University of Sarajevo
Faculty of Electrical Engineering
Department of Computing and Informatics

# Generating Music Using Deep Learning

FINAL THESIS

- Bachelor Studies -

**Candidate:**
**Selma Kurtović**

**Supervisor:**
**Doc. dr Senka Krivić**

Sarajevo,
September 2023

**University of Sarajevu**
**Faculty Of Electrical Engineering**
**Department of Computing and Informatics**

# Statement of the Thesis' Autenticity

Bachelor's thesis of the 1st cycle of studies

Given name and family name: Selma Kurtovic
Title of the thesis: Generating Music Using Deep Learning
Type of the thesis: Final thesis for the 1st cycle of studies, Bachelor's studies
Number of pages:

## I hereby confirm:

- That I have read the documents which relate to plagiarism, as defined by the Statute of the University of Sarajevo, Ethical Codex of the University of Sarajevo, and regulations of studies which relate to the 1st and the 2nd cycle of studies, integrated studies program of the 1st and 2nd cycle of studies, and the 3rd cycle of studies at the University of Sarajevo, as well as the instructions on plagiarism mentioned on the web page of the University of Sarajevo;
- That I am aware of the disciplinary regulations of the University regarding plagiarism;
- That the submitted work is entirely mine, independent work, except for the part where it is indicated;
- That the paper is not submitted, entirely or partially, for obtaining a title at the University of Sarajevo or any other higher education institution;
- That I have clearly indicated the presence of the cited or paraphrased materials and that I have referenced all the sources;
- That I have consistently indicated, used, and cited sources or bibliography by some of the recommended styles of citing, with indicating complete references that completely include bibliographical description of the used and cited source;
- That I have appropriately indicated all help that I have got, besides the help of my mentor and academic tutor.

Sarajevo, September 2023

Signature:

Selma Kurtović

**Elektrotehnički fakultet, Univerzitet u Sarajevu**
**Odsjek za Računarstvo i informatiku.**
**Doc. dr Senka Krivić, dipl.el.ing**
**Sarajevo, MJESEC 2023**

## Postavka zadatka završnog rada I ciklusa:
## **Generacija muzike korištenjem tehnika dubinskog učenja**

U okviru ove teze neophodno je istražiti tehnike dubokog učenja, kao što su rekurentne neuronske mreže (RNN), konvolutivne neuronske mreže (CNN) i generativne mreže (GAN), kako bi razumjeli kako te tehnike rade u stvaranju muzike. Istovremeno će student razviti temeljno razumijevanje muzičke teorije, uključujući skale, akorde i melodije, što će im omogućiti da uspješno prenesu te koncepte u modele mašinskog učenja učenja i kreiranje adekvatne strukture podataka. Također će naučiti kako prikupiti i obraditi muzičke podatke, poput MIDI datoteka ili audio zapisa, kako bi ih pripremili za treniranje modela za učenje. Na kraju teze student će usavršiti vještinu razvoja modela, uključujući dizajn, implementaciju i fino podešavanje tih dubokih modela, što će rezultirati stvaranjem originalnih muzičkih kompozicija spajanjem tehnologije i umjetnosti.

**Mentor:** Doc. dr Senka Krivić

**Polazna literatura:**

- Ferreira, P.; Limongi, R.; Fávero, L.P. Generating Music with Data: Application of Deep Learning Models for Symbolic Music Composition. Appl. Sci. 2023, 13, 4543. https://doi.org/10.3390/app13074543

- Wu, A. H., (ur.), Deep Learning for Music, Stanford, 2016

Doc. dr Senka Krivić, dipl. ing. el.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Problem Statement and Rationale

Music holds a profound and enduring significance in the lives of individuals across the globe. It possesses the remarkable ability to touch the deepest recesses of human emotion, offering solace in times of sorrow and amplifying joy in moments of celebration. Beyond its cultural roots, music stands as a universal language, transcending boundaries and fostering connections between people from diverse backgrounds and experiences. This universal appeal allows music to serve as a powerful force for unity, bringing together individuals who may otherwise find themselves worlds apart. [1].
Delving into the realm of research, it becomes evident that music's influence extends far beyond the realm of emotions. Scientific studies have illuminated the positive impact that engaging with music can have on both brain development and cognitive abilities. The intricate interplay of melody, rhythm, and harmony stimulates neural pathways, fostering growth and adaptability within the brain's intricate network. This heightened neural activity is believed to contribute to enhanced cognitive functions, potentially leading to improvements in memory, problem-solving skills, and overall mental agility.[2].
Throughout history, making music was mostly for talented and educated individuals. With advancements in technology and the rise of AI, we wonder if everyone can now create music.
Using Artificial Intelligence to make music brings many benefits. People can make music that suits their own tastes and feelings. This helps them feel a stronger connection to their music and enhances their creative experience
They can also make music in the style of their favorite musicians, even if those musicians are no longer alive. This not only pays tribute to musical legends but also gives new energy to their iconic styles.

## 1.2 Hypothesis

### 1.2.1 Hypothesis 1: *Deep neural networks excel at creating music.*

When we listen to music, we're essentially hearing a series of notes played in a particular order. Deep neural networks have the ability to generate these sequences, making them well-suited for composing music.
What defines a song is the arrangement of these notes, known as the melody. Musicians play a crucial role in crafting melodies by using their knowledge of music theory.
Deep learning, a sophisticated form of machine learning, is adept at recognizing patterns and

connections in input data. When fed enough information, neural networks can grasp the relationships between different musical elements, much like an experienced musician. This means that, when trained on a specific artist's musical data, they can learn to create new music in a similar style.

### 1.2.2 Hypothesis 2: *Using a larger set of features during neural network training leads to better results.*

Making music with artificial methods is a complex task. Musicians have to think about things such as which notes and chords to use, measure the duration of each note, factor in a host of other musical elements, and many other details. So, when we're trying to teach a computer to create music, we need to think deeply about all these aspects.
By including more musical elements as inputs and outputs for the neural network, we hope to make the computer-generated music that resembles human-created sound. This means the computer will have a better grasp of the intricate decisions and nuances involved in making music. Ultimately, this should lead to computer-generated music that sounds more genuine and true to human expression.

## 1.3 Description of the Proposed Solution, Objectives of the Work, and Methodology

### 1.3.1 Description of the Proposed Solution

The proposed solution presented in this paper introduces an algorithm based on Long Short-Term Memory (LSTM) networks. This algorithm functions as an independent music generation system, eliminating the need for human intervention in the creative process. Specifically tailored to leverage the capabilities of deep learning our model is designed with the primary objective of composing music.
This model possesses an ability to absorb knowledge from a dataset comprising intricate note features. It analyzes this data, utilizing its neural network architecture to discern patterns and structures, ultimately culminating in the creation of entirely new musical compositions. This fusion of cutting-edge technology with the rich musical heritage of The Beatles marks a significant leap forward in the realm of automated music composition.
What sets our approach apart is its inclination to emulate the distinctive style of The Beatles. By training on a comprehensive dataset of MIDI files attributed to this legendary band, our model is poised to encapsulate the essence and nuances that defined The Beatles' musical legacy.
In order to facilitate this process, we employ the versatile music21[3] library. This tool enables the systematic processing of MIDI files, extracting critical features that serve as the building blocks for our model's creative endeavors. These extracted features are then transformed into a structured JSON file.

### 1.3.2 Objectives of the Work

Our research endeavor encompasses a multifaceted approach, beginning with the meticulous curation and structuring of an extensive dataset containing pertinent MIDI files. These files are prepared in a format optimized for seamless integration into our research framework. Building upon this foundation, we delve into the training phase, leveraging the neural network model

outlined in the research paper 'Deep Learning for Music.' This training process is pivotal, as it empowers the model to autonomously generate novel musical compositions, affording us valuable insights into its inherent capabilities and potential limitations.
As we progress, our focus shifts towards model enhancement, where we strive to elevate the performance and creative output of our neural network. This involves the introduction of innovative features and techniques, aimed at refining and augmenting the model's proficiency in music generation. Subsequently, a rigorous evaluation ensues, deploying a diverse range of evaluation metrics to comprehensively gauge the model's performance. This step is crucial in ensuring a nuanced understanding of the model's capabilities and areas for potential refinement. To gain a comprehensive perspective, we undertake a meticulous analysis, scrutinizing the strengths and weaknesses of both the original and enhanced models. This comparative assessment serves as a cornerstone for identifying areas of improvement and innovation. Moreover, our exploration extends towards the horizon of future enhancements in the realm of music generation through deep learning techniques. We aim to chart potential pathways for further advancements, ensuring a continual evolution in this dynamic field. Through this holistic approach, we aim to contribute meaningfully to the convergence of artificial intelligence and music composition, pushing the boundaries of what is achievable in this exciting domain.

