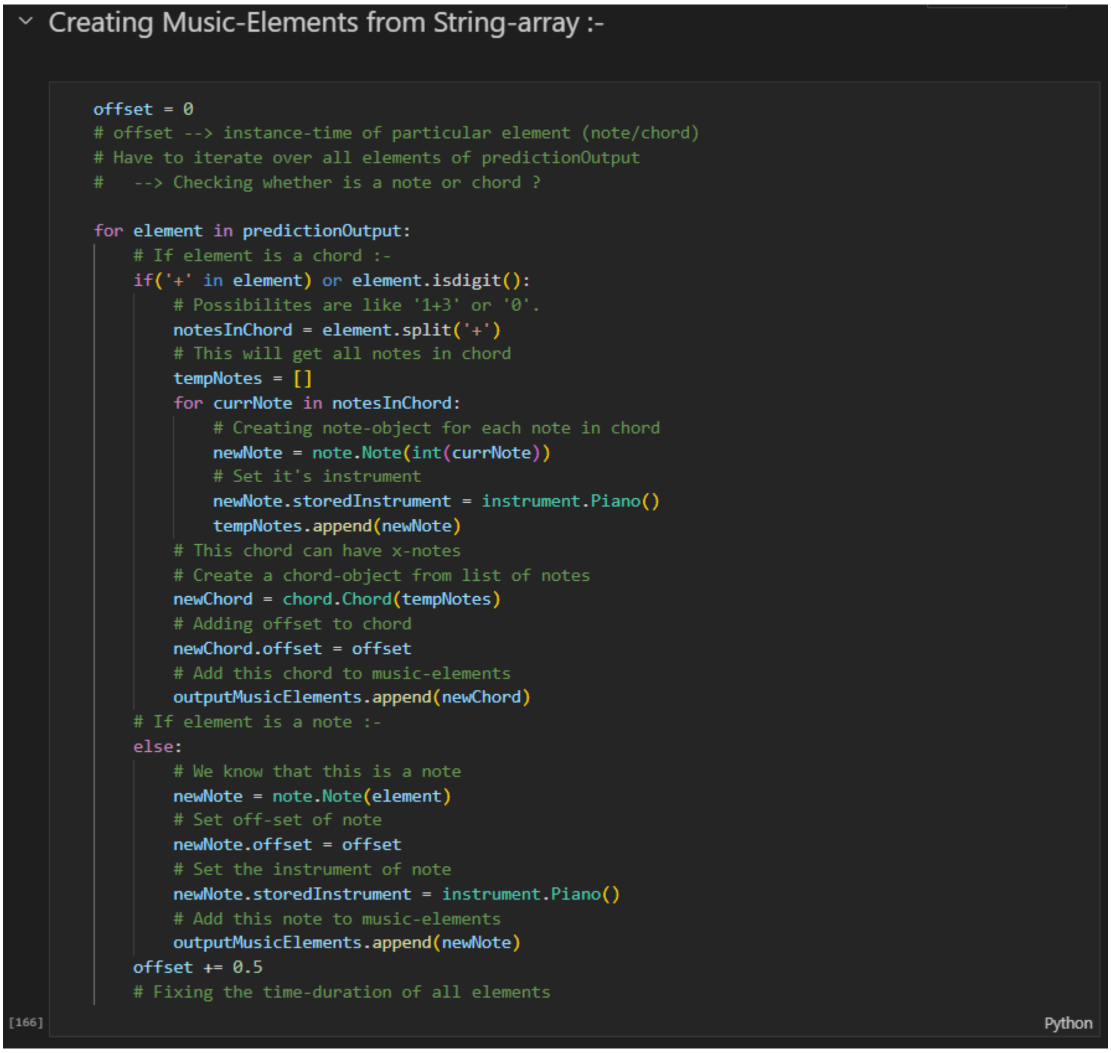
Figure 3 : Prediction using model

Generating Music out of Predicted-data :

- After revere mapping, we have elements in terms of strings. As the dataset is considered, there can be two possibilities i.e., notes and chords.
- For discriminating notes and chords, the character ‘+’ was added in chords for concatenating the integers corresponding

to constituent notes. And then processing of constituent notes and the note elements are similar.

- For generating the notes, used the Note() method of note class of music21 library of python, which is basically reverse generating the note using which we generated the integer using the same method of same class.
- Also setting the instrument of each note, as it might be storing a garbage value by default. As the input dataset was only of piano, so manually setting instrument of all the notes as piano. Also setting the offset manually, which is the instance of that music element in the audio file’s duration.
- As we are setting the offset manually, therefore the output music will be mostly sound flat seeing the rhythm. To make better music prediction, we can store the offset of input music elements in another array, and also the instrument of different music elements in another array, and similarly other factors. If these factors are considered, we can replicate the rhythm and music arrangement patterns of input music.



