MUSIC GENERATION USING DEEP LEARNING

A Project Report Submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

By

B SHISHIR (190330030)

G INDRA ANURAAG (190330069)

M ADITHYA YADAV(190330144)

NEERAJ K(190330169)

DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING
K L DEEMED TO BE UNIVERSITY
AZIZNAGAR, MOINABAD , HYDERABAD-500 075

MARCH 2023

BONAFIDE CERTIFICATE

This is to certify that the project titled **MUSIC GENERATION USING DEEP LEARNING** is a bonafide record of the work done by

B SHISHIR (190330030)

G INDRA ANURAAG (190330069)

M ADITHYA YADAV(190330144)

NEERAJ K(190330169)

in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **COMPUTER SCIENCE AND ENGINEERING** of the **K L DEEMED TO BE UNIVERSITY, AZIZNAGAR, MOINABAD , HYDERABAD-500 075**, during the year 2022-2023.

Vijay Kumar Damera	**Dr.Babu Rao.K**
Project Guide	Head of the Department

Project Viva-voce held on _____

Internal Examiner	**External Examiner**

ABSTRACT

This report explores current solutions to the task of music generation.
Our aim was to replicate the algorithms mentioned and train the models on Bach classical music in order to observe the training process. Besides successfully generating fairly realistic music, this project has also shown the limitations of recurrent neural networks when dealing with a high number of classes in the context of music generation and the importance of data representation for the same dataset.

ACKNOWLEDGEMENT

We would like to thank the following people for their support and guidance without whom the completion of this project in fruition would not be possible.

Vijay Kumar Damera, our project guide, for helping us and guiding us in the course of this project .

Dr.Babu Rao.K, the Head of the Department, Department of DEPARTMENT NAME.

Our internal reviewers, **Mrs.Anuradha Nandula** , **Dr.Babu Rao.K** for their insight and advice provided during the review sessions.

We would also like to thank our individual parents and friends for their constant support.

TABLE OF CONTENTS

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background of the Project

Music generation using deep learning techniques has been a topic of interest for the past two decades. Music proves to be a different challenge compared to images, among three main dimensions: Firstly, music is temporal, with a hierarchical structure with dependencies across time. Secondly, music consists of multiple instruments that are interdependent and unfold across time. Thirdly, music is grouped into chords, arpeggios and melodies – hence each time-step may have multiple outputs. A few of the characteristics of audio data resemble some of the traditional topics of deep learning research. Recurrent Neural Networks can be used for NLP, which is similar to how the music is consecutive. Convolutional Neural Networks can be used to create many ”channels” of audio (in terms of tones and instruments) that are similar to images. Deep generative models are an interesting new field of study that has the ability to produce synthetic data that is realistic. The ability to generate music using machines, can also be applied to various other applications. For instance, it can be used along with computer vision for various use cases.

1.2 Problem Statement

AI has been widely applied to the domain of Arts. There are many Deep Learning models that can generate Paintings, Sketches, Stories, Music etc. In this project, we aim to develop a deep learning model that successfully generates music. Through this

project, it enables users with minimal knowledge of music theory to get into music composition and can also assist in giving new ideas.

1.3 Objectives

Our main objective is to develop a deep learning model that can generate short musical melodies with minimal human intervention.

1.4 Scope of the Project

This project is used to generate short musical melodies which can help people in various domains. It can help an artist to compose a musical piece , gives a lot of variety of ideas for composing a song. It can help in understanding the music generated by different musical instruments and also serves as a platform for common people to get into music. The project is a trained deep learning model that generates music.

Chapter 2

Literature Review

2.1 Overview of related works

The following section will contain some important and necessary background to understand the design and working of the deep learning model used in the project for music generation by giving an overview of the common models used in this field.

2.1.1 Deep Learning

Deep learning is a subset of machine learning, which is essentially a neural network with three or more layers. These neural networks attempt to simulate the behavior of the human brain—albeit far from matching its ability—allowing it to "learn" from large amounts of data. While a neural network with a single layer can still make approximate predictions, additional hidden layers can help to optimize and refine for accuracy.
Many artificial intelligence (AI) apps and services are powered by deep learning, which enhances automation by carrying out mental and physical tasks without the need for human intervention. Deep learning is the technology that powers both established and emerging technologies, like voice-activated TV remote controls, digital assistants, and credit card fraud detection (such as self-driving cars).