### 1.3.3 Methodology

- **Creating a Dataset:** We begin by collecting a comprehensive dataset of MIDI files, which serve as the foundation for our music generation system.
- **Data Preprocessing:** The collected MIDI data undergoes preprocessing, including data cleaning and formatting, to ensure consistency and suitability for training our neural network.
- **Feature Extraction:** Essential musical features, such as note pitches, durations, and played attributes, are extracted from the preprocessed data to serve as input features for the model.
- **Creating a Model:** Our model architecture comprises Dense and LSTM layers, which are configured to capture temporal dependencies in the music data. Three separate branches handle pitch, duration, and played attributes.
- **Data Postprocessing:** After music generation by the model, we apply postprocessing techniques to the generated sequences to ensure musical coherence and structure.
- **Model Evaluation:** We rigorously evaluate the performance of our LSTM-based music generation system using a range of evaluation metrics. This assessment provides insights into the system's ability to generate music in a desired style.

## 1.4 Expected Results

Through this research undertaking, our primary aim is to develop an advanced neural network model that builds upon the foundational framework of its predecessor. This evolution will be fueled by integrating innovative features and techniques aimed at elevating the model's proficiency in music generation. We anticipate that this enhanced iteration will manifest in a discernible improvement across various facets, including the generated compositions' quality, intricacy, and diversity, showcasing a notable leap forward in AI-driven musical creativity.

To gauge the efficacy of our enhanced neural network model, we will embark on a thorough performance evaluation utilizing a diverse array of evaluation metrics. This comprehensive assessment will give us invaluable insights into the model's inherent capabilities and potential limitations, providing a nuanced understanding of its areas of strength and opportunities for refinement.
Additionally, a detailed analysis will be conducted to meticulously scrutinize the strengths and weaknesses of both the original and enhanced neural network models. This comparative examination will serve as the bedrock for discerning potential pathways toward future enhancements and breakthroughs in the dynamic realm of music generation through deep learning techniques.
Overall, our research aims to not only enhance the capabilities of deep neural networks, particularly LSTM-based models but also to pioneer new horizons in the creative sphere of music composition. By pushing the boundaries of what is achievable in this domain, we aim to contribute significantly to the convergence of artificial intelligence and artistic expression, furthering our understanding of how technology can revolutionize the world of music.

## 1.5 Thesis overview

In the opening chapter of this thesis, we delve into the motivation and sources of inspiration behind our research paper. This section establishes the hypotheses we intend to investigate and validate in our subsequent work. Initially, we explicitly outline our objectives and anticipated outcomes.
The second chapter provides an overview of relevant systems and innovations within the AI music domain. Additionally, we offer a concise description of the research paper that served as inspiration for our own work.
The third chapter provides insight into the machine learning methods used during the research and the discovery of the optimal solution.
The fourth chapter offers a detailed description of the data used for training and the architecture of the model we have developed.
The fifth and final chapter provides a description of the technologies employed in the implementation of this project, along with a discussion of the results we have achieved.

# Chapter 2

# Review of the Literature and Existing Systems

The realm of music generation through artificial intelligence (AI) has emerged as a pivotal and cutting-edge area of inquiry, with its roots extending far into the annals of history. One seminal milestone in the history of AI music can be attributed to Lejaren Hiller, whose creation of the Illiac Suite marked a watershed moment. In 1956, the ILLIAC 1, acknowledged as the inaugural 'supercomputer' employed by an academic institution, undertook the composition process. This innovative machine orchestrated the suite by orchestrating an array of algorithmic choices, as supplied by Hiller. Although initially met with bewilderment, the accomplishments of this computer are now hailed as remarkably avant-garde. [4]
In 1981, David Cope embarked on his exploration of algorithmic composition as a remedy for his creative impasse. The offspring of his endeavors was "Experiments in Musical Intelligence," affectionately known as Emmy—a semi-automatic system amalgamating Markov chains, musical grammars, and combinatorics. Emmy, drawing inspiration from the pioneering works of Iannis Xenakis and Lejaren Hiller, expeditiously garnered acclaim for its adeptness at learning from and emulating other composers, cementing its status at the vanguard of AI-driven music generation. Cope's meticulous documentation of Emmy's metamorphosis, spanning from erudite papers to open-source code on GitHub, underscores its monumental standing in the intersection of music and artificial intelligence—an emblematic fusion of human ingenuity and technological prowess. [5]
In the present day, a plethora of contemporary AI music generators such as Magenta and Jukebox persistently redefine the boundaries of what is achievable in the realm of AI-propelled musical composition. Magenta, for instance, stands as an open-source research initiative dedicated to probing the convergence of machine learning with the creative processes underlying art and music. It furnishes an assortment of neural network models, encompassing sequential models, variational autoencoders, and neural synthesizers, facilitating the generation of music across an array of styles and attributes. Likewise, Jukebox, an innovation by OpenAI, harnesses the power of neural networks to craft music spanning diverse genres and styles, thereby showcasing the ongoing evolution and innovation in AI music composition. These state-of-the-art systems epitomize the culmination of decades of toil and progress in the domain of AI music generation, forging a bridge between technology and artistic expression.
The pursuit of advancing music generation through artificial intelligence continues to be a vibrant and dynamic field, with countless scientists and researchers tirelessly dedicating themselves to pushing the boundaries of what is achievable. This collective effort represents a tapestry of creativity and innovation, where each contributor strives to refine and enhance their

respective music generation systems.
In our own journey within this domain, we draw inspiration from the seminal work "Deep Learning for Music" authored by Allen Huang and Raymond Wu, luminaries in the field associated with the prestigious Stanford University. Their pioneering endeavor culminated in the creation of a generative model characterized by a sophisticated deep neural network architecture. This architectural marvel is meticulously designed to transcend the realm of mere computational output, instead aiming to craft musical compositions endowed with the intricate interplay of melody and harmony, akin to the artistic expression emanating from a human composer's soul.[6]

# Chapter 3

# Methods

## 3.1 Artifical Neural Networks

An Artificial Neural Network (ANN) is an information processing paradigm inspired by the way biological nervous systems, such as the human brain, process information. ANNs consist of interconnected processing elements called neurons, which work collaboratively to solve specific problems. These networks learn from examples and adjust the connections (synaptic connections) between neurons, similar to how biological systems learn. ANNs are used in various applications, such as pattern recognition, data classification, and more, making them a valuable tool in machine learning and artificial intelligence.[7]

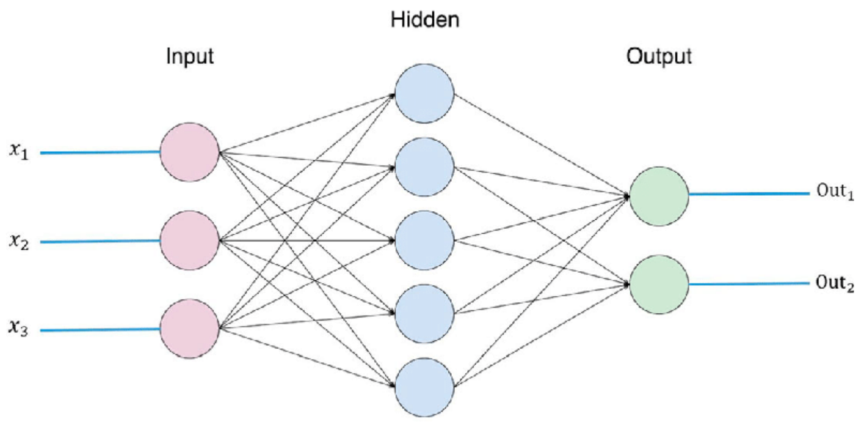

**Figure 3.1:** Graphical representation of Artificial Neural Networks

## 3.2 Long Short Term Memory Neural Networks

Long short-term memory (LSTM) is a specialized variant of an Recurrent Neural Networks (RNNs) model. What makes LSTM architecture stand out is it's ability to handle long time-series data and it's automatic control for retaining relevant features or discarding irrelevant features in the cell state. LSTM effectively addresses the vanishing gradient problem by implementing mechanisms that prevent it from happening during training.

