

**MONOPHONIC MUSIC GENERATION USING
ARTIFICIAL INTELLIGENCE THROUGH DEEP
LEARNING TECHNIQUES**

**A THESIS SUBMITTED TO
THE INSTITUTE OF GRADUATE STUDIES
OF
NEAR EAST UNIVERSITY**

**By
ARA AHMED SHARIF**

**In Partial Fulfilment of the Requirements for
the Degree of Master of Science
in
Computer Engineering**

NICOSIA, 2021

**ARA AHMED
SHARIF**

**MONOPHONIC MUSIC GENERATION USING ARTIFICIAL
INTELLIGENCE THROUGH DEEP LEARNING TECHNIQUES**

**NEU
2021**

**MONOPHONIC MUSIC GENERATION USING
ARTIFICIAL INTELLIGENCE THROUGH DEEP
LEARNING TECHNIQUES**

**A THESIS SUBMITTED TO
THE INSTITUTE OF GRADUATE STUDIES
OF
NEAR EAST UNIVERSITY**

**By
ARA AHMED SHARIF**

**In Partial Fulfilment of the Requirements for
the Degree of Master of Science
in
Computer Engineering**

NICOSIA, 2021

Ara Ahmed Sharif: MONOPHONIC MUSIC GENERATION USING ARTIFICIAL INTELLIGENCE THROUGH DEEP LEARNING TECHNIQUES

Approval of Director of Institute of Graduate Studies

Prof. Dr. K. Hüsnü Can Başer

**We certify this thesis is satisfactory for the award of the degree of Master of Science
in Computer Engineering**

Examining Committee in Charge:

Assoc. Prof. Dr. Kamil
DİMİLİLER

Committee Chairman, Department of Automotive
Engineering, NEU

Assoc. Prof. Dr. Boran
ŞEKEROĞLU

Committee member, Department of Information
Systems Engineering, NEU

Prof. Dr. Rahib Abiyev

Committee member, Supervisor, Department of
Computer Engineering, NEU

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: Ara Sharif

Signature: 

Date:

ACKNOWLEDGEMENTS

Music and computer engineering science are two of my passions that I wanted to merge in this master thesis topic which my supervisor Prof. Dr. Rahib Abiyev encouraged me to deal with it, many thanks to him for all his valuable time and great advice. I want to sincerely thank all Examining Committee for agreeing to offer their time for evaluating and commenting on my thesis.

My deep thanks to all my professors and teachers during my academic career, especially those who contributed to enlightening my way to reach this academic stage of studying for a master's degree: Prof. Dr. Fadi Al-Turjman, Assist. Prof. Dr. Melike Sah Direkoglu, Assist. Prof. Dr. Kaan Uyar, Assist. Prof. Dr. Elbrus İmanov, Assoc. Prof. Dr. Parham Moradi, Assoc. Prof. Dr. Fardin Akhlaghian Taba and Assoc. Prof. Dr. Sadoon Azizi.

Special thanks are due to music master: Ako Aziz Eynayat for his valuable effort when he accepted and analyzed generated melodic samples with his deep musical experience. I would like to thank all the participants in the human evaluation of the melodies generated from the models; their contribution gave value to this project.

Last but not least I am especially thankful to all my family members for their strong support and consistent trust.

To who said: “I see my life in terms of music”...

ABSTRACT

This thesis shows how to use deep learning, which is a branch of Artificial Intelligence (AI), in the field of music generation, especially monophonic melodies. Two of the most common and improved architectures of the Recurrent Neural Network have been used; Long Short Term Memory (LSTM) and Gated Recurrent Units (GRUs) with the option of using Bidirectionality and Attention mechanism to train on two types of MIDI format dataset, one is stylistically single-genre (Folk) and the other multi-genre (folk, pop, rock, etc.), each encoded with two formats; pianoroll and magenta melody. The main purpose of this work is to reveal the effect of architecture designs, dataset specifications, and encoding formats on the generated melody samples. We discussed the side-by-side comparison between LSTM and GRUs, as well as the influence of modifying hidden layers has been investigated. Objectively, with the assistance of professional composers and expert musicians in my reachable area, we carried out the analysis of the generated melody samples by the different datasets and models, although the impact of dataset types, architecture designs, Bidirectionality, and Attention mechanisms on the generated melodies have discussed. The most noticeable results after experiments: For better learning with a multi-genre dataset we need more extra training sequences. While applying the Bi-directional LSTM with Attention mechanism on a single-genre folk dataset we obtained more pleasant emotional melodies. Bidirectionality and Attention mechanisms both improve learning. Generated samples with the magenta melody format encoding have melodic characteristics but pianoroll formatting generated more rhythmically samples. Subjectively, human evaluation has been made on the samples of the best model.

Keywords: Artificial Intelligence; deep learning; monophonic music generation; RNN; LSTM; GRUs; bi-directional RNNs; attention mechanism.

ÖZET

Bu tez, Yapay Zekanın (AI) bir dalı olan derin öğrenmenin müzik üretimi alanında, özellikle de monofonik melodiler alanında nasıl kullanılacağını göstermektedir. Yinelenen Sinir Ağının en yaygın ve geliştirilmiş mimarilerinden ikisi kullanılmıştır; Uzun Kısa Süreli Bellek (LSTM) ve Geçitli Tekrarlayan Birimler (GRU'lar) iki tür MIDI formatı veri kümesi üzerinde eğitmek için Çift Yönlü ve Dikkat mekanizmasını kullanma seçeneği ile, biri biçimsel olarak tek tür (Halk) ve diğeri çok tür (folk) , pop, rock, vb.), her biri iki formatla kodlanmıştır; piyanorol ve eflatun melodi. Bu çalışmanın temel amacı, mimari tasarımların, veri seti spesifikasyonlarının ve kodlama formatlarının üretilen melodi örnekleri üzerindeki etkisini ortaya çıkarmaktır. LSTM ve GRU'lar arasındaki yan yana karşılaştırmanın yanı sıra, gizli katmanları değiştirmenin etkisi de araştırıldı. Öznel olarak, ulaşılabilir alanındaki profesyonel besteciler ve uzman müzisyenlerin yardımıyla, üretilen melodi örneklerinin analizini farklı veri seti ve mimari modellerle gerçekleştirdik, buna rağmen veri seti türlerinin, mimari tasarımların, çift yönlülüğün ve dikkatin etkisi. üretilen melodiler üzerine mekanizmalar tartıştık. Deneylerden sonra en dikkat çekici sonuçlar: Çok türden bir veri kümesiyle daha iyi öğrenme için daha fazla ekstra eğitim dizisine ihtiyacımız var. Dikkat mekanizmalı çift yönlü LSTM tek bir tür folk veri setine uygularken daha hoş duygusal melodiler elde ettik. Çift yönlülük ve dikkat mekanizmaları öğrenmeyi geliştirir. Macenta melodi formatı kodlamasıyla oluşturulan örnekler melodik özelliklere sahiptir, ancak piyano rulosu formatlaması daha ritmik örnekler oluşturmuştur. Sübjektif olarak, en iyi modelin örnekleri üzerinde insan değerlendirmesi yapılmıştır.

Anahtar Kelimeler: Yapay zeka; derin öğrenme; monofonik müzik üretimi; RNN; LSTM; GRU'lar; çift yönlü RNN'ler; dikkat mekanizması.

TABLE OF CONTENTS

ACKNOWLEDGEMENT.....	i
ABSTRACT.....	iii
ÖZET.....	iv
TABLE OF CONTENTS.....	v
LIST OF TABLES.....	viii
LIST OF FIGURES.....	ix
LIST OF ABBREVIATIONS AND SYMBOLS.....	xi
 CHAPTER 1: INTRODUCTION	
1.1 Music Composition.....	2
1.2 Problem Statement.....	2
1.3 Motivation.....	4
1.4 Structure.....	4
 CHAPTER 2: BACKGROUND AND STATE OF THE ART	
2.1 Music Theory.....	5
2.1.1 The basic building elements of music.....	5
2.1.2 Monophonic and polyphonic music.....	8
2.2 Artificial Intelligence.....	10
2.3 Machine Learning.....	11
2.4 Deep Learning.....	12
2.4.1 Feedforward neural networks.....	12
2.4.2 Recurrent neural networks.....	13
 CHAPTER 3: METHODOLOGY & ARCHITECTURE	
3.1 LSTM and GRUs.....	29
3.2 LSTM vs GRUs.....	30
3.3 Gradient Descent Optimization Techniques.....	30
3.4 Bi-directional Mechanism.....	32
3.5 Attention Mechanism.....	32

3.6	Architecture Type.....	33
3.7	Temperature.....	34
3.8	Human Evaluations: Professionally and Conventionally.....	35
3.9	Trained Models: Architecture Description.....	35
3.9.1	Model No.1: model_folk_melody_bi2lstm64_attention.....	48
3.9.2	Model No.2: model_folk_pianoroll_bi2lstm64_attention.....	40
3.9.3	Model No.3: model_hook_melody_bi2lstm64_attention.....	41
3.9.4	Model No.4: model_hook_pianoroll_bi2lstm64_attention.....	42
3.9.5	Model No.5: model_folk_melody_2lstm64_attention.....	43
3.9.6	Model No.6: model_folk_melody_2lstm64_noattention.....	45
3.9.7	Model No.7: model_folk_melody_bi2GRU64_attention.....	56

CHAPTER 4: COLLECTING AND PROCESSING DATA

4.1	Data set.....	48
4.2	Multi-Genre Dataset.....	49
4.3	Single-Genre Dataset.....	49
4.4	Music Representation Formats.....	49
4.4.1	ABC format representation.....	50
4.4.2	Simplified Plaine & Easie format representation.....	50
4.4.3	Kern format representation.....	51
4.4.4	MusicXML format representation.....	52
4.4.5	MIDI format representation.....	52
4.5	Format Encoding.....	55
4.5.1	Pianoroll format encoding.....	55
4.5.2	Magenta melody format encoding.....	56
4.6	Preprocessing.....	57

CHAPTER 5: RESULTS AND DISCUSSIONS

5.1	Results.....	60
5.1.1	Format encoding impacts.....	60
5.1.2	Dataset nature impacts (Single-genre vs Multi-genre).....	64

5.1.3 RNN type impacts (LSTM vs GRUs).....	67
5.1.4 Bi-directional mechanism impacts.....	69
5.1.5 Attention mechanism impacts.....	72
5.1.6 Bi-directional & Attention impacts.....	74
5.2 Human Evaluation.....	76
5.2.1 Survey analysis and results.....	77
CHAPTER 6: CONCLUSION.....	83
REFERENCES.....	85
APPENDICES	
Appendix 1: Ethical Approval Document.....	96
Appendix 2: Similarity Report.....	97
Appendix 3: Workflow and Implemented Codes Samples.....	98
Appendix 4: Screenshot Samples for Implementations.....	121
Appendix 5: Survey Resources.....	123

LIST OF TABLES

Table 2.1:	Architectural types of Recurrent Neural Networks.....	16
Table 3.1:	Models architecture per changes in the independent variables.....	37
Table 5.1:	Experts objectively comparison for revealing the encoding influences.	62
Table 5.2:	Experts objectively comparison for revealing the dataset nature influences.....	65
Table 5.3:	Experts objectively comparison for revealing the Bidirectionality influences.....	70
Table 5.4:	Evaluation results of the participants per group experience level.....	78
Table 5.5:	Evaluation results of all participants.....	80
Table 5.6:	Evaluation results of all participants per each experience level.....	81