2.1.2 Multilayer Perceptron (MLP)

A deep artificial neural network (ANN), also known as a feedforward network, is at the core of contemporary deep learning. They consist of an input layer, at least one hidden layer, and an output layer, which together make up at least three layers of nodes. Every

node, with the exception of the input nodes, is a neuron with a nonlinear activation function. When an MLP is stated to be completely connected, it means that, aside from the input layer, every node is linked to every node in the preceding layer.

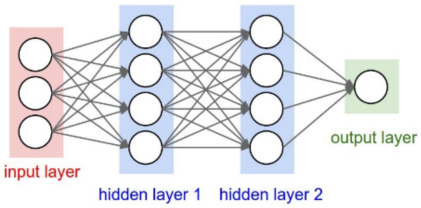

Figure 2.1: MLP

2.1.3 Recurrent Neural Networks (RNN)

A recurrent neural network (RNN) is a type of artificial neural network which uses sequential data or time series data. These deep learning algorithms are commonly used for ordinal or temporal problems, such as language translation, natural language processing (nlp), speech recognition, and image captioning.They are distinguished by their "memory" as they take information from prior inputs to influence the current input and output.

2.1.4 Generative Adversarial Networks (GANs)

Generative Adversarial Networks, or GANs for short, are an approach to generative modeling using deep learning methods, such as convolutional neural networks.GANs are a clever way of training a generative model by framing the problem as a supervised learning problem with two sub-models: the generator model that we train to generate new examples, and the discriminator model that tries to classify examples as either real (from the domain) or fake (generated). The two models are trained together in a zero-sum game, adversarial, until the discriminator model is fooled about half the time, meaning the generator model is generating plausible examples.

2.2 Advantages and Limitations of existing systems

Many applications of music generation in ML already exist. For example, Google's Magenta, offers a python library generating music using deep learning, as well as manipulating images.

There have been various approaches to generating music using deep learning models. Some existing approaches use autoregressive Recurrent Neural Networks to generate the sequence of output music. LSTM has been applied to music generation tasks to solve the vanishing gradients problem in simple RNN models and to learn the complex relationship between chords. Performance RNN is an LSTM-based model that uses event-based representation as its output. Even though Performance RNN generates music that sounds plausible for a short while, it lacks long-term structure and coherence.

Transformers have also been used to capture the long-term structure of music generation. MuseNet uses a large- scale transformer model to predict the next token in the music sequence, and it is also able to take instruments and composers as input for music generation. Music Transformer uses relative attention to focus on relational features which outperform the original transformer model.

Generative Adversarial Networks (GAN) have been applied to several domains including music generation. Transformer-GAN uses Transformer-XL as the generator and BERT as the discriminator. It achieves better per- formance compared to transformer models that maximize likelihood alone.

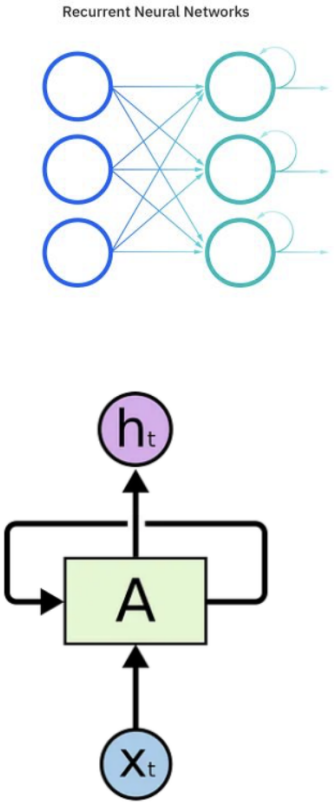

Figure 2.2: RNN

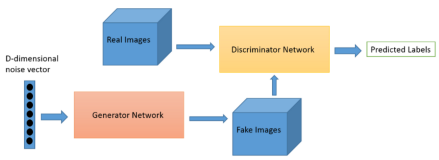

Figure 2.3: GAN's

Chapter 3

Proposed System

3.1 System Requirements

Hardware Requirements:- OS: Minimum Windows 11, MacOS Ventura

Storage: 8GB RAM and 256 GB Storage

Software Requirements:- Microsoft Visual Studio Code, PyCharm IDE, Google Colab, TensorFlow.