The core innovation of LSTMs revolves around their ability to uphold a "cell state" - a continuous conduit permeating through the network. This cell state empowers LSTMs to persistently convey and preserve information across multiple time steps without degradation, a common issue in standard RNNs. The regulation of this cell state is managed by specialized structures known as gates, which govern the flow of information.

LSTMs are equipped with three primary gates, the input gate, forget gate, and output gate. Forget Gate dictates which information from the preceding cell state should be disregarded or erased, and Input Gate determines which new information should be incorporated into the cell state. The third one would be the Output Gate which filters the cell state to generate the final output based on the prevailing context. [8] These gates give LSTMs ability to selectively retain, update, and retrieve information, rendering them exceptionally effective in tasks demanding the modeling of long-term dependencies.
LSTMs are extremely successful in solving tasks that include some type of sequences, such as natural language processing, speech recognition, and time series analysis.

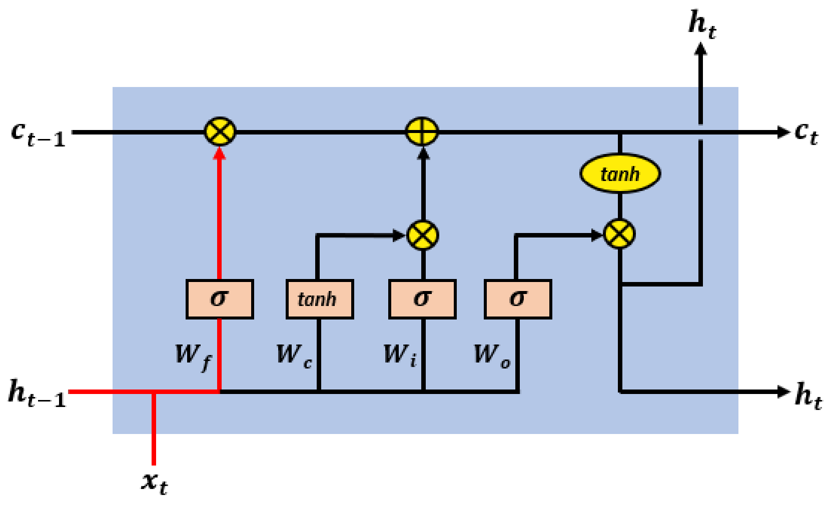

**Figure 3.2:** Graphical representation of LSTM Neural Networks

## 3.3 Activation functions

An activation function is a function that is incorporated into an artificial neural network architecture in order to help the network learn complex patterns in the data.
The role of an activation function is to convert the input signal of a network's layer into an output. This output signal is then passed as input to the next layer within the network. This conversion involves the calculation of a sum, which consists of inputs and their respective weights and bias of that layer. This computed sum is then subjected to an activation function, resulting in the output for that specific layer, which is subsequently forwarded as input to the following layer in the network.
Incorporating an activation function within a neural network is crucial because without it, the network's output signal would be a basic linear function, essentially a first-degree polynomial. While linear equations are straightforward and easy to work with, their capabilities are limited, and they lack the capacity to learn and recognize intricate patterns within data. [9]

### 3.3.1 Softmax Activation Function

Softmax serves as an activation function responsible for transforming values into probabilities. Its output is a vector comprising probabilities for each potential outcome, and the sum of these probabilities within the vector always equals one.
Mathematically, Softmax is defined as,

$$\text{Softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \quad \text{for } i = 1, 2, \ldots, n$$

Where:

$\mathbf{x}$ is the input vector of scores for each class.

$\text{Softmax}(\mathbf{x})_i$ is the i-th element of the output Softmax vector.

$e^{x_i}$ is the exponential of the i-th score in the input vector.

$\sum\limits_{j=1}^{n} e^{x_j}$ is the sum of the exponentials of all scores in the input vector.

$i$ ranges from 1 to $n$, representing each class or category.

The Softmax function is commonly used in neural networks for multi-label classification tasks.

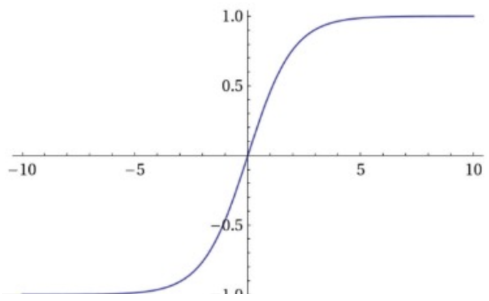

**Figure 3.3:** Graphical representation of Softmax Activation Function

### 3.3.2 Sigmoid Activation Function

A sigmoid function is an activation function with a characteristic "S"-shaped curve or sigmoid curve. It transforms any real number to a number between 0 and 1.
Mathematically, Sigmoid is defined as,

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Where:

$\sigma(z)$ : Sigmoid activation output

$z$ : Input to the sigmoid function

$e$ : Base of the natural logarithm

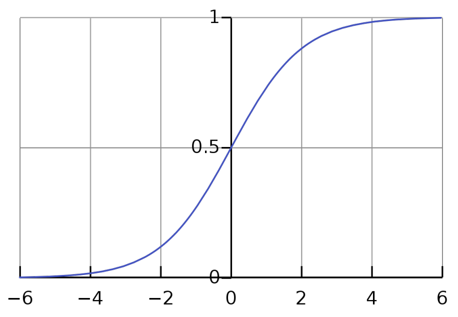

**Figure 3.4:** Graphical representation of Sigmoid Activation Function

The sigmoid function is commonly used in neural networks for binary classification tasks.

# Chapter 4

# Proposed System Overview

The system presented in this paper was inspired by and is an enhanced version of the system developed in the research paper titled "Deep Learning from Music."[6] It has been implemented using the Keras library in Python. The neural network architecture comprises two Long Short-Term Memory (LSTM) layers, each with 128 neurons, configured to return sequences of data. For training this network, the authors of the original work used compositions by Bach and successfully created pieces that resemble Bach's style. This is why we chose this network as our inspiration and decided to build upon it.
We initially conducted training on this network using our dataset and observed its competence in generating music in The Beatles' style. However, a noticeable limitation was the monotony and lack of rhythm in the generated compositions. This was primarily attributed to fixed note durations applied during post-processing of the neural network results.
Another drawback identified in this model was the absence of pauses in the generated music. Naturally, musical compositions include pauses, especially when performed using a single instrument. To address this limitation and enhance the model's performance, we decided to incorporate pauses into our dataset.

## 4.1 Dataset

The first step in creating the model we envisioned is creating a dataset. Our dataset consists of 183 The Beatles songs which were collected from online sources. We chose not to filter the dataset; our aim was to include all available songs. This decision was made to ensure that our model had access to a sufficient amount of data to learn and capture the unique style of The Beatles.

### 4.1.1 Data source

Musical Instrument Digital Interface (MIDI) is a standard to transmit and store music. Since it's creation, It has become a popular tool in the world of music production and composition.
What sets MIDI apart is its ability to represent music in a highly flexible and editable format. Instead of recording audio waveforms, MIDI records musical data such as note pitches, durations, velocity, and even control changes. This data can then be played back on different MIDI-compatible instruments, allowing for a different instruments and sounds to be used in a composition without the need for physical performers or recording sessions.
In the context of our project, MIDI files serve as an ideal input and output source because they

contain the essential elements of musical compositions in a format that can be manipulated and features can be easily extracted.

### 4.1.2 Features

Features that we concluded are important for our neural network are pitch and duration of a note. We extracted them from files using music21 library.

#### Music21

Music21[3] is a Python library developed by MIT, designed to assist in computer-aided musicology. It serves as a valuable tool for scholars and other users, enabling them to manipulate musical compositions and gain knowledge about music.
In music21, musical compositions are represented using a complex hierarchical structure that comprises various classes and subclasses. Scores encapsulate the entire musical composition, including information about the instruments used, the notes played, their durations, and their arrangement in time. Parts, on the other hand, represent individual voices or instrument lines within a score.

#### Pitch

Pitch is the attribute of a sound that defines its perceived highness or lowness, and it is directly related to the frequency of sound vibrations. In music, pitch plays a fundamental role in melody and harmony, allowing for the creation of distinct musical notes and tones. [10]
Given the significance of pitch in music, we have chosen it as a feature for both the input and output of our neural network. Our dataset consists of 302 distinct pitches, and in the following chart, we've visualized the ten most frequently occurring pitches in our dataset.