LIST OF FIGURES

Figure 1.1:	Input independent variables vs output dependent variable.....	3
Figure 2.1:	Note length symbols represent duration in sheet music.....	6
Figure 2.2:	Key numbers and their pitch-register designation.....	7
Figure 2.3:	A key signature of G Major/E Minor.....	8
Figure 2.4:	An example of a sequence.....	8
Figure 2.5:	Visual diagram and music notation sample of monophonic texture	9
Figure 2.6:	Visual diagram and music notation sample of monophonic texture	9
Figure 2.7:	The relationship between different AI disciplines.....	11
Figure 2.8:	An example of Multilayer Perceptron.....	13
Figure 2.9:	An example of a Recurrent Neural Network.....	14
Figure 2.10:	Unfolded Recurrent Neural Network.....	15
Figure 2.11:	Gradient descent algorithm uses derivatives of a function to follow the function downhill.....	18
Figure 2.12:	Local and global minimum reaching challenge of gradient descent	18
Figure 2.13:	Comparison of Adam to Other Optimization Algorithms.....	19
Figure 2.14:	Dropout Strategy. (a) A standard neural network. (b) Applying dropout to the neural network on the left by dropping the crossed units.....	20
Figure 2.15:	Long Short-term Memory Cell.....	21
Figure 2.16:	Gated Recurrent Unit.....	23
Figure 2.17:	The general structure of the Bi-directional recurrent neural network (BI-RNN) unfolded in time for three-time steps.....	24
Figure 2.18:	Bi-directional Recurrent Neural Network.....	25
Figure 2.19:	The encoder-decoder model with additive attention mechanism.....	26
Figure 2.20:	Alignment matrix of “L’accord sur l’Espace économique européen a été Signé en août 1992” translation.....	27
Figure 2.21:	Matrix of alignment illustrates the association between the source and target words.....	27
Figure 3.1:	Music generation many-to-one RNN architecture type.....	34
Figure 3.2:	Generated melody sample converted from MIDI file.....	34
Figure 3.3:	Architecture design of model No.1.....	39
Figure 3.4:	Architecture design of model No.2.....	40

Figure 3.5:	Architecture design of model No.3.....	41
Figure 3.6:	Architecture design of model No.4.....	42
Figure 3.7:	Architecture design of model No.5.....	44
Figure 3.8:	Architecture design of model No.6.....	45
Figure 3.9:	Architecture design of model No.7.....	47
Figure 4.1:	ABC format representation.....	50
Figure 4.2:	Simplified Plaine & Easie Code representation.....	51
Figure 4.3:	Kern representation format.....	51
Figure 4.4:	MusicXML format representation.....	52
Figure 4.5:	A combination of notation and its pianoroll representation.....	53
Figure 4.6:	MIDI Note Numbers for Different Octaves.....	54
Figure 4.7:	Each MIDI number is equivalent to an octave in the left-hand column and a note in the top row.....	56
Figure 4.8:	Preprocessing workflow.....	59
Figure 5.1:	Loss graphs for models No.1 and No.2.....	61
Figure 5.2:	Seed and generated samples for showing format encoding impacts	63
Figure 5.3:	Loss graphs for models No.1 and No.3.....	64
Figure 5.4:	Seed and generated samples to show dataset nature impacts.....	66
Figure 5.5:	Loss graphs for models No.1 and No.7.....	67
Figure 5.6:	Seed and generated samples to demonstrate the RNN type impacts	68
Figure 5.7:	Loss graphs for models No.1 and No.5.....	69
Figure 5.8:	Seed and generated samples to demonstrate Bidirectionality impacts.....	71
Figure 5.9:	Loss graphs for models No.5 and No.6.....	72
Figure 5.10:	Seed and generated samples to demonstrate Attention mechanism impacts.....	73
Figure 5.11:	Loss graphs for models No.1 and No.6.....	74
Figure 5.12:	Seed and generated samples to show Bi-directional & Attention impacts.....	75
Figure 5.13:	Distribution of participants according to experience levels.....	77
Figure 5.14:	Participants' evaluation results per low and high experience level (group1 and group2).....	79
Figure 5.15:	The overall participant evaluation results.....	80
Figure 5.16:	Participants' evaluation results for each experience level.....	82

LIST OF ABBREVIATIONS AND SYMBOLS

AdaGrad:	Adaptive Gradient Algorithm
Adam:	Adaptive Moment Estimation
AI:	Artificial Intelligence
AIVA:	Artificial Intelligence Virtual Artist
Bi-GRUs:	Bi-directional Gated Recurrent Units
Bi-LSTM:	Bi-directional Long Short Term Memory
Bi-RNN:	Bi-directional Recurrent Neural Network
cuDNN:	GPU-accelerated library for Deep Neural Network
DL:	Deep Learning
GRUs:	Gated Recurrent Units
LSTM:	Long Short Term Memory
MIDI:	Musical Instrument Digital Interface
ML:	Machine learning
RMSProp:	Root Mean Square Propagation
RNNs:	Recurrent Neural Networks

CHAPTER 1

INTRODUCTION

Music is humanity's greatest creation and an important part of our lives. It surrounds us and reaches our ears through multiple sources, we listen to it when we study or when we are working. It is a way to express both our feelings and emotions, and it has obvious psychological effects on us (Sloboda, 2010). So this puts it in an interesting place to research.

Artificial Intelligence (AI) is the ability of machines to learn, make decisions, and perform tasks similar to humans; it is a core of the fourth industrial revolution, the way our daily lives look is affected by many daily impacts of AI, there's no doubt that it is an integral part of our daily lives. The first AI invention goes back to the 1950s and has exploded in recent years; owing to the huge amount of data we produce every day and the computational resources available (Marr, 2019). AI with its great deep learning techniques is an effective and important field to study, especially in the area of generating music; the greatest creation of mankind.

The programs that produce music have long been in history, the first attempts to produce melodies with computers date back to 1956 when Pinkerton designed Markov's first-order model (Pinkerton, 1956). Recently many researches indicated that deep learning techniques have significant efficiency when use to enforce long-term structure. In 1994 Mozer was used Recurrent Neural Networks (RNNs) to generate music in his work (Mozer, 1994).

In this project, we benefited from previous valuable works and researches in the field of music generation. Practically, we aided from (Mitroi, 2019; Velardo, 2020; Marinescu, 2019; Sigurgeirsson, 2020), and theoretically, we mentioned all sources in the context of explaining topics in their appropriate places.

1.1 Music Composition

When we talk about composing music, regardless of whether it was composed by a human or if the computer that generated it through its learning and training on previous mankind melodies and compositions as a database, in the end, it is for human listening or uses, this confirms that Generally, composing music is seen as a human-specific talent or skill. So a key aspect of music is structure. It explains how the various parts are positioned together in a piece of music to shape the composition. In music composing, *contrast*, *repetition*, and *continuity* are three main principles. A piece of music has a combination of musical thoughts. To develop continuity and a coherent whole, a composer needs to think carefully about how to repeat and contrast these ideas in parts (Ben, 2019). Obviously, good music has a cohesive structure, and it pleases the ear as well (De Coster, 2017).

The core of our research topic revolves around composing music using deep learning as one of AI techniques through using the computer and more specifically, generating a monophonic melody by using recurrent neural networks Long-Short Term Memory (LSTM) and Gated Recurrent Units (GRUs), So it is useful first to present summaries of some fundamentals related to music in general and closer to our topic in particular, and secondly, we need a brief presentation of topics related to AI and its techniques in general, and a detailed explanation as much as possible of the aspects related to generating monophonic music which is presented in the form of symbols more precisely. The details of this knowledge are explained in the second chapter of this work.

1.2 Problem Statement

The study and evaluation of the influence of the independent variables used in our work as inputs on the dependent variable outputs is the main problem addressed in this thesis and it is relevant to the core of our subject: Monophonic music generation via RNNs. We seek through this work to investigate the effects of the input on the output results and to define the dialectical relationship between them. Independent variables can be divided into:

1. Data types in terms of style (genre), as we used two data groups to train our systems to learn; data consisting of multi-genre melodies (jazz, rock, pop, classic,

- folk, etc.), and single-genre melodies (folk); "Musicmap" project claims that today there are nearly 234 genres of music (Musicmap, 2020). We explained this issue in detail in Chapter 4, in terms of dataset characteristics, and we presented information in Chapter 5, in terms of its impact on the performance of our system.
2. The type of data format and methods of encoding it into a language that is generally understood by a computer and that fits the architecture of the recurring neural networks specifically used in our work. We used Musical Instrument Digital Interface (MIDI) type data and two encoding formats: melody format encoding and pianoroll format encoding. There is also a summary of the methods of encoding and their influence on the results in Chapters four and six respectively.
 3. The types of recurrent neural network architectures and designs employed (LSTM and GRUs) with the use of Attention and Bi-directional techniques to analyze their impacts. In chapters two and five, explanations are available.

Another problem addressed in this study is the generation of a particular form of musical texture called monophonic music; it seems to be simpler to model since the generation of monophonic music is a two-dimensional issue: the dimension of time and pitch (De Coster, 2017). See Figure 1.1

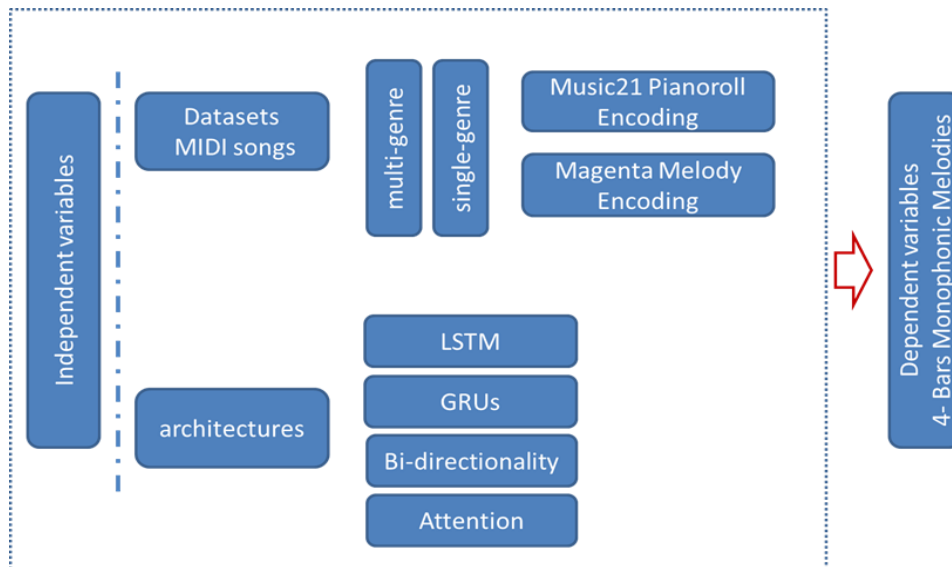


Figure 1.1: Input independent variables vs output dependent variable.

The overall goals can be summarized in finding the answers to these questions:

- 1- What is the influence of the dataset on the melody samples and their models?
- 2- How preprocessing dataset impact the training and the generating process?
- 3- How the most two popular RNN models LSTM and GRUs improve learning?
- 4- Do Bidirectionality and attention mechanisms have a positive impact on models?

1.3 Motivation

The main motivation for this project is my passion for music as a performer and sound engineer. Practically, I always touched on the power of the positive effect of technology on music; qualitatively and quantitatively. Technology in this creative field has always been a helpful tool for increasing beauty, quality, and accelerating production. Without technological development, we would not have kept this huge amount of audio and the visual archive of music. There is no doubt that the recent and continuous revolution in the field of AI and the rapid development of its techniques have brought us into the important field, which is the creation and generation of music by the computer as the human composer does. On the other hand, music as the fine arts, and its composition process also the prerogative of the human being, so I think that composing by the machine cannot be an alternative to the creative artistic composing process by humans, at least at this period that we are observing at in this area, but it can act as a powerful assistant for composers to find new and abundant motives, phrases and musical ideas.