3.2 Design of the System

The basic design of music generation using the Wavenet Architecture

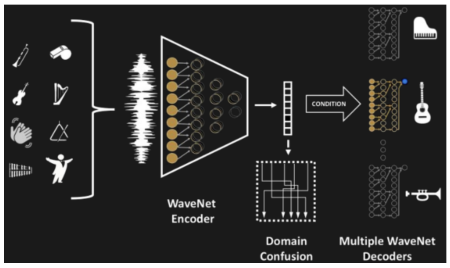

Figure 3.1: Design of the System

3.3 Algorithms and Techniques used

3.3.1 WaveNet

WaveNet is an audio generative model based on the PixelCNN architecture. In order to deal with long-range temporal dependencies needed for raw audio generation, architectures are developed based on dilated causal convolutions, which exhibit very large

receptive fields. It is a deep neural network for generating raw audio waveforms. The model is fully probabilistic and autoregressive, with the predictive distribution for each audio sample conditioned on all previous ones.

WaveNet is a powerful new predictive technique that uses multiple Deep Learning strategies from Computer Vision (CV) and Audio Signal Processing models and applies them to longitudinal time-series data. WaveNet is a deep autoregressive, generative model, which produces human-like voice, where raw audio is feeded as input to the model, taking speech synthesis to another level. It is a combination of two different ideas: wavelet and Neural networks. Raw audio is generally represented as a sequence of 16 bits.

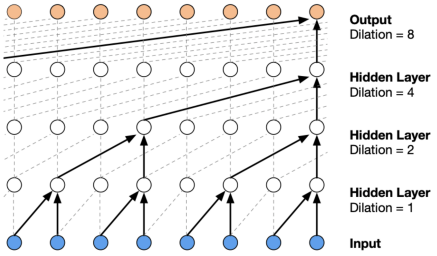

Figure 3.2: WaveNet

Chapter 4

Implementation

4.1 Tools and Technologies used

4.1.1 Music21

Music 21 is a Python library developed by MIT for understanding music data. MIDI is a standard format for storing music files. MIDI stands for Musical Instrument Digital Interface. MIDI files contain the instructions rather than the actual audio. Hence, it occupies very little memory. That's why it is usually preferred while transferring files.

4.1.2 TensorFlow

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML, and gives developers the ability to easily build and deploy ML-powered applications.

TensorFlow provides a collection of workflows with intuitive, high-level APIs for both beginners and experts to create machine learning models in numerous languages. Developers have the option to deploy models on a number of platforms such as on servers, in the cloud, on mobile and edge devices, in browsers, and on many other JavaScript platforms. This enables developers to go from model building and training to deployment much more easily.

4.1.3 Keras

Keras is a high-level, deep learning API developed by Google for implementing neural networks. It is written in Python and is used to make the implementation of neural networks easy. It also supports multiple backend neural network computation. Keras is a central part of the tightly-connected TensorFlow 2 ecosystem, covering every step of the machine learning workflow, from data management to hyperparameter training to deployment solutions.

4.2 Modules, their descriptions and Flow of the System

4.2.1 Data Preparation

1. Music21 toolkit for python to convert midi files into a list of notes chords.

2. Mapping function from above categorical data to numerical data.

3. Understanding the data. For example, the total number of unique notes.

4. Creating input output sequences to train the model.

4.2.2 Training

1. The input would be a set of nodes and chords since we are generating music.

2. Dropout - to regularize by setting a fraction of input to 0 at each update during training to prevent overfitting.

• Determine what fraction to drop.

3. Dense - fully connected neural net where each input node is connected to an output node .

4. Activation - what activation function to use to calculate output .

5. Determine how many of each layer we need .

6. Calculate loss for each iteration of training. Loss functions in Keras:

• Mean squared error - too many categories ?

• Mean absolute percentage error - too many categories ?