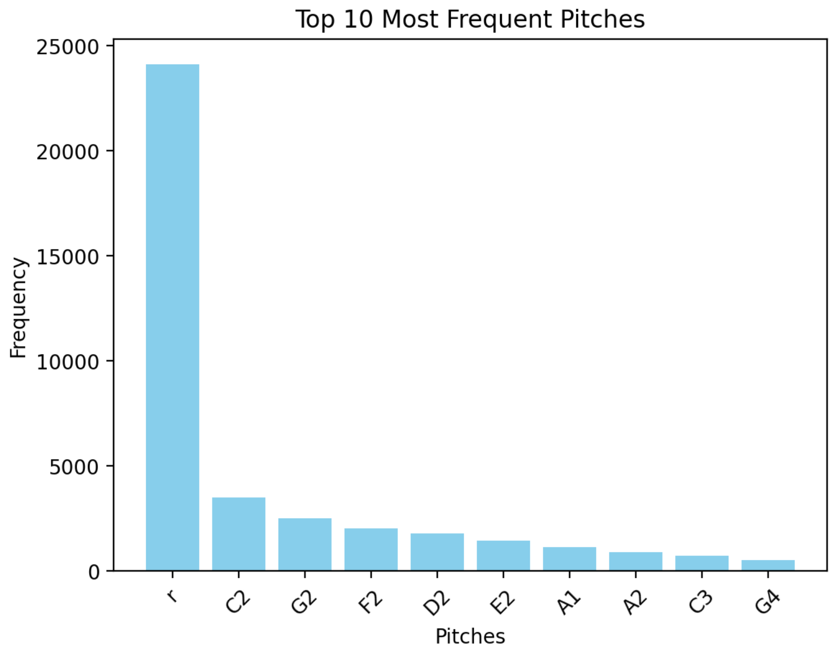

**Figure 4.1:** Top 10 Most Common Pitches in the Dataset

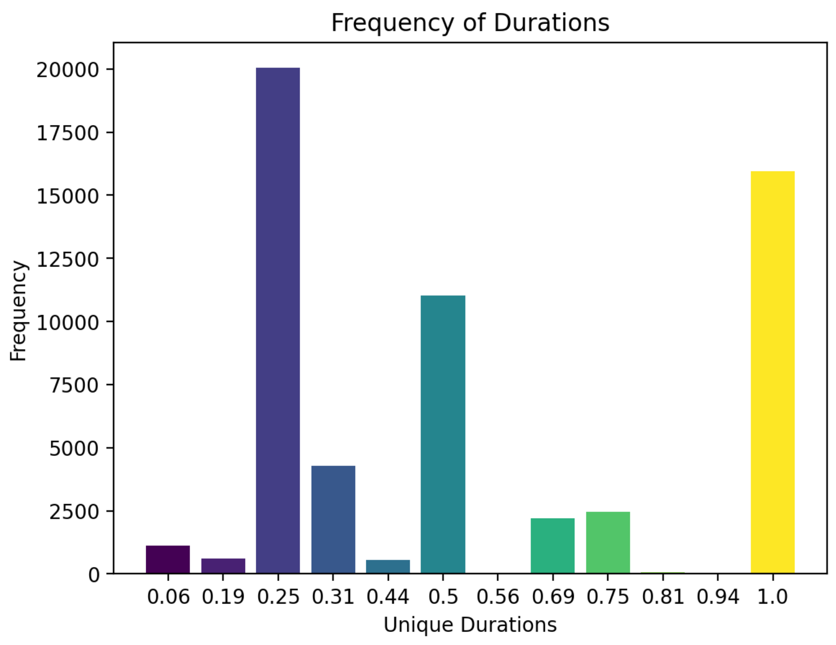

**Figure 4.2:** Frequency of Durations

### Note Duration

Note duration is the length of time that a note is played. This aspect of music is vital in shaping the rhythm and timing of a musical piece, enabling musicians to craft vibrant and engaging compositions.
As mentioned before, one of the disadvantages of the original system is the lack of variation in note durations, leading to monotonous musical compositions. To address this and create musical pieces with richer rhythmic patterns, we have chosen to include duration as both an input and an output of our neural network.
In our dataset, there are 52 unique duration values. To improve accuracy during training, we have chosen to convert these values into 12 unique classes.
The following chart illustrates the frequency distribution of these 12 classes within our dataset. The values of note durations have been represented as twelfths. This meant that each duration was rounded to the nearest twelfth. For durations less than 1, we simply rounded them, while those greater than 1 required additional processing. We split these notes into multiple notes with the same pitch. For example, if a note had a duration of 2.75, it was split into three notes with the following durations: 1, 1, 0.75, and then converted into twelfths like the others. To indicate that a note had been split and to allow some notes to have durations greater than 1 during generation, we introduced a new attribute named "played".
Played is a binary value. When splitting a note, only the first note has a "played" attribute value of 1, while the others have a value of 0.

## 4.1.3 Data representation

After extracting all the features, we need to save the data somewhere. For this purpose, we have chosen the JSON file format. This format was selected because it allows us to represent all the notes clearly and systematically. Each element of the JSON array will contain attributes such as pitch, duration, and played, which will be mapped from MIDI files in the manner described

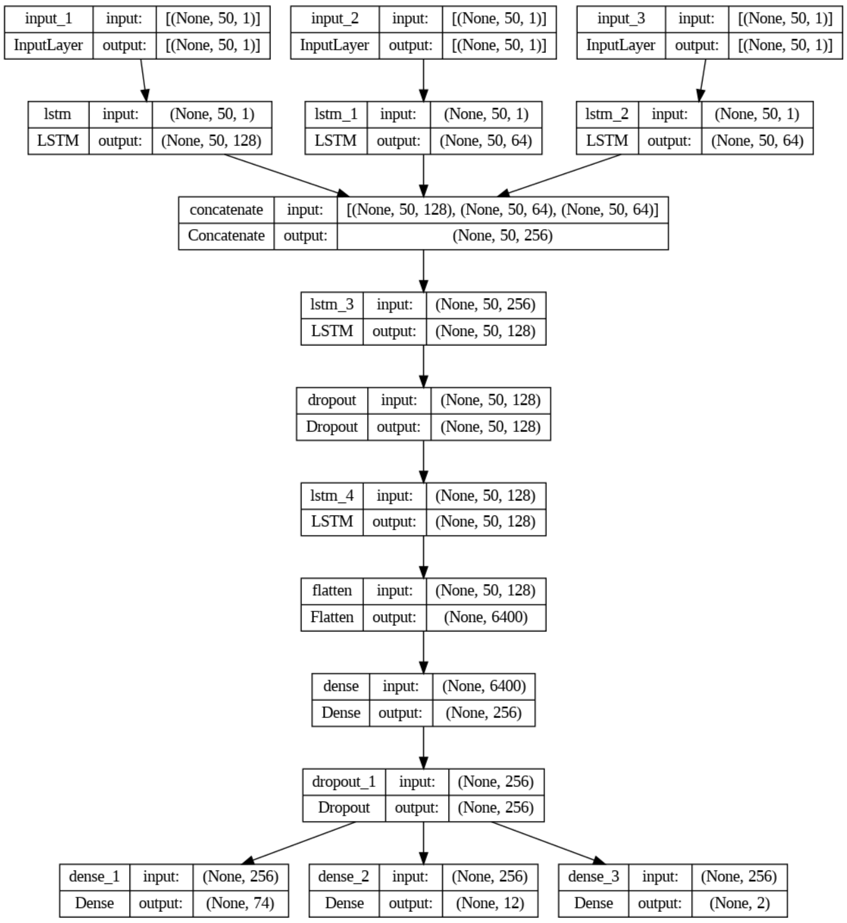
input_1 | InputLayer | input: [(None, 50, 1)] | output: [(None, 50, 1)]
input_2 | InputLayer | input: [(None, 50, 1)] | output: [(None, 50, 1)]
input_3 | InputLayer | input: [(None, 50, 1)] | output: [(None, 50, 1)]
lstm | LSTM | input: (None, 50, 1) | output: (None, 50, 128)
lstm_1 | LSTM | input: (None, 50, 1) | output: (None, 50, 64)
lstm_2 | LSTM | input: (None, 50, 1) | output: (None, 50, 64)
concatenate | Concatenate | input: [(None, 50, 128), (None, 50, 64), (None, 50, 64)] | output: (None, 50, 256)
lstm_3 | LSTM | input: (None, 50, 256) | output: (None, 50, 128)
dropout | Dropout | input: (None, 50, 128) | output: (None, 50, 128)
lstm_4 | LSTM | input: (None, 50, 128) | output: (None, 50, 128)
flatten | Flatten | input: (None, 50, 128) | output: (None, 6400)
dense | Dense | input: (None, 6400) | output: (None, 256)
dropout_1 | Dropout | input: (None, 256) | output: (None, 256)
dense_1 | Dense | input: (None, 256) | output: (None, 74)
dense_2 | Dense | input: (None, 256) | output: (None, 12)
dense_3 | Dense | input: (None, 256) | output: (None, 2)

**Figure 4.3:** Graphical representation of our neural network

above.