1.4 Structure

This thesis is composed of six chapters, and the remainder of it organized as follows: Chapter 2 reviews the relevant researches to this project and focuses on the state of the art in the field of generating music using AI techniques. Different options of algorithms, processes, architectures and learning models are analyzed and are explored as a methodology in Chapter 3. Chapter 4 complemented all necessary information and required preprocessing on the datasets. The results of trained models analyzed objectively as well as subjectively are presented in Chapter 5. Finally, Chapter 6 is a conclusion. Here achieved results, points for improvement, and future directions have been summarized.

CHAPTER 2

BACKGROUND AND STATE OF THE ART

A detailed analysis of relevant researches applicable to this project will be provided in this current chapter, and discoveries and relevant concerns that are crucial to this work will be addressed. Using machine learning techniques as the main branch of AI, we presented powerful theories and backgrounds from deep learning and monophonic music generation. While this project is about music and generating melodies it is useful to explain some knowledge in music theory for more clarity.

2.1 Music Theory

To better understand this study dealing with the generation of monophonic melody; a very simple and important type of piece of music through Artificial Intelligence techniques, actually we need to know about some basic details in music theory, particularly those most relevant to music composition. Here we shall try to summarize most related topics from some of the best references, such as (Hosken, 2010; Levitin, 2006; Ben, 2019).

2.1.1 The basic building elements of music

As described by Levitin, (2006), actually while listening to the music we experience several attributes like tone (note), pitch, duration (rhythm), tempo, timbre, loudness, contour, spatial location, and reverberation, those are fundamental construction blocks of music.

Here's a short description of some of those most relevant to this work:

- *Tone and Note*: tone is the sound we can hear it but a note is a written symbol on a music notation sheet, the latter as shown in Figure 2.1, can be described as the basic symbol for sound and can be altered to signify length (duration) in several ways. The largest single value in common usage today is the note as a whole (O);

other notes have fractional relationships with the note as a whole and obtain one-half its value, one-quarter its value, etc.

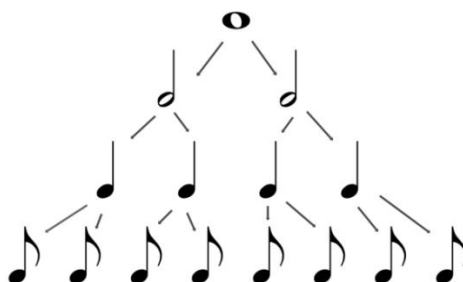


Figure 2.1: Note length symbols represent duration in sheet music.

- *Pitch*: mental construct, respectively relevant to the real frequency of a specific tone; the frequency at which a given note vibrates (measures by Hertz which is equal to the number of vibrations per seconds of the sound-producing device such as string vibration of such string instrument or wind instrument air column vibration), and its relative musical scale place; the note that plays on the musical instrument (e.g. playing A4 on the piano keyboard which has 440 Hz frequency) (Levitin, 2006; Ammer, 2004).
- *Pitch class*: functionally the category of all pitches connected to the octave equivalence. In other words, it is the set of pitches related to each other by octaves. A4, A3, A2, etc. are all members of pitch class A. Twelve pitch classes A, A# or Bb, B, C, C# or Db, D, D# or Eb, E, F, F# or Gb, G, G# or Ab, constitute the color (chromatic) music scale: (Itoh et al., 2019), Figure 2.2.

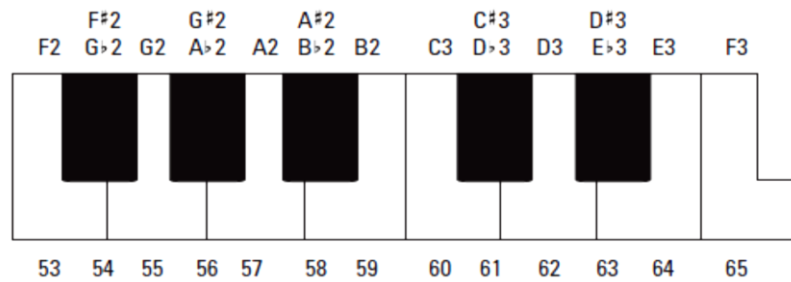


Figure 2.2: Key numbers and their pitch-register designation (middle C is C3) (Hosken, 2010).

- *Rhythm*: the steady or unsteady knocks arrangement produces strong and weak beats in music.
- *Duration*: shows the length of the sequence of notes, and how they organize into groups.
- *Tempo*: The beat speed which states the piece's total velocity or speed; it typically remains constant in western music for a given passage, though it may vary from one section to the next (Henry et al., 2018).
- *Key signature*: Music in Major scale gives joyful sense while Minor gives listener unhappy feeling. In music notation or lead sheets Key signature represents the mood of the music piece through some sharp or flat symbols as shown in Figure 2.3 we have a sharp sign on the fifth line and it means that the music is on G major scale and has somehow a happy mood.
- *Sequence*: “more or less exact repetition of a passage at a higher or lower level of the pitch” (Kennedy, 2013). A sample is shown in Figure 2.4.
- *Tonality*: the key signature determines the tonality of the music piece.
- *Texture*: Monophonic; music with a single melody, Homophonic; music with one melody and cords, Polyphonic music with more than one melody, and Heterophonic are types of texture.

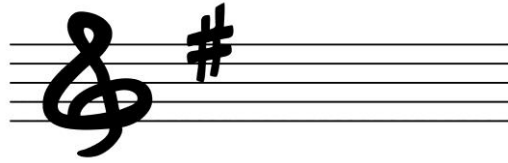


Figure 2.3: key signature of G Major/E Minor.

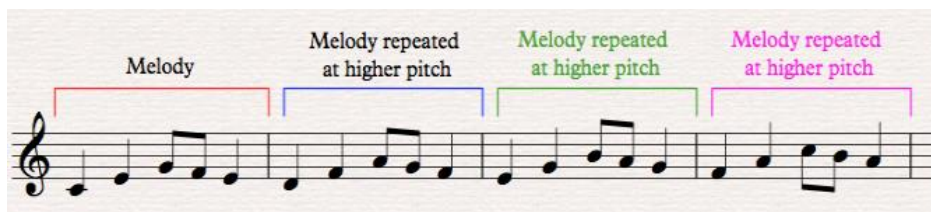


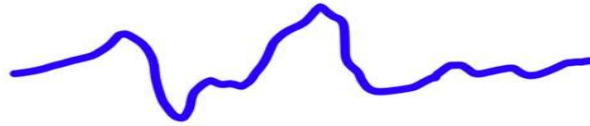
Figure 2.4: An example of a sequence; the first bar phrase transposed and repeated.

2.1.2 Monophonic and polyphonic music

The Greek meaning of monophonic music means “one sound”. Without any harmony or any other form of accompaniment, a monophonic texture has a single line of melody (Ben, 2019), Figure 2.5.

The Greek (poly-phonic) meaning of polyphonic texture means "many sounds". This distinguishes music in which many pieces or voices are blended in counterpoint (Ben, 2019), Figure 2.6.

Monophonic



THE STARS AND STRIPES

JOHN STAFFORD SMITH



Figure 2.5: Visual diagram and music notation sample of monophonic texture.

Polyphonic

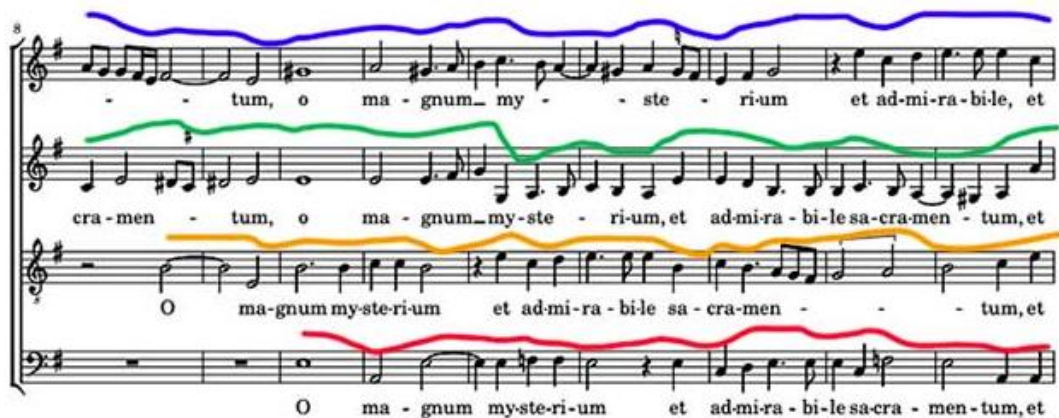
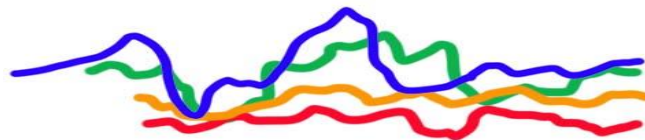


Figure 2.6: Visual diagram and music notation sample of monophonic texture.

2.2 Artificial Intelligence

AI can be distinct as "The science and engineering of making intelligent machines", according to the inventor of the expression "Artificial Intelligence" John McCarthy in 1956.

Frank Chen, (2016), also had a very clear and logical definition; to attempt to imitate human intelligence, Artificial Intelligence is a collection of algorithms and intelligence. Most of them are machine learning, and one of the machine learning techniques is deep learning. Simply put, AI is a machine's ability through many techniques or algorithms to mimic intelligent human behavior (McClelland, 2017). Figure 2.7 explains the relationship between these different AI disciplines.

Researches on AI as a Music Composer started decades ago, and since then, many companies have been aggressively implementing AI technologies that can compose music without human intervention. A machine learning algorithm on Beatles songs was trained by Sony's AI system to compose the song 'Daddy's Car', enabling the technology to compose a song based on what it had learned (Globant, 2017). Artificial Intelligence Virtual Artist (AIVA) is one of the most popular AI music compositions. They're focusing on creating classical music at the moment. AIVA, along with TensorFlow deep learning algorithms, uses a GPU-accelerated library of Deep Neural Network (cuDNN) which is programmed with reinforcement learning techniques (AIVA, 2020).