• Categorical Cross entropy

7. Get the weights to use for the model .

8. Callback - A callback is an object that can perform actions at various stages of training (e.g. at the start or end of an epoch, before or after a single batch, etc).

4.2.3 Testing and Final Output Generation

To generate new music, we create a function that does the following:

1. Select a random array of sample values as a starting point to model

2. Now, the model outputs the probability distribution over all the samples

3. Choose the value with the maximum probability and append it to an array of samples

4. Delete the first element and pass as an input for the next iteration

5. Repeat steps 2 and 4 for a certain number of iterations

6. The last step is to convert the data back to a MIDI format file.

Chapter 5

Results and Analysis

5.1 Performance Evaluation

When compared to the other existing datasets such as the, Maestro dataset: The Maestro dataset is a large dataset of classical music compositions, annotated with expressive timing and dynamic information. It can be used for a variety of music generation tasks. The training dataset that we have used in our project is relatively small when compared to the Maestro Dataset. Hence, we can fine-tune a pre-trained model to build a robust system. The model's performance is subjective and can be better assessed by hearing the music generated directly. From our testing, we were able to conclude that the model was able to generate proper music and Technically, prevented overfitting.

5.2 Comparison with existing systems

LSTMs are a variant of Recurrent Neural Networks and work well in cases of sequential data. They make predictions based on the current input as well as the previous inputs instead of treating every beat independently. Since a beat in music also depends on the previous beats, it is also a type of sequential data and an LSTM model is suited for it whereas our model made using wavenet is a powerful new predictive technique that uses multiple Deep Learning strategies from Computer Vision and Audio Signal Processing models and applies them to longitudinal time-series data. WaveNet is a deep autoregressive, generative model, which produces human-like voice, where raw audio is feeded as input to the model, taking speech synthesis to another level.

5.3 Limitations and future scope

This project currently has few limitations. While this model found more success with regards to the musicality of the produced notes, it was very limited in utility as it could only produce a single music melody in an instance. In the future, our goal is to enable the music generator to be able to generate multiple musical melodies in a single instance.

Another, future goal is to develop a GUI to make the interaction between the user and the deep learning model much easier, so that this project can be accessible to a wider audience.

Chapter 6

Conclusion and Recommendations

6.1 Summary of the Project

Deep Learning has a wide range of applications in our daily life. The key steps in solving any problem are understanding the problem statement, formulating it and defining the architecture to solve the problem. We had a lot of fun (and learning) while working on this project. Music is a part of our life and it was quite intriguing combining music with wavenet and producing music.

This project has been a great learning experience about the prospect of music generation using deep learning. We researched and studied various existing music generation models developed by using Deep Learning, and even if this project's main focus has been to examine and understand different algorithms through various experiments, Through this project, we developed a Music Generation model using Deep learning.

This is an exciting prospect for AI enthusiasts around the world as it lowers the barriers of entry of the field and it has already given rise to some interesting web applications leveraging the power of deep learning to create, for example, machine assisted music improvisations.

6.2 Contributions and achievements

All the members of the group have contributed equally to this project, and have made significant contributions in each and every stage of the project, right from researching various approaches and existing methods, Dataset collection, Data Pre-processing,

model building, training and testing.

6.3 Recommendations for future work

A key future goal with this project is to improve the model architecture and its accuracy, and to train the model using music recorded from various musical instruments to add more complexity and enable the deep learning model to generate better and realistic musical compositions.

Bibliography

[1] * https://ai.plainenglish.io/building-a-lo-fi-hip-hop-generator-e24a005d0144

[2] * https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-68786834d4c5

[3] * https://towardsdatascience.com/creating-a-pop-music-generator-with-the-transformer-5867511b382a

[4] * https://towardsdatascience.com/practical-tips-for-training-a-music-model-755c62560ec2

[5] * https://towardsdatascience.com/a-multitask-music-model-with-bert-transformer-xl-and-seq2seq-3d80bd2ea08e

[6] https://towardsdatascience.com/how-to-remix-the-chainsmokers-with-a-music-bot-6b920359248c

[7] MuseGAN: https://arxiv.org/abs/1709.06298

[8] MidiNet: https://arxiv .org/abs/1703.10847

[9] https://github.com/ashishpatel26/Best-Audio-Classification-Resources-with-Deep-learningdl4m- details

[10] Eck, D., J. Schmidhuber. A first look at music composition using lstm recurrent neural networks. Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale, 103:48, 2002.