## 4.2 Neural Network Architecture

The neural network we've developed is a multi-branch model capable of processing input data from three distinct sources: pitch information, note duration, and the "played" attribute, which we elaborated on previously. The network model is developed using the Keras library implementing separate branches for each category of input data. Within each branch, we've incorporated Long Short-Term Memory (LSTM) layers, with the first branch having 128 neurons in it's LSTM layer, the second 64, and the third also 64.
The pitch, duration, and played inputs each pass through their respective LSTM layers to capture sequential patterns. The outputs of these branches are then concatenated to combine the information learned from each input source. After the concatenation, additional LSTM and dense layers are applied to further process the combined data and extract relevant features. Dropout layers are used for regularization to prevent overfitting.
The network has three distinct output branches, each responsible for predicting a specific aspect: pitch, duration, and the "played" attribute. Activation functions, such as softmax for pitch and duration and sigmoid for "played," are applied to the output layers to produce the final predictions. The model is compiled with specific loss functions and optimization techniques tailored to each output task.

### 4.2.1 Input layer

The input layer consists of three separate branches, each assigned the task of processing input data and passing it to the next combined layer. Every branch receives a sequence containing 50 values as its input data. These sequences are generated during the preprocessing phase by dividing musical notes into segments, each containing 51 notes or chords. For input data, we extract 50 notes/chords, leaving the last one to serve as the expected output for that particular input.

### 4.2.2 Hidden layers

In the architecture, we employ a series of Long Short-Term Memory (LSTM) layers, which are specialized components of recurrent neural networks (RNNs). The first three LSTM layers, each equipped with a specific number of neurons (128, 64, and 64, respectively), are responsible for processing individual branches of input data, namely pitch, duration, and the "played" attribute. These layers are configured to return sequences of data, making them ideal for recognizing and retaining the temporal nuances present in music.
Following the LSTM layers, we introduce a concatenation layer that merges the outputs of the previous branches into a unified sequence. This fusion allows for a holistic view of the music data, incorporating information from all three branches. Continuing our journey through the hidden layers, we encounter two additional LSTM layers, each comprising 128 neurons. These layers further dissect the combined sequence, capturing intricate patterns and relationships vital for music generation. To enhance the model's generalization and mitigate overfitting, dropout layers with a rate of 0.2 are applied. Subsequently, a Flatten layer reshapes the data, transitioning it into a one-dimensional vector. This transformation prepares the information for further processing by a Dense layer with 256 neurons and a ReLU activation function. As a final touch, another dropout layer, with a rate of 0.3, is employed to refine the model's generalization and ensure optimal performance.

### 4.2.3 Output layer

The output layer consists of three distinct branches, mirroring the structure of the input layer. Each branch serves a unique purpose, and we'll describe them individually.
In the first and second branches, the output data is encoded using a one-hot encoding technique, which is commonly employed in multi-label classification tasks. The number of neurons in the output layer matches the number of potential classes. For activation, we employ the softmax function. Softmax assigns a probability to each neuron, representing a class, and the neuron with the highest probability is activated and class is predicted. The first branch accommodates 301 possible classes, whereas the second branch handles 12 classes, corresponding to the number of classes in each of them, respectively.
The third branch is used for binary classification task. It consists of two neurons which make decision if note is played or not.
Each branch works on its own, producing results tailored to its task. This lets us assess each branch's performance and accuracy separately, helping us understand how they contribute to our neural network's abilities.

# Chapter 5

# Evaluation

## 5.1 Overview of Used Technologies

Our project was implemented using Python, a widely adopted language in the machine-learning community. Python's popularity stems from its rich collection of libraries and tools that greatly facilitate data preprocessing and model development. We made use of essential libraries like NumPy and music21 for data analysis and preparation, ensuring our dataset was ready for training. TensorFlow, a powerful deep learning framework, played a key role in building and training our neural network model. Additionally, Matplotlib was employed to create visualizations and charts for better understanding and analysis. These tools and technologies collectively supported our exploration of automated music composition through machine learning.

## 5.2 Experimental Setup

The implementation and training of our project were carried out in Google Colab, a cloud-based Jupyter notebook environment. To accelerate the training process, we leveraged the computational power of the Tesla T4 GPU.
For the training of our neural network model, we adopted specific configurations. We used a batch size of 50 as the input data size for each training iteration. The training process consisted of 200 epochs, allowing the model to learn intricate patterns and relationships within the data.
During training, we utilized the ModelCheckpoint callback to save the best-performing model weights. This ensured that we retained the most optimal version of the model for later use.
Furthermore, we normalized the input data to enhance the training process. This normalization step helped in aligning the data across different branches of our multi-input model.
Overall, these configurations and tools were instrumental in training our neural network model and achieving the desired results in genrating music using neural networks.

## 5.3 Performance Evaluation Methods

To evaluate the performance of our neural network, we will consider both the loss and accuracy of our model.
The loss function exhibits a consistent decline throughout all epochs of the training, suggesting that the network is performing well. Towards the end of the training, the loss function converges to a value of nearly 0, which strongly indicates the success of the training process.
Regarding accuracy, we need to analyze each branch separately since accuracy is calculated

for each branch individually. The branches exhibit accuracies of 0.5799, 0.2928, and 0.7526, respectively. It is natural that we did not achieve a perfect score, given the specific objectives of our network.
In our network, the selection of the next note/chord and its duration is determined after processing a sequence of 50 notes. Even a human composer does not consistently choose the same note on every occasion. It is possible that in our dataset, there were instances where different next notes were chosen for the same preceding sequence.

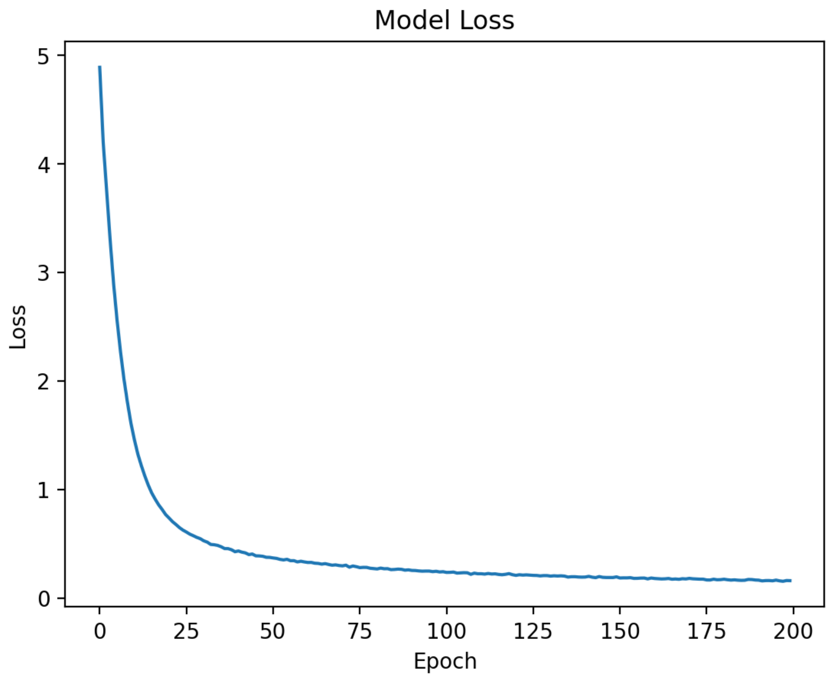

**Figure 5.1:** Loss function

## 5.4 Results

After training our model, we achieved a decent level of accuracy, and it seemed that generating music would be an easy task. However, upon generating music, we encountered a significant issue where our model exhibited a preference for certain notes or chords and durations over others. While we are yet to pinpoint the exact root cause of this behavior, we suspect that the uneven distribution of notes in our dataset may be a contributing factor.
This issue notably impacted the third branch of our model, which determines whether a note should be played or not. Unfortunately, we consistently received zero predictions from this branch, which would have resulted in the omission of all notes if we had relied solely on its output. To address this limitation, we made the decision to exclude this branch entirely from the music generation process. Instead, we are in the process of developing an alternative system for predicting whether a note should be played, ensuring a more balanced and harmonious musical output.
To overcome this issue, we decided to perform post-processing on the generated data.