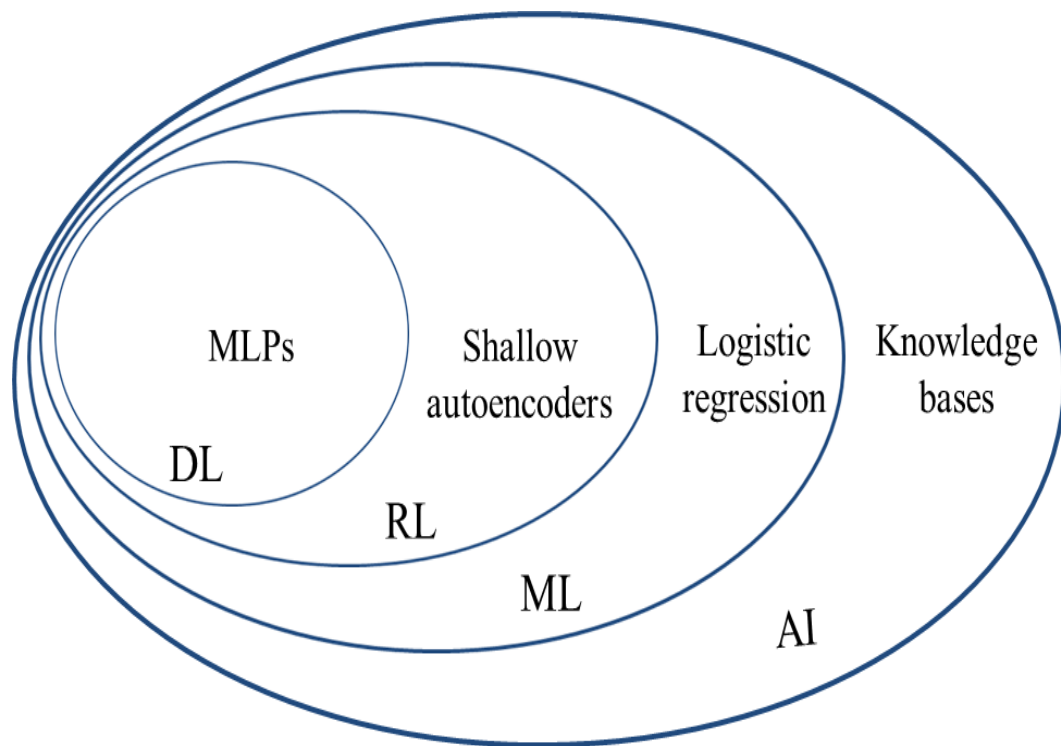


Figure 2.7: The relationship between different AI disciplines (Goodfellow et al., 2016).

2.3 Machine Learning

ML is a subset of AI techniques that allows computers without being directly programmed to execute tasks. It is based on models and techniques taken from statistics and the theory of probability (White, 2019). Murphy, (2012) defined Machine Learning as is a collection of techniques that can discover patterns in data automatically and then use the discovered patterns to forecast future data or conduct other forms of decision-making under uncertainty.

2.4 Deep Learning

DL is a subset of ML which focuses on identifying data patterns rather than solving any particular problem, it became popular in 2012. Goodfellow et al., (2016) defined DL principally as ML branch and computer systems booster. In dynamic, real-world conditions, DL is the mere developer technique for AI systems.

Content generation (generation of text, images, and music) is a growing field of deep learning applications coming up after the two machine learning conventional tasks; classification and prediction, such as translation and recognition of images and voices.

For automatically learning musical styles from arbitrary musical corporations without human user interaction and then producing samples from the predicted distribution, the skill of deep learning architectures and training techniques has been used. (Briot and Pachet, 2017)

2.4.1 Feedforward neural networks

Multilayer Perceptrons and Feedforward Neural Networks are the same and it is a perfect deep learning model example. They are pointed to as feedforward since data flows from input through the neurons, and finally to the estimated output. Figure 2.8 is an example of the MLP structure. Feedforward networks for machine learning experts are extremely important. They form the base of many significant commercial applications; object recognition from images with convolutions networks special types of feedforward neural networks. It is also the cornerstone of the recurrent neural network.

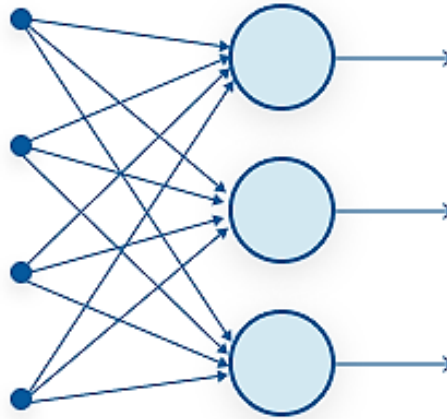


Figure 2.8: An example of MLP (AI Wiki, 2020).

2.4.2 Recurrent neural networks RNNs

It is conceptually easy to switch from a feedforward to a recurrent neural network. The first assumes data samples distributed identically and independently and typically maps from fixed-size inputs to fixed-size outputs, whereas RNN with its two special types (LSTM and GRUs) is a continuation of a conventional neural network that can be used by retaining state in the so-called recurring layers to model data with temporal dependencies (Jordan, 1997), it naturally works on input sequences of variable length and maps output sequences of variable length. It is apparent that data such as audio signals, text, and music also have temporal dependencies in the real world, so RNN has a strong capacity to model those data types, and it is suitable for dealing with data that have time-series and sequential properties (DiPietro and Hager, 2019; Chung et al., 2014).

Briot et al., (2017) was published a very comprehensive survey book about the deep learning techniques used for music generation; almost all RNN techniques were clarified. Figure 2.9 is an example of the RNN structure, the solid lines indicate feed-forward connections while loop lines indicate connections over time: from time phase (t) to ($t + 1$).

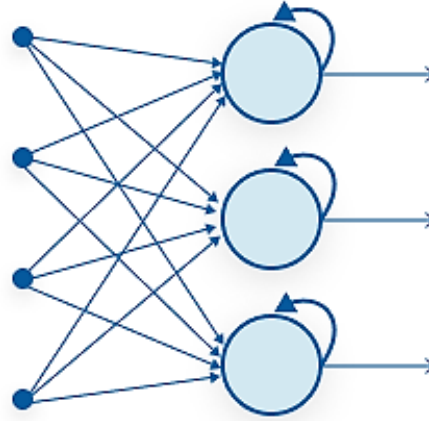


Figure 2.9: An example of RNN (AI Wiki, 2020).

The RNN input is a sequence of input vectors X (X_0, X_1, \dots, X_{t-1}). The arbitrary length of the sequence is (t) . For the input (X_i) of each time step, there is a corresponding network output (y_i) as shown in Figure 2.10. The state vector (h) of a layer is updated based on current inputs and the previous state as shown in Equation 2.1 (Goodfellow et al., 2016).

$$h_t = f(W_x X_t + W_h h_{t-1}) \quad (2.1)$$

Where W_x and W_h are weight matrices applied to the inputs and the state respectively and $f=\sigma(z)$ typically is a sigmoid function or a tanh function as shown in Equations 2.2 & 2.3 respectively (Chung et al., 2014).

$$\sigma(z) = \frac{1}{1 + \exp(-z)} \quad (2.2)$$

$$\sigma(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)} \quad (2.3)$$

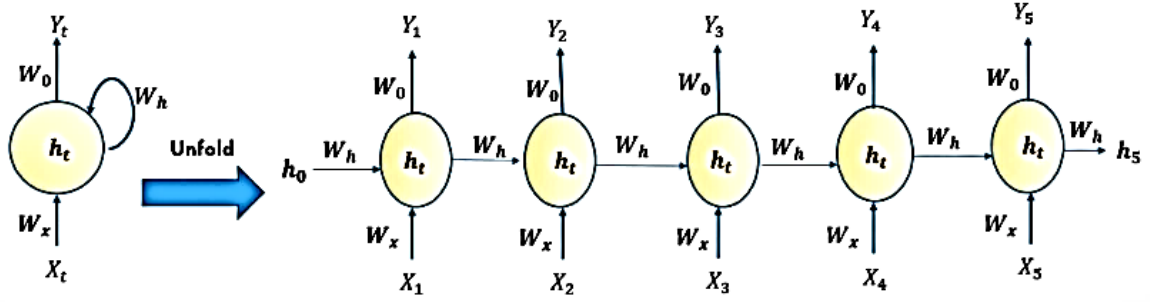


Figure 2.10: Unfolded RNN (Anwla, 2020).

In this work, we are interested in evaluating the performance of those recently proposed recurrent units (LSTM and GRU) on sequence modeling. Before the empirical evaluation, we first describe each of those recurrent units with some important mechanisms such as Bidirectionality and Attention in this section.

1. RNNs applications


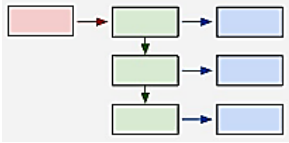
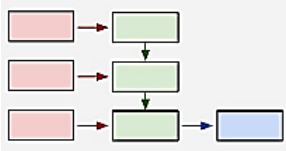
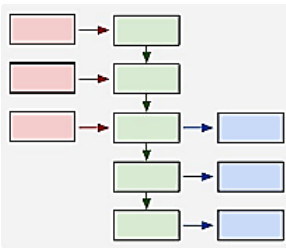
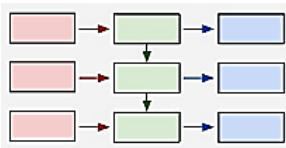
We may spot the current applications for RNN by reading and studying previous works of other researchers. (Cho et al., 2014; Sutskever et al., 2014) have a contribution in the field of machine translation, in text-generation, there are (Sutskever et al., 2011; Karpathy, 2015) papers, for video game generation application we can explore (Summerville and Mateas, 2016), and in music generation field which is associate to this project topic, we can reference (Mozer, 1994; Oord et al., 2016; Eck and Schmidhuber, 2002; Liu et al., 2014; Boulanger-Lewandowski et al., 2012; Choi et al., 2016; Walder, 2016).

2. RNNs architectural types

A major disadvantage of Pure Neural Networks (MLPs) is that they accept as input a fixed-sized vector and produce as output a fixed-sized vector for example input as an image and probabilities of different classes as the output. This mapping is carried out by these types of models using a defined number of model layers it means fixed computation steps

number. The recurrent networks are more excitement and this is because of their working style over vector sequences: input sequences, output sequences, or both, in the most general case. This property leads to a differentiation between many RNN types of architecture; many-to-one, one-to-many and many-to-many networks as Karpathy labels them and provides some examples of each type (Karpathy, 2015). Table 2.1 represents graphically these architectural types and also the MLPs one-to-one type.

Table 2.1: Architectural types of RNNs, derived from (Karpathy, 2015).

Type of RNN	Illustration	Example
One-to-one $T_x = T_y = 1$		Traditional neural network
One-to-many $T_x = 1, T_y > 1$		Music generation
Many-to-one $T_x > 1, T_y = 1$		Sentiment classification/ Music generation
Many-to-many $T_x = T_y$		Name entity recognition
Many-to-many $T_x \neq T_y$		Machine translation

3. Network optimization

To minimize losses, we use techniques (methods or algorithms) which are adjusting the weights and learning rate of neural networks, these techniques are called optimizers. There are many types of an optimizer; here we try to mention some of them used for training such as Gradient Descent, Stochastic Gradient Descent (SGD), Momentum, Nesterov Momentum, AdaGrad, RMSProp, and Adam which is used in our work. We explain it in some detail in the following section (Karpathy, et al., 2016; Kingma & Ba, 2015).

a. Gradient Descent

The gradient descent algorithm is the base of a vast majority of artificial neural networks (Brownlee, 2017). Before learning neural networks, it is important to comprehend the principles of this algorithm which is used for determining the lowest point by reaching zero derivatives of the function. Let's suggest minimizing the ($f(x)$) function that differentiates As shown in Figure 2.11, the gradient descent algorithm starts at an arbitrary location and recursively breaks down to the minimum point in some final value of (x).

Goodfellow et al., (2016) were explained thoroughly all issues about the gradient descent algorithm work, they denoted the challenges of determining the optimal global minima especially when the function has many local minimums. in gradient descent approaches we have to take into account the fact that the algorithm will lead to a local minimum instead of reaching global minimum which is the entire $f(x)$ minimum value, Figure 2.12 demonstrate this problematic issue. This predominant obstacle will occur for the reason that the gradient does not have any more route at these locations to move forward, in mathematical word at the points which are known as critical points When $f'(x) = 0$, it means that the slope has no direction information for moving.