[11] Choi, K., G. Fazekas, M. Sandler. Text-based lstm networks for automatic music composition. arXiv preprint arXiv:1604.05358, 2016.

[12] Simon, I., S. Oore. Performance rnn: Generating music with expressive timing and dynamics. https:// magenta.tensorflow.org/performance-rnn, 2017.

[13] Waite, E. Generating long-term structure in songs and stories, 2016. https://magenta.tensorflow.org/ 2016/07/15/lookback-rnn-attention-rnn.

[14] Payne, C. M. Musenet, 2019. https://openai.com/blog/musenet/.

[15] Huang, J., Z. Chen, Q. Le, et al. Music transformer: generating music with long-term structure. arXiv preprint arXiv:1809.04281, 2018.

[16] Goodfellow, I., J. Pouget-Abadie, M. Mirza, et al. Generative adversarial networks. Communications of the ACM, 63(11):139–144, 2020.

[17] Muhamed, A., L. Li, X. Shi, et al. Symbolic music generation with transformer-gans. Proceedings of the AAAI Conference on Artificial Intelligence, 35(1):408–417, 2021.

Appendices

Appendix A

Source code

```
# -*- coding: utf-8 -*-
"""Music-Generation-v1.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1PhtUUUNaTAVj4lf2H5RezEWTZnAYZy6s
"""

from music21 import *

#read MIDI files return notes
def read_midi(file):
    # initialize
    notes=[]
    notes_to_parse = None

    #parsing a midi file - converting to music21 objects(Scores)
    midi = converter.parse(file)

    #separate as instruments
    s2 = instrument.partitionByInstrument(midi)

    #Looping over all the instruments
    for part in s2.parts:
        #Piano and Guitar elements
        if ('Piano' in str(part) or ('Guitar' in str(part))):
            notes_to_parse = part.recurse()

            #note or chord
            for element in notes_to_parse:

                #handling a note
                if isinstance(element, note.Note):
                    notes.append(str(element.pitch))

                #handling a chord
                elif isinstance(element, chord.Chord):
                    notes.append('.'.join(str(n) for n in element.normalOrder))