### 5.4.1 Results without data preprocessing

As mentioned previously, during the music generation process using our neural network, we encountered a repetitive pattern where the same note/chord and duration were consistently produced. Specifically, the first layer of our model consistently generated the "Rest" element, which is the most prevalent input element in our dataset. Similarly, in the second duration layer, we repeatedly obtained "one-halfs" and "one-quarters" durations, which are among the most frequent durations found in our dataset.

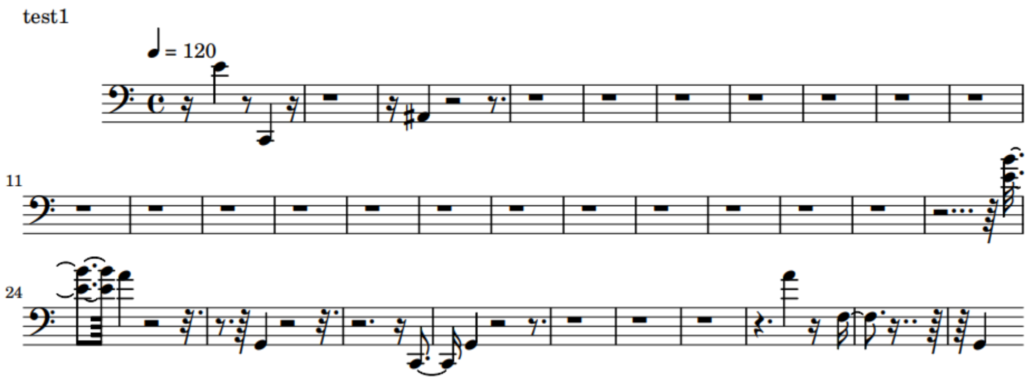

**Figure 5.2:** A music sheet created without prior data preprocessing

### 5.4.2 Data post-processing

In our pursuit of enhancing the diversity and richness of our generated music compositions, we grappled with the challenge of consistently producing the same note/chord and duration as the output. To address this, we deliberately introduced an element of controlled randomness into the results. As elucidated earlier, the softmax function plays a pivotal role in generating an array of probabilities in the output layer. Conventionally, our music generation process would zero in on the note and duration with the highest probability, deeming it the predicted output. However, in a deliberate departure from the norm, we opted for a more exploratory approach. Instead of singling out just one outcome, we expanded our horizons by considering the three elements with the highest probabilities as potential candidates.
Subsequently, we harnessed the Python random function to introduce an element of stochasticity, allowing for the random selection of a single outcome from this refined set of possibilities. Instead of blindly selecting the highest probability note, it introduces an element of randomness by considering the top three predicted values for both pitch and duration. This randomness allows for a more diverse and less repetitive musical composition.
In the case of the "played" attribute, employing a randomized approach wouldn't yield meaningful results due to the binary nature of this attribute, where there are only two possible outcomes: played or not played. Randomization in this context would not contribute to improved results. Therefore, we opted for a weighted probability method to handle this attribute. To achieve this, we utilized the NumPy function named "choice," which supports probability distributions, allowing us to make informed selections based on predefined probabilities.
To decide whether a note should be played or not, the function takes into account the previous pitch. If the previous pitch aligns with the currently predicted pitch, there is a higher probability that the note will be played. Conversely, if the previous pitch differs from the predicted pitch, the probability of playing the note decreases. This consideration adds a layer of musical

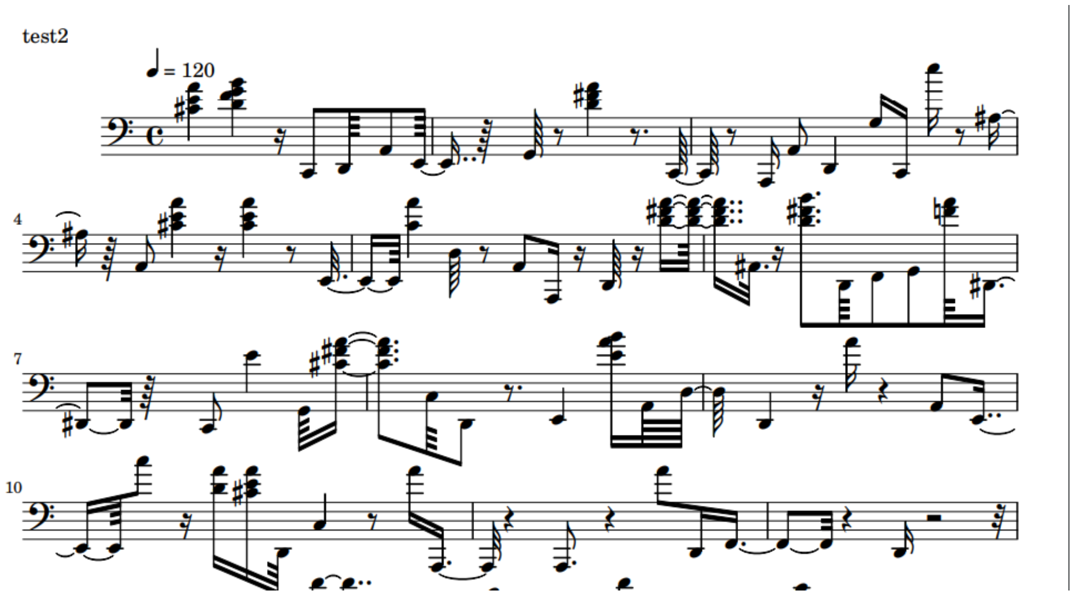

**Figure 5.3:** A music sheet created after data preprocessing

coherence to the generated composition, making it sound more natural.
In essence, this weighted probability approach ensures that the generated music exhibits both variety and musicality. It leverages the machine learning model's predictions while infusing an element of chance, resulting in musical compositions that strike a balance between predictability and creativity.

# Chapter 6

# Conclusion

Our objective was to enhance the existing model and adapt it to a new dataset. To accomplish this goal, we introduced a new data structure for representing important features of musical notes. This innovative data structure not only addressed the challenges related to dataset cleaning and manipulation but also paved the way for additional refinements and model enhancements. Overcoming one of the key challenges involved crafting and fine-tuning an appropriate machine learning model capable of effectively processing this data and extracting valuable insights from it.

The process proved to be lengthy and challenging due to the experimental nature of the task. Achieving the right architecture on the first attempt was nearly impossible, necessitating persistence and experimentation with various implementations. Initially, our objective was to make minor modifications to an existing neural network without significantly altering its core architecture. Our goal was to expand the output layer to include duration predictions while simultaneously predicting pitch and duration outputs. However, this approach did not yield the desired results, as the loss function consistently increased during training.

Our subsequent attempt involved adding multiple input layers while retaining the rest of the network's structure. Once again, we encountered the same issue and were compelled to seek a new solution. The final architecture we developed and presented utilizes multiple input and output branches. We concluded that these changes were necessary because the desired outcome was considerably more intricate than what the original paper had proposed. Achieving this model's success required fine-tuning, layer adjustments, and multiple training iterations.

We can claim that we achieved the goal we set, resulting in songs with a more intricate structure characterized by varying pitches and durations. The significant modification we implemented involved the inclusion of rests. Since this neural network is designed to generate music for a single instrument, the presence of rests is entirely expected. Furthermore, the songs even incorporate brief melodies, which could be further extracted and developed into complete compositions, thereby improving their rhythm and making them more akin to something created by a human composer.

The model could certainly be improved by making its structure more complex and conducting additional data analysis and cleaning. However, we can say that we are satisfied with the goals we have achieved.