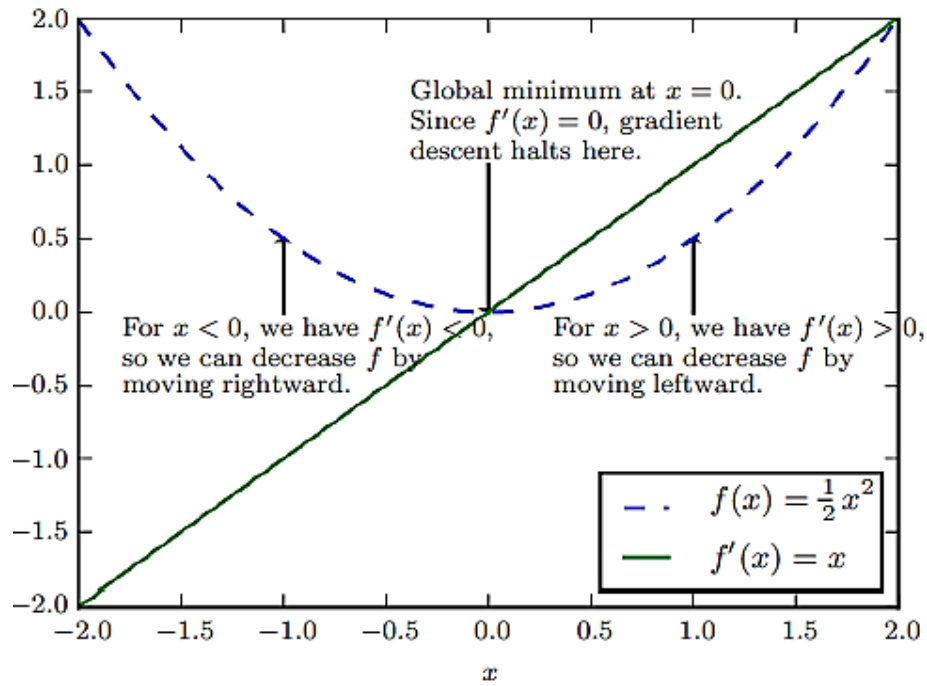


Figure 2.11: Gradient descent reaches the global minimum when $f'(x) = 0$

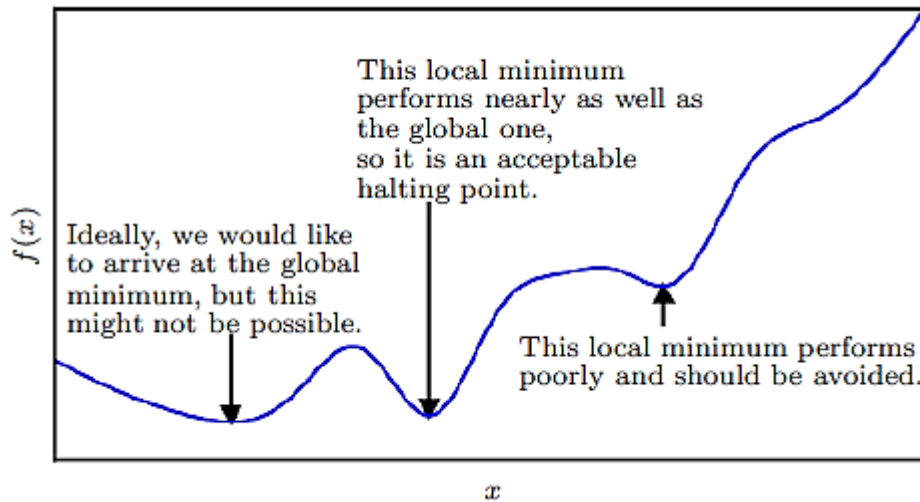


Figure 2.12: Local and global minimum reaching challenge of gradient descent

b. Adam optimizer

Adam is a continuation of stochastic gradient descent and can be used to update the weights of the network. Jimmy Ba and Diederik Kingma have presented Adam optimizer in 2015 in the "Adam: A Method for Stochastic Optimization" paper. Adam is defined as combining the benefits of the Adaptive Gradient Algorithm (AdaGrad) and the Root Mean Square Propagation (RMSProp) (Kingma & Ba, 2015). Concerning Adam Configuration Parameters, the learning rate ($\alpha = 0.001$), the first moment decay rate ($\beta_1 = 0.9$), the second moment decay rate ($\beta_2 = 0.999$) and a tiny number for avoiding any division by zero ($\epsilon = 10^{-8}$) are the default tested settings in machine learning implementations (Kingma & Ba, 2015). Adam adapted in papers that deal with DL for benchmarks. It has been used by (Gregor et al., 2015; Xu, 2015). Figure 2.13 ensures that Adam optimizer has the best result to reach minimum training cost.

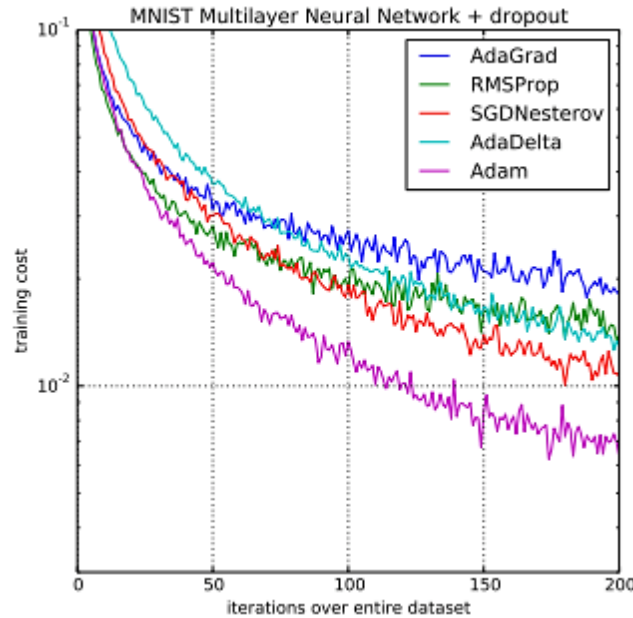


Figure 2.13: Adam optimizer has the best result to reach to minimum training cost (Kingma & Ba, 2015).

c. Dropout

Dropout is an important strategy and a very useful technique that has effects similar to regularization for avoiding overfitting of the neural network. Through adopting this powerful strategy, partially excludes neurons from the network. The key concept, as seen in Figure 2.14, is to eliminate the hidden neurons randomly selected together with their connections throughout the training process (Srivastava et al., 2014). By dropout rate as a probability which is between 0 and 1, any weight between units is set to zero in each epoch according to this probability. For example with the rate of 0.5 means eliminate half of the existing units over each epoch.

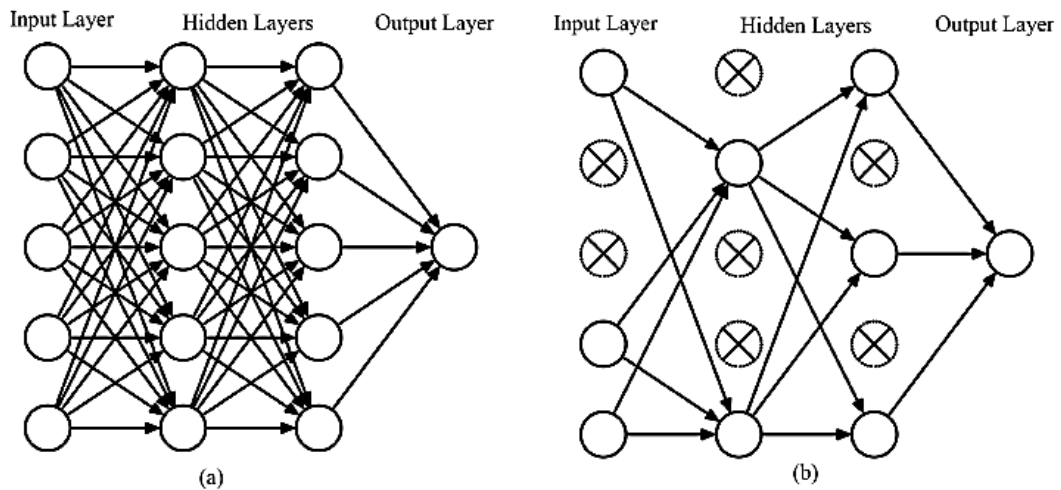


Figure 2.14: (a) A typical neural network. (b) A typical neural network after applying dropout technique (Srivastava et al., 2014).

4. Long short term memory LSTM

Originally (LSTM) proposed via (Hochreiter and Schmidhuber, 1997). Several slight changes have been made to the original LSTM unit since then. (Graves and Schmidhuber, 2005) was clarified that a series of memory blocks make up an LSTM layer that is connected recurrently. Somehow, they are similar to the memory chips used in digital computers, and each one comprises one or more repetitive memory cells as demonstrated via Equation 2.7 with the input, output, and forget gates, calculating their values by Equations 2.4, 2.5 & 2.6 respectively. In this project, we assume the LSTM implementation same as used in (Graves, 2013). Figure 2.15 demonstrates a particular memory cell of LSTM.

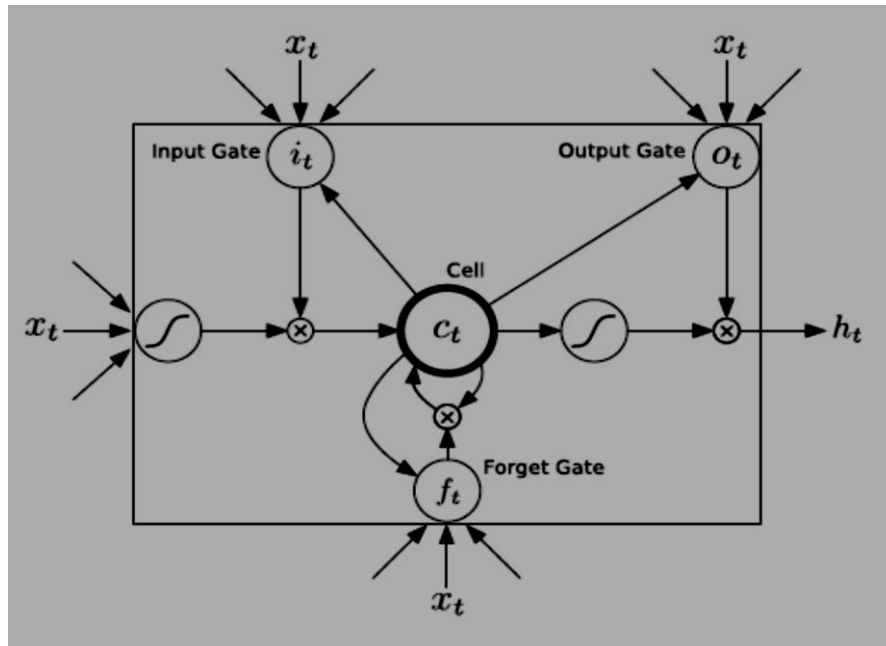


Figure 2.15: Memory cell of LSTM (Graves, 2013).

Via integrated gates LSTM unit may determine whether to retain the existing memory if a significant feature of an input sequence is identified by the LSTM device at an early time;

it easily carries this information over a long distance. This is unlike the conventional recurring unit that discards its content at each time step. In the version of LSTM used in this thesis, the following merged functions are executed for determining the function of hidden layer H , see Equation 2.8.

$$it = \sigma (W_{xi} x_t + W_{hi} h_{t-1} + W_{ci} c_{t-1} + b_i) \quad (2.4)$$

$$ot = \sigma (W_{xo} x_t + W_{ho} h_{t-1} + W_{co} c_t + b_o) \quad (2.5)$$

$$ft = \sigma (W_{xf} x_t + W_{hf} h_{t-1} + W_{cf} c_{t-1} + b_f) \quad (2.6)$$

$$ct = ft c_{t-1} + it \tanh (W_{xc} x_t + W_{hc} h_{t-1} + b_c) \quad (2.7)$$

$$ht = ot \tanh (ct) \quad (2.8)$$

- Sigmoid function denoted as σ .
- Input, output, forget gates, and the cell for storing information denoted respectively as i , o , f , and c .