```

```
    return np.array(notes)

import numpy as np
import os

path = '/content/midi/'

#read all the filenames
files=[i for i in os.listdir(path) if i.endswith(".mid")]

#reading each midi file
notes_array = np.array([read_midi(path+i) for i in files])

#converting 2D array into 1D array
notes_ = [element for note_ in notes_array for element in note_]

#No. of unique notes
unique_notes = list(set(notes_))
print(len(unique_notes))

from collections import Counter
import matplotlib.pyplot as plt

freq = dict(Counter(notes_))
no=[count for _,count in freq.items()]

#set the figure size
plt.figure(figsize=(5,5))
plt.hist(no)
plt.xlabel('Notes')
plt.ylabel('Frequency')
plt.show()

frequent_notes = [note_ for note_, count in freq.items() if count
    >=40]
print(len(frequent_notes))


new_music=[]

for notes in notes_array:
    temp=[]
    for note_ in notes:
        if note_ in frequent_notes:
            temp.append(note_)
    new_music.append(temp)

new_music = np.array(new_music)

no_of_timesteps = 32
x = []
y = []

for note_ in new_music:
    for i in range(0, len(note_) - no_of_timesteps, 1):

        #preparing input and output sequences
        input_ = note_[i:i + no_of_timesteps]
```

```
        output = note_[i + no_of_timesteps]

        x.append(input_)
        y.append(output)

x=np.array(x)
y=np.array(y)

# converting notes to ints
unique_x = list(set(x.ravel()))
x_note_to_int = dict((note_, number) for number, note_ in enumerate(
    unique_x))

#preparing input sequences
x_seq=[]
for i in x:
    temp=[]
    for j in i:
        #assigning unique integer to every note
        temp.append(x_note_to_int[j])
    x_seq.append(temp)

x_seq = np.array(x_seq)

unique_y = list(set(y))
y_note_to_int = dict((note_, number) for number, note_ in enumerate(
    unique_y))
y_seq=np.array([y_note_to_int[i] for i in y])

from sklearn.model_selection import train_test_split
x_tr, x_val, y_tr, y_val = train_test_split(x_seq,y_seq,test_size
    =0.2,random_state=42)

import keras
from keras.layers import *
from keras.models import *
from keras.callbacks import *
import keras.backend as K

K.clear_session()
model = Sequential()

#embedding layer
model.add(Embedding(len(unique_x), 100, input_length=32,trainable=
    True))

model.add(Conv1D(256,3, padding='causal',activation='relu'))
model.add(Dropout(0.2))
model.add(MaxPool1D(2))

model.add(Conv1D(128,3,activation='relu',dilation_rate=2,padding='
    causal'))
model.add(Dropout(0.2))
model.add(MaxPool1D(2))

model.add(Conv1D(64,3,activation='relu',dilation_rate=4,padding='
    causal'))
model.add(Dropout(0.2))
```

```
model.add(MaxPool1D(2))

model.add(GlobalMaxPool1D())

model.add(Dense(256, activation='relu'))
model.add(Dense(len(unique_y), activation='softmax'))

model.compile(loss='sparse_categorical_crossentropy', optimizer='adam')

model.summary()

# Save best model after each epoch based on min val_loss
mc=ModelCheckpoint('best_model.h5', monitor='val_loss', mode='min', save_best_only=True, verbose=1)

history = model.fit(np.array(x_tr),np.array(y_tr),batch_size=128, epochs=50, validation_data=(np.array(x_val),np.array(y_val)), verbose=1, callbacks=[mc])

#loading best model
from keras.models import load_model
model = load_model('best_model.h5')

# plots
print(history.history.keys())
# loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()

import random
ind = np.random.randint(0,len(x_val)-1)

random_music = x_val[ind]

predictions=[]
for i in range(20):

    random_music = random_music.reshape(1,no_of_timesteps)

    prob  = model.predict(random_music)[0]
    y_pred= np.argmax(prob,axis=0)
    predictions.append(y_pred)

    random_music = np.insert(random_music[0],len(random_music[0]),y_pred)
    random_music = random_music[1:]

print(predictions)

x_int_to_note = dict((number, note_) for number, note_ in enumerate(unique_x))
predicted_notes = [x_int_to_note[i] for i in predictions]
```

```
202
203 def convert_to_midi(prediction_output):
204
205     offset = 0
206     output_notes = []
207
208     # create note and chord objects based on the values generated by
    the model
209     for pattern in prediction_output:
210
211         # pattern is a chord
212         if ('.' in pattern) or pattern.isdigit():
213             notes_in_chord = pattern.split('.')
214             notes = []
215             for current_note in notes_in_chord:
216
217                 cn=int(current_note)
218                 new_note = note.Note(cn)
219                 new_note.storedInstrument = instrument.Piano()
220                 notes.append(new_note)
221
222             new_chord = chord.Chord(notes)
223             new_chord.offset = offset
224             output_notes.append(new_chord)
225
226         # pattern is a note
227         else:
228
229             new_note = note.Note(pattern)
230             new_note.offset = offset
231             new_note.storedInstrument = instrument.Piano()
232             output_notes.append(new_note)
233
234         # increase offset each iteration so that notes do not stack
235         offset += 1
236     midi_stream = stream.Stream(output_notes)
237     midi_stream.write('midi', fp='music.mid')
238
239 convert_to_midi(predicted_notes)
```

Appendix B

Screen shots

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 32, 100)           13500

 conv1d (Conv1D)             (None, 32, 256)           77056

 dropout (Dropout)           (None, 32, 256)           0

 max_pooling1d (MaxPooling1D  (None, 16, 256)          0
 )

 conv1d_1 (Conv1D)           (None, 16, 128)           98432

 dropout_1 (Dropout)         (None, 16, 128)           0

 max_pooling1d_1 (MaxPooling  (None, 8, 128)           0
 1D)

 conv1d_2 (Conv1D)           (None, 8, 64)             24640

 dropout_2 (Dropout)         (None, 8, 64)             0

 max_pooling1d_2 (MaxPooling  (None, 4, 64)            0
 1D)