## 6.1 Future work

As mentioned before, the model is not perfect and could benefit from some modifications. One of the ideas that could be easily implemented is removing the endings of songs from input

sequences. That way, we could achieve better harmony and perhaps post-process the endings later.
Something that would significantly improve our model is multiple-instrument music generation, as that would create sound more similar to The Beatles' music. However, this would be a challenging modification and would require extensive research to synchronize all instruments and produce pleasant-sounding music.

# Chapter 7

# Appendix

This chapter contains the code implementation of our project.

```
def get_notes(songs):
    notes = []
    for file in songs:
        try:
            midi = converter.parse(file)
            notes_to_parse = []
            try:
                Score = instrument.partitionByInstrument(midi)
            except:
                pass
            if Score and len(Score)>1:
                notes_to_parse = Score.parts[1].recurse()
            else:
                notes_to_parse = midi.flat.notes

            for element in notes_to_parse:
                if isinstance(element, note.Note):
                    element={"pitch": str(element.pitch),
                    "duration": str(element.duration.quarterLength),
                    "played":1 }
                    notes.append(element)
                elif isinstance(element, chord.Chord):
                    normalOrderChord='.'.join(str(n)
                    for n in element.normalOrder)
                    element={"pitch":  normalOrderChord,
                    "duration": str(element.duration.quarterLength),
                    "played":1 }
                    notes.append(element)
                elif isinstance(element, note.Rest):
                    element={"pitch": "r",
                    "duration": str(element.duration.quarterLength),
                    "played":1 }
                    notes.append(element)
        except Exception as e:
            print(f"Error parsing MIDI file {file}: {e}")
            continue
    # Save the notes to a file
    with open('Data/notes.json', 'w') as filepath:
        json.dump(notes, filepath)

    return notes

def prepare_duration_dictionary(notes):
     durations = [element["duration"] for element in notes]
     duration_names = sorted(set(item for item in durations))
     unique_duration_num=len(duration_names)
     duration_to_int = {note: value for value,
     note in enumerate(duration_names, start=0)}

     return  unique_duration_num,duration_to_int

def prepare_pitch_dictionary(notes):
     pitches = [element["pitch"] for element in notes]
     pitch_names = sorted(set(item for item in pitches))
     unique_count=len(pitch_names)
     pitch_to_int = dict((note, number) for number,
     note in enumerate(pitch_names))

     return unique_count,pitch_to_int

def transform_duration_value(duration):
   num=1
   base_value= 1/16
   middle=1/32
   decimalValue=1.0
   while(duration > base_value):
    duration=duration-base_value
    num=num+1
    decimalValue=duration
   if(decimalValue<middle):
    num=num-1
   duration=num/16
   return duration

def prepare_duration_array(notes):
  durations = [element["duration"] for element in notes]
  #transform to real numbers
  for note in notes:
    note["duration"]=round(eval(note["duration"]),2)
  #transform to 1/16 based values
  base_value= 1/16
  comparing_value=0.001
  for index, note in enumerate(notes):
    duration=note["duration"]
    if(duration<1):
      new_value=transform_duration_value(duration)
      note["duration"] = new_value
    else:
      #create new notes, min 2
      is_it_float=0
      additional_value=duration-int(duration)
      if(additional_value>0):
         is_it_float=1
         #additional_value=round(additional_value,2)
      numberOfNotes=int(duration)+ is_it_float
      numberOfInsertedNotes=numberOfNotes-1
      note["duration"]=1
      for i in range(0,numberOfInsertedNotes-1):
        element={"pitch": note["pitch"], "duration": 1, "played":0 }
        notes.insert(index+1,element)
      if(is_it_float):
        new_duration=transform_duration_value(additional_value)
        element={"pitch": note["pitch"], "duration": new_duration, "played":0 }
        notes.insert(index+numberOfInsertedNotes,element)


def prepare_sequences(notes):
    sequence_length = 50
    prepare_duration_array(notes)
    unique_pitch_num, pitch_dictionary=prepare_pitch_dictionary(notes)
    unique_duration_num,duration_dictionary=prepare_duration_dictionary(notes)
    network_input = []
    network_out=[]
    network_output_played=[]
    network_output_pitch=[]
    network_output_duration=[]
    pitch_network_input=[]
    duration_network_input=[]
    played_network_input=[]
    for i in range(0, len(notes) - sequence_length, 1):
        output_element=[]
        notes_sequence = notes[i: i + sequence_length]
        sequence_in = []
        pitch_sequence_in = []
        duration_sequence_in = []
        played_sequence_in=[]
        for note in notes_sequence:
            pitch_sequence_in.append(pitch_dictionary[note["pitch"]])
            duration_sequence_in.append(duration_dictionary[note["duration"]])
            played_sequence_in.append(note["played"])
        pitch_network_input.append(pitch_sequence_in)
        duration_network_input.append(duration_sequence_in)
        played_network_input.append(played_sequence_in)

        output_pitch=pitch_dictionary[notes[i + sequence_length]["pitch"]]
        output_duration=duration_dictionary[notes[i + sequence_length]
        ["duration"]]

        output_played=notes[i + sequence_length]["played"]
        length=unique_duration_num+unique_pitch_num+1

        output_pitch= to_categorical(output_pitch, num_classes=unique_pitch_num)
        output_duration=to_categorical(output_duration,
        num_classes=unique_duration_num)
        output_played=to_categorical(output_played, num_classes=2)

        #output arrays

        network_output_pitch.append(output_pitch)
        network_output_duration.append(output_duration)
        network_output_played.append(output_played)

    network_input.append(pitch_network_input)
    network_input.append(duration_network_input)
    network_input.append(played_network_input)

    network_output_pitch=np.array(network_output_pitch)
    network_output_duration=np.array(network_output_duration)
    network_output_played=np.array(network_output_played)

    print(network_input[0][0])
    print(network_input[1][0])
    print(network_input[2][0])

    return  (network_input, network_output_pitch,
    network_output_duration,network_output_played)

def create_network(unique_pitch_num, unique_duration_num, num_of_sequences):
    pitchInput = Input(shape=(50, 1))
    durationInput = Input(shape=(50, 1))
    playedInput = Input(shape=(50, 1))

    x = LSTM(128, return_sequences=True)(pitchInput)
    y = LSTM(64, return_sequences=True)(durationInput)
    z = LSTM(64, return_sequences=True)(playedInput)

    combined = concatenate([x, y, z])
    w = LSTM(128, return_sequences=True, recurrent_dropout=0,
    activation="tanh", recurrent_activation="sigmoid")(combined)
    w = Dropout(0.2)(w)
    w = LSTM(128, return_sequences=True, recurrent_dropout=0,
    activation="tanh", recurrent_activation="sigmoid")(w)
    w = Flatten()(w)
    w = Dense(256, activation='relu')(w)
    w = Dropout(0.3)(w)

    pitch_output = Dense(unique_pitch_num, activation='softmax')(w)
    duration_output = Dense(unique_duration_num, activation='softmax')(w)
    played_output = Dense(2, activation='softmax')(w)
    model = Model(inputs=[pitchInput, durationInput, playedInput],
                  outputs=[pitch_output, duration_output, played_output])

    optimizer = Adam(learning_rate=0.001)
    model.compile(optimizer=optimizer,
                  loss=['categorical_crossentropy', 'categorical_crossentropy',
                  'binary_crossentropy'],
                  metrics=[Accuracy()])

    return model


def normalize_data(data):
  scaler = MinMaxScaler()
  normalized_data = scaler.fit_transform(data)
  return normalized_data

def train(model, network_input, network_output_pitch, network_output_duration,
network_output_played, epochs):
    # Create checkpoint to save the best model weights.
    filepath = 'weights.hdf5'
    checkpoint = ModelCheckpoint(filepath, monitor='loss', verbose=0,
    save_best_only=True)
    pitch_input=normalize_data(network_input[0])
    duration_input=normalize_data(network_input[1])
    played_input=normalize_data(network_input[2])
    print(duration_input[0])
    with tf.device(device_name):
      history = model.fit(
          [pitch_input, duration_input, played_input],
          [network_output_pitch, network_output_duration, network_output_played]
          epochs=epochs, batch_size=50, callbacks=[checkpoint]
    )
    return history

def train_model():
    epochs = 200
    notes = json.load(open('./Data/notes.json'))
    print('Notes processed')
    network_input, network_output_pitch, network_output_duration,
    network_output_played = prepare_sequences(notes)

    num_of_sequences = len(network_input[0])
    unique_pitch_num = len(set([item['pitch'] for item in notes]))
    unique_duration_num = len(set([item['duration'] for item in notes]))

    print('Input and Output processed')

    with tf.device(device_name):
      model = create_network(unique_pitch_num, unique_duration_num,
       num_of_sequences)
        # Visualize the model architecture
    plot_model(model, to_file='model.png', show_shapes=True,
     show_layer_names=True)

    # Display the image inline
    Image('model.png')
    print('Model created')

    print('Training in progress')
    print(model.summary())
    history=train(model,network_input, network_output_pitch,
     network_output_duration, network_output_played, epochs)
    print('Training completed')

    # Visualize training loss
    plt.plot(history.history['loss'])
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.show()

    return model


def generate_notes(model, network_input, pitch_dictionary, duration_dictionary,
  unique_pitch_num, unique_duration_num):

    start = np.random.randint(0, len(network_input)-1)
    int_to_pitch = dict((number, note) for number,
     note in enumerate(pitch_dictionary))
    int_to_duration = dict((number, note) for number,
     note in enumerate(duration_dictionary))

    pitch_input=network_input[0][start]
    duration_input=network_input[1][start]
    played_input=network_input[2][start]

    pitch_input=np.squeeze(pitch_input)
    duration_input=np.squeeze(duration_input)


    pitch_prediction=[]
    duration_prediction=[]
    played_prediction=[]
    prediction_output=[]


    list_pitch=[]
    list_duration=[]
    for note_index in range(150):


        batch_size = 1
```

```
time_steps = 50
input_features = 1


pitch_input_normalized=normalize_data(np.reshape(pitch_input, (-1, 1)))
duration_input_normalized=normalize_data(np.reshape(duration_input,
 (-1, 1)))
played_input_normalized=played_input

pitch_input_reshaped = np.reshape(pitch_input_normalized, (batch_size,
 time_steps, input_features))
duration_input_reshaped = np.reshape(duration_input_normalized,
 (batch_size, time_steps, input_features))
played_input_reshaped = np.reshape(played_input_normalized,
 (batch_size, time_steps, input_features))

input_list = [pitch_input_reshaped, duration_input_reshaped,
 played_input_reshaped]

prediction = model.predict(input_list, verbose=0)

random_integer = random.randint(-3,-1)

pitch = prediction[0].ravel()
duration = prediction[1].ravel()

sorted_pitch = np.sort(pitch)
sorted_duration = np.sort(duration)

pitch_to_find = sorted_pitch[random_integer]
duration_to_find= sorted_duration[random_integer]

pitch_result = np.where(pitch == pitch_to_find)[0][0]
duration_result=np.where(duration == duration_to_find)[0][0]

list_pitch.append(pitch_result)
list_duration.append(duration_result)

if note_index==0:
  played=1
else:
  list_of_candidates=[1,0]
  previous_pitch=list_pitch[note_index-1]
  if previous_pitch == pitch_result:
    probability_distribution=[0.15, 0.85]
  else:
    probability_distribution=[1.00, 0.00]


  played=choice(list_of_candidates, 1,
        p=probability_distribution)
```

```
            print(played)

            # Mapping the predicted interger back to the corresponding note
            pitch = int_to_pitch[pitch_result]
            duration = int_to_duration[duration_result]


            pitch_prediction = np.append(pitch_prediction, pitch)
            duration_prediction = np.append(duration_prediction, duration)
            played_prediction = np.append(played_prediction, played)

            #add predicted value
            pitch_input=np.append(pitch_input,pitch_result)
            duration_input=np.append(duration_input,duration_result)
            played_input=np.append(played_input,played)

            # Next input to the model
            pitch_input = pitch_input[1:len(pitch_input)]
            duration_input = duration_input[1:len(duration_input)]
            played_input = played_input[1:len(played_input)]

        prediction_output.append(pitch_prediction)
        prediction_output.append(duration_prediction)
        prediction_output.append(played_prediction)

        print('Notes Generated...')

        return prediction_output


def generate():
    current_directory = os.getcwd()
    print("Current Directory:", current_directory)
    notes = json.load(open('./Data/notes-bezRest.json'))
    print('Notes processed')
    network_input, network_output_pitch,
      network_output_duration, network_output_played = prepare_sequences(notes)
    unique_pitch_num,pitch_dictionary=prepare_pitch_dictionary(notes)
    unique_duration_num,duration_dictionary=prepare_duration_dictionary(notes)
    network_input=np.array(network_input)
    num_of_sequences = len(network_input[0])
    print('Input and Output processed')

    with tf.device(device_name):
      model = create_network(unique_pitch_num, unique_duration_num,
       num_of_sequences)
    print('Loading Model weights.....')
    os.chdir('./Models')
    current_directory = os.getcwd()
    print("Current Directory:", current_directory)
    model.load_weights('model1_weights.hdf5')
    print('Model Loaded')
    os.chdir('..')
    prediction_output = generate_notes(model, network_input,
      pitch_dictionary,duration_dictionary, unique_pitch_num, unique_duration_nu

    create_midi(prediction_output)

def create_midi(prediction_output):
    offset = 0
    output_notes = []

    # create note and chord objects based on the values generated by the model
    output_notes_index=-1
    for index, pattern in enumerate(prediction_output[0]):
       note_duration = prediction_output[1][index]
       played=prediction_output[2][index]
       if(played==0):
          print(output_notes_index)
          print("--")
          print(len(output_notes))
          offset += prediction_output[1][index]
          old_duration= prediction_output[1][index-1]
          new_duration= old_duration+note_duration
          output_notes[output_notes_index].duration =
           duration.Duration(new_duration)
          continue
       else:
          output_notes_index=output_notes_index+1
          if pattern == 'r':
            # Handle rests

            new_rest = note.Rest()
            new_rest.duration = duration.Duration(note_duration)
            new_rest.offset = offset
            output_notes.append(new_rest)
        # pattern is a chord
          elif('.' in pattern) or pattern.isdigit():
              notes_in_chord = pattern.split('.')
              notes = []
              for current_note in notes_in_chord:
                  new_note = note.Note(int(current_note))
                  new_note.storedInstrument = instrument.Piano()
                  notes.append(new_note)
              new_chord = chord.Chord(notes)
              new_chord.duration = duration.Duration(note_duration)
              new_chord.offset = offset
              output_notes.append(new_chord)
          # pattern is a note
          else:
              new_note = note.Note(pattern)
```

```
                new_note.offset = offset
                new_note.duration = duration.Duration(note_duration)
                new_note.storedInstrument = instrument.ElectricGuitar()
                output_notes.append(new_note)

            # increase offset each iteration so that notes do not stack
            offset += prediction_output[1][index]


    midi_stream = stream.Stream(output_notes)

    folder_path = '/content/ZavrsniRad/GeneratedSongs'
    number = len(os.listdir(folder_path))+10
    filename = f"test{number}.mid"
    midi_stream.write('midi', fp=filename)

with tf.device(device_name):
    model=train_model()
generate()
```