The above equations are founded in the work of (Graves, 2013).

5. Gated recurrent units GRUs

Cho et al., (2014) was suggested Gated Recurrent Units GRUs allow each recurrent unit to grasp different time scale dependencies iteratively. It has gating units, similar to the LSTM unit, which modulates the information flow within the unit, but without having separate memory cells. Through the use of the reset and update gates, their vectors determine what data should be transferred to the output; GRUs aim to resolve the vanishing gradient issue of a conventional RNN. Figure 2.16 illustrates the workflow and design of GRUs (Kostadinov, 2017). For determining the function of the final hidden layer H , the following merged functions in Equations 2.9, 2.10, 2.11, and 2.12 are performed.

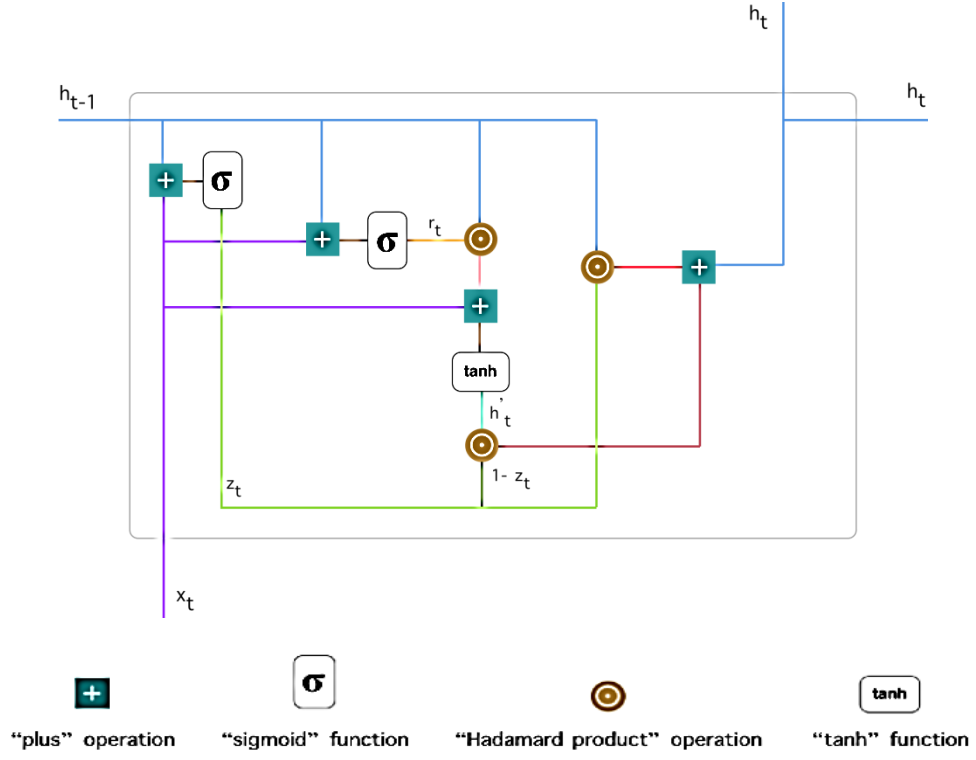


Figure 2.16: Gated Recurrent Unit

$$z_t = \sigma (W^{(z)} x_t + U^{(z)} h_{t-1}) \quad (2.9)$$

$$r_t = \sigma (W^{(r)} x_t + U^{(r)} h_{t-1}) \quad (2.10)$$

$$h'_t = \tanh (W x_t + r_t \odot U h_{t-1}) \quad (2.11)$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h'_t \quad (2.12)$$

- Sigmoid function denoted as σ .
- Update gate, Reset gate, the content of current memory and current time step final memory are denoted respectively as z_t , r_t , h'_t and h_t .

6. Bidirectionality

Schuster and Paliwal, (1997), pointed out some of the drawbacks of RNNs in their paper, such as RNNs cannot recognize the upcoming context during training and suggested an RNN changed version and they named it Bi-directional recurrent neural network (BI-RNN), which overcomes this limitation, for example. Their work can be recognized as a starting point for using and inventing this mechanism. Its name indicates that the fundamental principle of the Bi-directional network is the forward and backward recurrent execution associated with the identical output layer, in another word (BI-RNN) joins only one output to two hidden layers, first running from left to right and the other in opposite direction, that is for collecting past and future states information (Andrew, 2019). This power designates that the BI-RNN has absolute, temporal knowledge about all points before and after, for any point in a time series and it's a useful function especially for symbolic music generating; the context of the input is important (Graves and Schmidhuber, 2005). Figure 2.17 illustrates the general structure of BI-RNN.

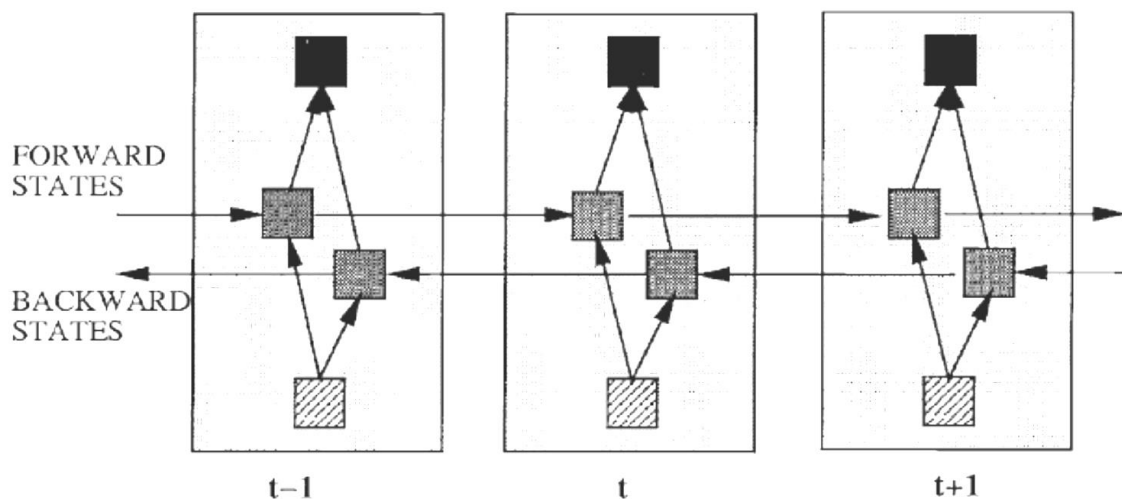


Figure 2.17: General structure of the unfolded BI-RNN (Schuster and Paliwal, 1997).

The formal definition of BI-RNN

Considering Forward RNN and Backward RNN manners data frontward from left to right, right to left respectively and Output y_t as shown in Equation 2.15 links and calculates the forward \vec{h}_t and backward \overleftarrow{h}_t outputs as shown in Equations 2.13 and 2.14 respectively. From Figure 2.18, and by merging equations, BI-RNN can be defined formally as follow. All equations below are taken from (Picheny et al., 2016).

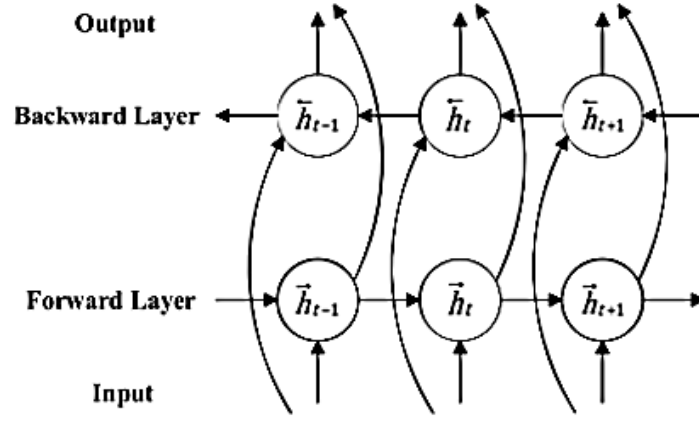


Figure 2.18: Bi-directional RNN (Picheny et al., 2016)

$$\vec{h}_t = \sigma (\vec{W} \cdot x_t + \vec{R} \cdot \overrightarrow{h_{t-1}} + \vec{b}) \quad (2.13)$$

$$\overleftarrow{h}_t = \sigma (\overleftarrow{W} \cdot x_t + \overleftarrow{R} \cdot \overleftarrow{h_{t+1}} + \overleftarrow{b}) \quad (2.14)$$

$$y_t = g (W_y [\vec{h}_t, \overleftarrow{h}_t] + b_y) \quad (2.15)$$

- σ : sigmoid function.
- $X_t \in \mathbb{R}^D$: Input vector sequence.
- $\vec{W}, \overleftarrow{W} \in \mathbb{R}^{n_{i-1} \times n_i}$: weight between Input and hidden.
- $\vec{R}, \overleftarrow{R} \in \mathbb{R}^{n_i \times n_i}$: weight between hidden layers t, and $\vec{b}, \overleftarrow{b} \in \mathbb{R}^{n_i}$: Bias.

7. Attention mechanism

With the evidence of (Bahdanau et al., 2015) paper, attention was born for translation and remembering lengthy sentences in translation with a neural machine. Unlike a conventional sequence-to-sequence model, Attention allows the input to be interpreted by the RNN to relay information for each word it detects, and then to concentrate on words as they become important for the RNN producing the output, whereas the former Sequence to Sequence S2S model has to refine the entire input down to a particular vector and then extend it back out. In a fact, deep learning attention can be generally taken as a vector of weights of relevance: to guess or deduce one element, like an image pixel or a sentence word, we measure how strongly it is associated with other elements using the attention context vector and taking the sum of its weighted values by the context vector as the target estimate (Weng, 2018). Principally as shown in Figure 2.19 the context vector contains information collected and arranged between input (source) and output (target).

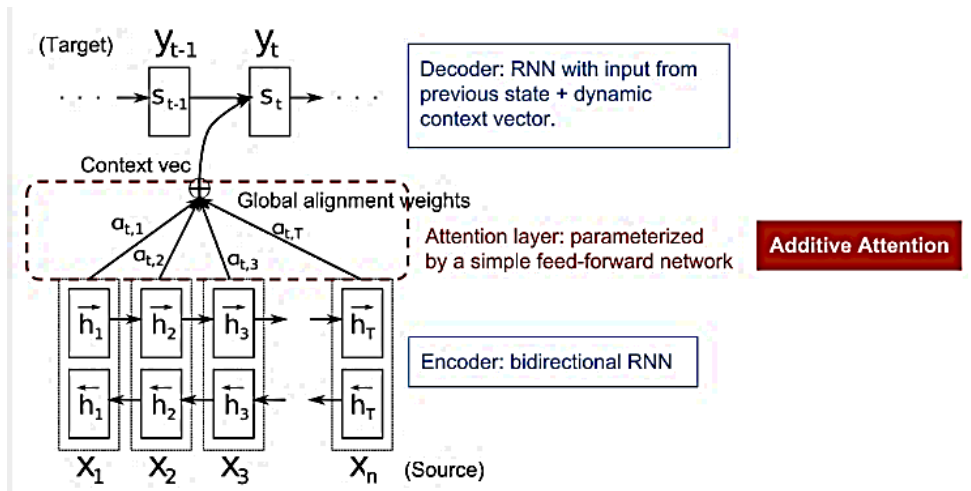


Figure 2.19: Attention mechanism between BI-RNN encoder and RNN decoder (Weng, 2018).