Creating Music-Elements from String-array :-

```python
offset = 0
# offset --> instance-time of particular element (note/chord)
# Have to iterate over all elements of predictionOutput
#   --> Checking whether is a note or chord ?

for element in predictionOutput:
    # If element is a chord :-
    if('+' in element) or element.isdigit():
        # Possibilites are like '1+3' or '0'.
        notesInChord = element.split('+')
        # This will get all notes in chord
        tempNotes = []
        for currNote in notesInChord:
            # Creating note-object for each note in chord
            newNote = note.Note(int(currNote))
            # Set it's instrument
            newNote.storedInstrument = instrument.Piano()
            tempNotes.append(newNote)
        # This chord can have x-notes
        # Create a chord-object from list of notes
        newChord = chord.Chord(tempNotes)
        # Adding offset to chord
        newChord.offset = offset
        # Add this chord to music-elements
        outputMusicElements.append(newChord)
    # If element is a note :-
    else:
        # We know that this is a note
        newNote = note.Note(element)
        # Set off-set of note
        newNote.offset = offset
        # Set the instrument of note
        newNote.storedInstrument = instrument.Piano()
        # Add this note to music-elements
        outputMusicElements.append(newNote)
    offset += 0.5
    # Fixing the time-duration of all elements
```

Figure 4 : Working code of music generation from predicted data

- From the music elements, combined them with an offset in between, and made out a midi file out of it. To visualize the input and output music out of the neural network model, plotted it side by side to each other. Where the offset of output music starts from where the input music ends.

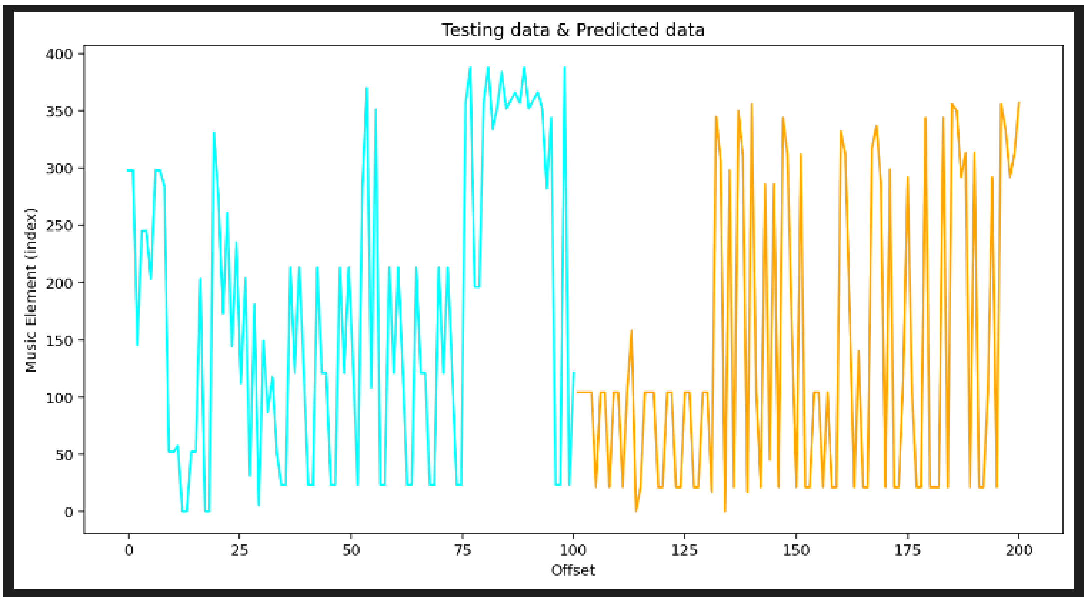


Figure 5 : Input music VS Output music

Conclusions :-

The predicted music follows some correlation trends of input music, but at the same time in most cases they are far from ideal music sounds. This is because we have done this only for one instrument, and for multiple instruments, the combination count increases exponentially. So, it is difficult to accumulate such cases, to predict a practical sounding music, specifically the training of neural network model, which will be the biggest task to handle. Also the offset is set manually, so every note of same length doesn't sound musically good for most of the times.

References :-

http://www.piano-midi.de/midi_files.htm (for midi files dataset)

https://abc.sourceforge.net/NMD/ (for midi files dataset)

Full Code :-