 global_max_pooling1d (Globa  (None, 64)               0
 lMaxPooling1D)

 dense (Dense)               (None, 256)               16640

 dense_1 (Dense)             (None, 135)               34695

=================================================================
Total params: 264,963
Trainable params: 264,963
Non-trainable params: 0
_________________________________________________________________
```

Figure B.1: Model Architecture

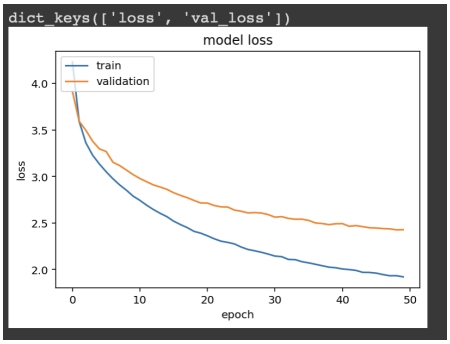

Figure B.2: Model loss graph

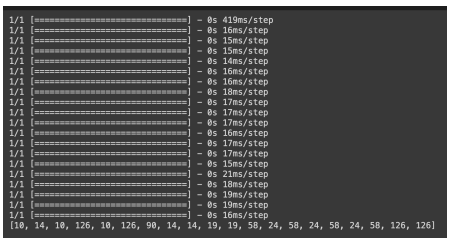

Figure B.3: Generated Music Sequence

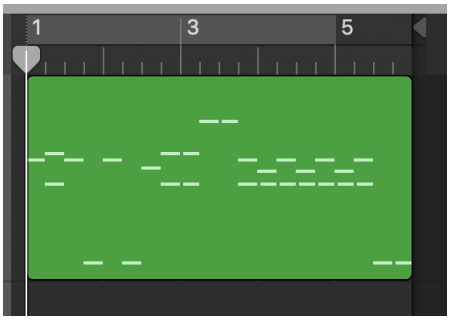

Figure B.4: Generated Music as an .mid file

Appendix C

Data sets used in the project

C.1 Data sets used in the project

The dataset used is the entire repertoire of Bach's Well-Tempered Klavier II, it contains 48 pieces of piano/harpsichord solo. The reason for that is that Bach has a large repertoire of keyboard works and as a composer from the Baroque period (1600-1750), his music is easy to understand from a listening perspective, holds less complexity in terms of nuances and "expressiveness" in the direct notation of the music mainly because of the harpischord's technical limitations but also because the time period preferred sacred music to profane "personal expression of emotions" music; however, baroque is also known for its dense polyphonic complexity as characterized by the fugue, a form of music where multiple independent melodic lines are performed simultaneously. As mentioned previously, the Baroque period highly valued sacred music which leads to the music being highly codified in terms of respect to the key and scale, time signatures, structure of pieces, instrumentation, etc. which makes the learning process for the model easier as these patterns and rigorous frameworks are easy to spot.

It is also worth noting that the file type chosen was MIDI. This presents itself as the obvious choice since MIDI files are essentially a list of events organized by tracks, events which will later be read by a music player adding a soundfont to the event to color the notes. The structure is as follows: The MIDI files are structured in chunks

MIDI File	Chunk				
	Type	Length	Data		
	MThd	6	<format>	<tracks>	<division>
	MTrk	<length>	<delta_time> <event> ...		

Figure C.1: Structure

and each chunk contains a fixed size type, fixed size length and fixed size (based on the length specified) data. Further, there are two types of chunks: header chunks (“MThd”) which will contain information on the entire file and track chunks (“MTrk”) which will contain information on a specific track. Different libraries exist to read MIDI files and retrieve information from them, the two most common ones, that were used throughout the project, are Music21 and Python-Midi.

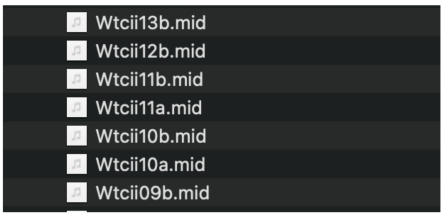

Figure C.2: Snapshots of the few files of the training data used in this project.