Figure 2.20 and its derivative one Figure 2.21 which are derived from (Bahdanau et al., 2015) paper, describe systematically the attention mechanism implemented in neural

machine translation. Having a sequence of source $x = [x_1, x_2, \dots, x_n]$ and try to produce a sequence of $y = [y_1, y_2, \dots, y_m]$ with attention to the relationship among words, for example, the word *Economic* has full attention with its French meaning which is *économique* as well as has relations with words such as, *européenne*, *zone*, and *la*.

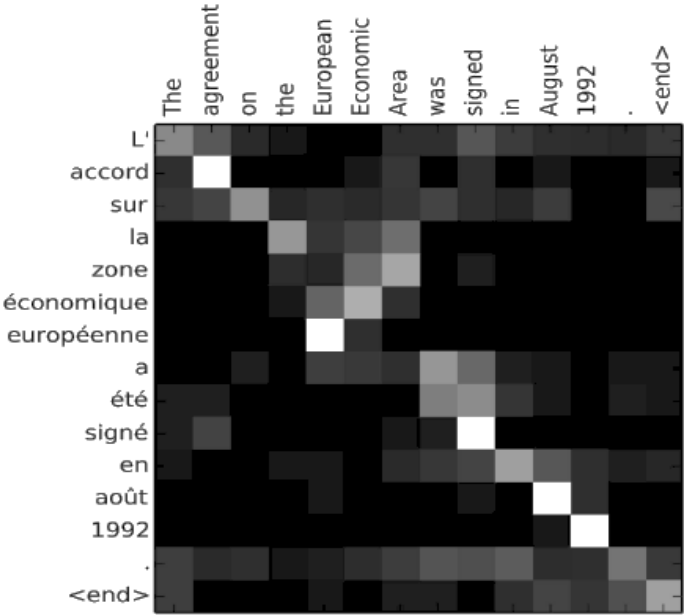


Figure 2.20: Alignment matrix of French and its English translation. (Image from Bahdanau et al., 2015).

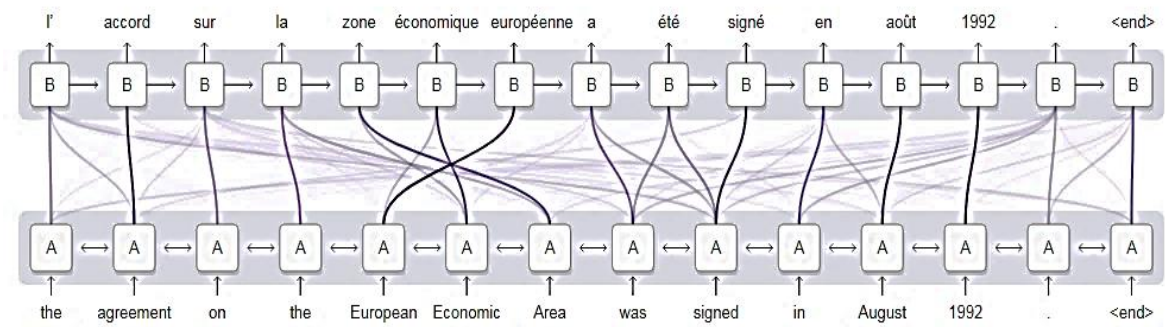


Figure 2.21: Matrix of alignment illustrates the association between the source and target words (Olah and Carter, 2016).

CHAPTER 3

METHODOLOGY & ARCHITECTURE

In this chapter, the logical compatibility of the various choices of algorithms, strategies, architectures, and learning models that have been used in this work, and their aspects to generate monophonic music was addressed, as well as the methodology of evaluating the generated samples was explained.

It is known that the use of deep learning techniques in any task in any of the different fields needs several procedures prerequisites and necessities. Here we present those much related to our work:

- Computational resources: Macintosh computer (Mac Pro Mid 2010) with these specifications has been used to accomplish this task in all its stages: Processor 2.8 GHz Quad-Core IntelXeon; memory 16 GB 1066 MHz DDR3; Startup Disk SSD; and Graphics ATI Radeon HD5770 1024 MB. Although the Mac was the best operating system that deals with music and sound engineering software but with respect to DL, there are many other OSs such as Linux and cloud platforms better for faster implementation such as AWS and Google Colab.
- Model Building: It was necessary to develop and train the machine learning model with the built data collection. As the techniques and algorithms the most popular RNNs such as LSTM, GRUs, have been used with the Bi-directional and Attention mechanism. Here in this chapter, we will explain the exact ways, techniques, and algorithms we were used and there are details about them in chapter 2.
- Programming languages and associated modules and libraries: Python 3.8, TensorFlow keras_self_attention, Keras, music21, sklearn, keras_tqdm, Pypianoroll, tqdm, librosa, pretty_midi, matplotlib, numpy, and magenta have been used. In the following sections, we will explain the coding language, libraries, and approaches that were used. All necessary screenshots of codes for implementation in this project have been addressed in Appendices.

- Datasets and preprocessing steps: multi-genre (pop, jazz, rock, etc.) and single-genre (folk) have been used with some useful preprocessing such as choosing only monophonic and 4/4 time signature music, and transposing them to C major and A minor Key signatures and converting all to 120 b/m tempo for simplicity and good analyzing understanding. Details have been explained in chapter 4.
- Evaluation: It was important to identify two mechanisms of evaluation for testing the model's performance. Firstly, objectively all results of implementations have been analyzed with the aid of experts in the music field, and secondly, subjective assessment has been done via analyzing a proposed survey to determine how generated samples from the models are rhythmically stable and melodically interested.

3.1 LSTM and GRUs

Recurrent Neural Networks (RNNs) allow long-term dependence to be integrated into the model. Theoretically, extremely long sequences can be recalled, but in reality, it is constrained by the vanishing gradient problem as well as the probability of an exploded gradient is also present, in which the gradient increases exponentially. As stated in (Hochreiter and Schmidhuber, 1997) Dependencies with large time steps can be managed by an LSTM or GRU network without being less successful in modeling short-term dependencies. So we decided to use the Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRUs) version of the RNN to mitigate these limitations. In the form of gates that regulate the flow of the learning, LSTMs and GRUs give a solution to the vanishing gradient problem and immune the error noise in the sequence data to learn the basic patterns of music structures by preserving an internal state guarded with the "forget" gate. More details about neural networks for generating music have been discussed in chapter 2.

3.2 LSTM vs GRUs

These two popular versions of RNNs compare to simple RNN, they are needed longer training times and more computing resources, as there are more training parameters for monitoring the respective gates (Sturm et al., 2016). Obviously, the GRUs has fewer parameters than LSTMs due to the cell number and their processes. So at some functions, it has been seen that they do as well or even better than LSTMs, the former needs shorter training times and fewer computation resources.

Chung et al, (2014), compared polyphonic music datasets with RNNs, LSTM, and GRU. They found that both LSTM and RNNs were outperformed by GRU. As well as GRU outperformed LSTM networks only marginally, so they concluded that the type of task and dataset are likely to depend on the best choice between the two. Associated with our work their effects on the results and the comparison between them have been discussed in chapter 5.

3.3 Gradient Descent Optimization Techniques

In chapter 2 there are sufficient details about general network optimization and its techniques, here we demonstrate those we used in models and designed architectures. For achieving neural network optimization and to obtain minimum training cost, two effective approaches or techniques have been used. The first one was for avoiding the overfitting via dropping out some of the hidden layers of the network randomly by a chosen ratio (Gal and Ghahramani, 2016), and as used by (Felbo et al., 2017; Johnson, 2017; and Dong, 2018), and then the second technique was to obtain weight optimization through the use of Adam optimizer which is used by (Kingma & Ba, 2015; Ruder, 2016).

In this work and for all experiments, each layer in the designed architecture either LSTM or GRU followed by a dropout layer with a 40% rate for reducing overfitting issue, the piece of code from Keras library which applied in this project for the purpose of controlling overfitting issue has been shown below.

```

This_layer = keras.layers.LSTM(
    cells,
    dropout=0.4,
    name='LSTM_%s' % I, return_sequence=ret_seq
)

```

Deadlock is another issue widely faced in this area when the machine prevents learning. This implies that the technique of gradient descent is actually running from one slope to another, not being able to come down. Through the use of Adam and Plateau codes from Keras library this issue can be solved by reducing the learning rate.

Concerning Adam parameter values, for almost all implemented experiments the same strategies have been proposed. The initial learning rate of 0.005 was used, and Validation loss plateaus values for *Factor*, *Patience*, and *Epsilon* were 0.5, 3, and 0.0005 respectively, the piece of code from Keras library which was implemented for this purpose has been shown below.

- *Factor*: states that by which amount the learning rate will change if validation loss does not optimized and not reduced.
- *Patience*: describes how many epochs the process should wait before changing the learning rate by the Factor value.
- *Epsilon*: it decides over each epoch, by which value of the difference between current and previous validation loss, the optimizer will start to change the learning rate value depending on the plateau rule.

```
optimizer = keras.optimizers.Adam(lr=0.005)
callbacks.append(ReduceLROnPlateau(monitor='val_loss',
                                   factor=0.5,
                                   patience=3,
                                   verbose=1
                                   mode='auto',
                                   epsilon=0.0005,
                                   cooldown=0,
                                   min_lr=0))
```

3.4 Bi-directional Mechanism

In the field of music generation, Bi-directional LSTMs were proposed successfully by many researchers, for example, (Dong, (2018); Mogren, (2016)) used Bidirectionality but with the GAN system. A thorough explanation of this mechanism was discussed in the previous chapter 2. Here in this project, a Bi-directional mechanism has been used with both LSTM and GRU networks with and without the use of attention mechanisms to identify its positive effect on the results which have been discussed in chapter 5.

3.5 Attention Mechanism

The attention mechanism was originally proposed for machine translation and has widespread use in this area (Bahdanau et al., 2015). Here we want to investigate that in the generation of symbolic music, does it have an impact on increasing learning, if it used with RRNs architecture? Subsequently, our emphasis is on the generation of music, We make the argument that whether attention enables the system to learn which parts of the musical piece are extremely vital to the next time sequence item prediction? So let's discuss some important principles of this mechanism to know the range of its effect on our samples made as described and done by (Felbo et al., 2017; Yang et al., 2016) in other research

fields. As humankind, if we are asked to describe the space in which we are seated, we pay close attention and we will look around at the things that we are describing. In real life the composers when attend to generate a simple melody they pay extensive attention to structuring motives, phrases, and music sentences by seeing what they repeat or what is the next note. There are many relations between current, next, and previous notes, bars, phrases, and many elements, that music structure consist of. Through using attention, neural networks may accomplish this same action; focusing on part of a subset of the data they are given (Olah and Carter, 2016).

Google Magenta team has hopeful results with using attention mechanisms in their open-source magenta project, especially in music generation (Waite, 2016). Inspired by their results we also attempted to use this mechanism in one of our experiments to investigate its effect on generated samples.