# Bibliography

[1] Zhang, S., (ur.), Emotional Responses to Music: Shifts in Frontal Brain Asymmetry Mark Periods of Musical Change, 2022.

[2] Rickard1, H.-A. A. H. P. S., (ur.), The Positive Influence of Music on the Human Brain, Ashley Hall, Charleston, SC, USA., 2022.

[3] https://web.mit.edu/music21/doc/.

[4] M., H. J. . L. A. . I. . L., (ur.), Musical composition with a high speed digital computer. In Audio engineering society convention 9. Audio Engineering Society., 1957.

[5] Price, J. B. . C. M. P. R. A. J. C., (ur.), Library Carpentry: software skills training for library professionals. Liber Quarterly: The Journal of European Research Libraries, 26(3), 141-162., 2016.

[6] Wu, A. H., (ur.), Deep Learning for Music, Stanford, 2016.

[7] s. Priyanka Wankar, M. S. M., (ur.), Research Paper on Basic of Artificial Neural Network, Datta Meghe Institute of Engineering, Technology  Research, Sawangi (M), Wardha, 2014.

[8] Nápoles, G. V. H. M., (ur.), A Review on the Long Short-Term Memory Model, 2020.

[9] Sharma, S. S., (ur.), Activation functions in Neural Network, UG Scholar, Dept. of Computer Science and Engineering, Global Institute of Technology, Jaipur, 2020.

[10] https://www.britannica.com/art/pitch-music.