3.6 Architecture Type

The models' architecture type in both training and generating phases is supervised many-to-one architecture as shown in Figure 3.1. The input $Tx = 63$ time steps. In the training phase, we will feed the networks with them to predict the next step (y^{\wedge}). In fact, we have 4 bars of 4/4 time signature monophonic melody, each bar consists of (16) time steps (16 of sixteenth note), this means the overall is equal to (64) time steps, then by eliminating (one) time step we will obtain (63) time steps as described in the equation 3.1, and then we neglect the first time step to predict another one every time by moving the window one step each time forward. In the sample generating phase, we will feed the trained models with a sample of melody as a seed to obtain (64) new notes, the full (4) bars monophonic melodies such as the one that is shown in Figure 3.2.

$$Tx = 4[\text{bars}] \cdot 4 [\text{quarter notes}] \cdot 4[\text{sixteenth note}] - 1 = 63 \quad (3.1)$$

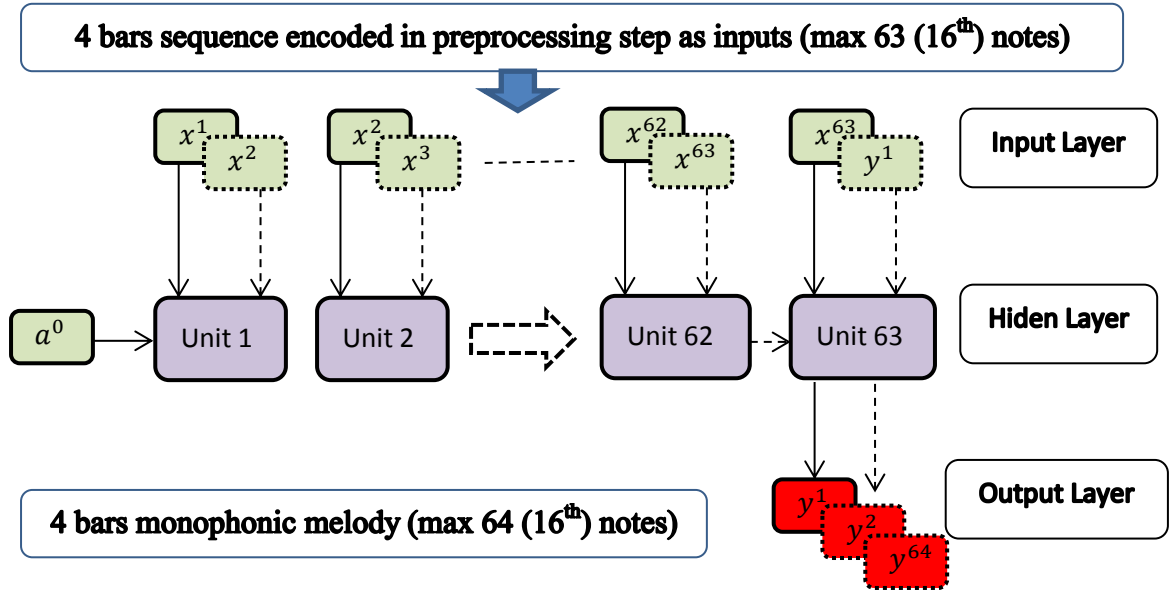


Figure 3.1: Music generation many-to-one RNN architecture type.



3.8 Human Evaluations: Professionally and Conventionally

Two different paths are proposed as a methodology of this thesis study part. The main process is to analyze two different perspectives; the first is from the point of view of masters and professionals in the music field, and the second, from the perspective of conventional participants or the audiences with multi-level musical talent as a survey analysis.

Concerning the first group, they were provided with several samples of generated music in sheet format to analyze mostly objectively through the basics and principles of music composition and theories of music. The samples obtained from each model have been implemented in the practical part of the work discussed in detail in its relative chapter.

Concerning the second group, ten randomly selected melody samples were given to participants to listen to them and rate each one rhythmically and for melodically pleasing using the Likert scale method from 1 to 5, as well as participants rate their experience level musically also from 1 to 5 (Tyler, 2014). This survey process is to analyze results and investigate model quality in generating monophonic melodies as well as to define the impact of any changes in all independent variables on the dependent one which is the generated sample from train models. This process can be classified as subjective decisions on the results. The statistical analysis has been explained in chapter 5 which is devoted to results and discussions.

3.9 Trained Models: Architecture Description

All Models trained and implemented in this project have independent variables, and can be classified into five categories:

- Dataset Types: Multi-genre and single-genre datasets have been used.
- Format encoding approaches: pianoroll format and magenta melody format have been proposed.

- Deep learning techniques: LSTM and GRUs have been tested.
- Learning mechanisms: Bi-directional and Attention mechanisms have been experienced.
- Hidden layers cell number: 64 cells have been used.

For training models in this work, the data set is randomly divided into a training set of 80% and a validation set of 20%. The training set contains data used to train the model, and the validation set consists of data to determine the model's generalization capability during the training phase. A batch size of 128 is used. All networks are trained for 50 epochs. In the Keras library, all the models are applied (Chollet et al., 2020). Below are the codes that are used for this purpose.

```
EPOCHS = 50
BATCH_SIZE = 128
def get_data(dataset) :
    dpath = dataset['path']
    dataset = np.memmap (dpath, mode="r",
                        dtype= "unit8", shape=dataset['shape'])
    x = dataset[:, :-1]
    y = dataset[:, -1]
    x_train, y_val, y_train, y_val = train_test_split(
        x, y, test_size = 0.2, random_state = 42, shuffle = True)
    print ('we have %s training files and %s validation files' %
          (len (y_train), len(y_val)))

    return x_train, y_val, y_train, y_val
```

Two types of datasets are used in this project; the first was a multi-genre dataset and consists of (19,877) songs from various styles. The second was a single-genre dataset contains (45,849) Irish folk songs, all necessary details about datasets and preprocessing are explained in chapter 4.

Here we try to focus on the Models that have been trained to make a comparison between them via evaluating their results (the melody samples are generated with them). So the overall trained model is consist of eight models fixed in Table 3.1, each has specific architecture depending on the changes in independent variables (encoding format, dataset nature, RNN types, Bi-directional mechanism, Attention mechanism, and hidden layers cell numbers) the particular architecture has given details in the following subsections.

Table 3.1: Models architecture per changes in the independent variables

Models No.	Models Name	Multi Genre dataset hook	Single genre dataset folk	Melody encoding	Pianoroll encoding	Bi-directional	2 LSTM layers	2 GRUs layers	64 cells	Attention
1	folk_melody_bi2lstm64_attention		*	*		*	*		*	*
2	folk_pianoroll_bi2lstm64_attention		*		*	*	*		*	*
3	hook_melody_bi2lstm64_attention	*		*		*	*		*	*
4	hook_pianoroll_bi2lstm64_attention	*			*	*	*		*	*
5	folk_melody_2lstm64_attention		*	*			*		*	*
6	folk_melody_2lstm64_noattention		*	*			*		*	
7	folk_melody_bi2GRU64_attention		*	*		*		*	*	*

3.9.1 Model No.1: model_folk_melody_bi2lstm64_attention

The architecture as shown in Figure 3.3 is as follows:

- *Input layer:* the network is fed from this layer depending on the number of vocabulary symbols and the input seed melody sample time steps and the architecture type of RNN which is many-to-one for generating purpose. Here the network has been fed for each iteration with 63 input time steps to predict the next note with the vocabulary symbols number of 58; the exact number of used vocabulary (note ranges) by the dataset songs (this model trained on Single-genre dataset).
- *Hidden layers:* consist of 2 layers of Bi-LSTM, each with 64 cells as described in the associated section of the mechanism of Bidirectionality in chapter 4; by adding a negative time path to LSTM, Bidirectionality doubles the number of cells. We, therefore, have 128 cells on each layer. A dropout layer, with a 40% rate, follows every layer.
- *Attention layer:* this model is designed with the Attention layer on top of the last Bi-LSTM hidden layer, in other words, it is applied after the last dropout layer.
- *Output layer:* this is for the prediction time step with the same vocabulary number 58 as the input layer, in this case, the aim was to produce only one next prediction note or time step, also depending on network architecture type many-to-one.

```
(magenta) Aras-Mac-Pro:gm arasharif$ python3 main.py --new --layers '2' --cells '64' --dataset
'folk_melody' --bi --att

Using TensorFlow backend.
training a new model...
we have 80000 training files and 20000 validation files
generated model id from args: model_folk_melody_bi2lstm64_attention
model id: model_folk_melody_bi2lstm64_attention
model dir: /Users/arasharif/Desktop/Aras_Master_Thesis/model_folk_melody_bi2lstm64_attention
```

Layer (type)	Output Shape	Param #
Input (InputLayer)	(None, 63, 58)	8
bi_0 (Bidirectional)	(None, 63, 128)	62976
bi_1 (Bidirectional)	(None, 63, 128)	98816
Attention (SeqWeightedAttent	(None, 128)	129
dense_outputs (Dense)	(None, 58)	7482

```

Total params: 169,403
Trainable params: 169,403
Non-trainable params: 0

```

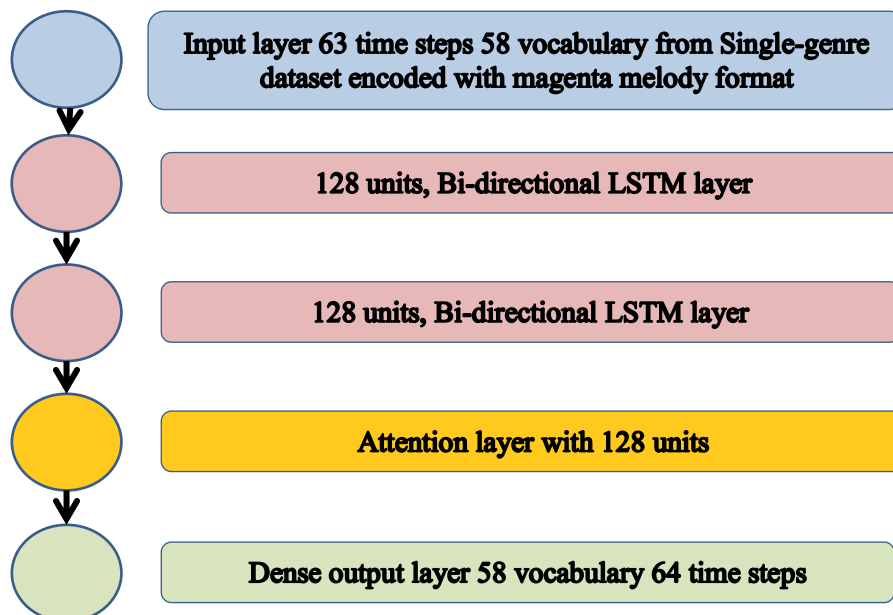


Figure 3.3: Architecture design of model No.1.

3.9.2 Model No.2: model_folk_pianoroll_bi2lstm64_attention

The architecture as shown in Figure 3.4 is as same as model No. 1 except for the input layer property. Depending on the encoding format used, the input layer varies according to the number of units. A vocabulary of 56 is produced by the pianoroll, while the melody has a vocabulary of 58. So the input layer nodes consist of 58 units.

```
Using TensorFlow backend.
training a new model...
we have 80000 training files and 20000 validation files
generated model id from args: model_folk_pianoroll_bi2lstm64_attention
model id: model_folk_pianoroll_bi2lstm64_attention
model dir: /Users/arasharif/Desktop/Ara_Master_Thesis/model_folk_pianoroll_bi2lstm64_attention
```

Layer (type)	Output Shape	Param #
Input (InputLayer)	(None, 63, 56)	0
bi_0 (Bidirectional)	(None, 63, 128)	61952
bi_1 (Bidirectional)	(None, 63, 128)	98816
Attention (SeqWeightedAttent	(None, 128)	129
dense_outputs (Dense)	(None, 56)	7224

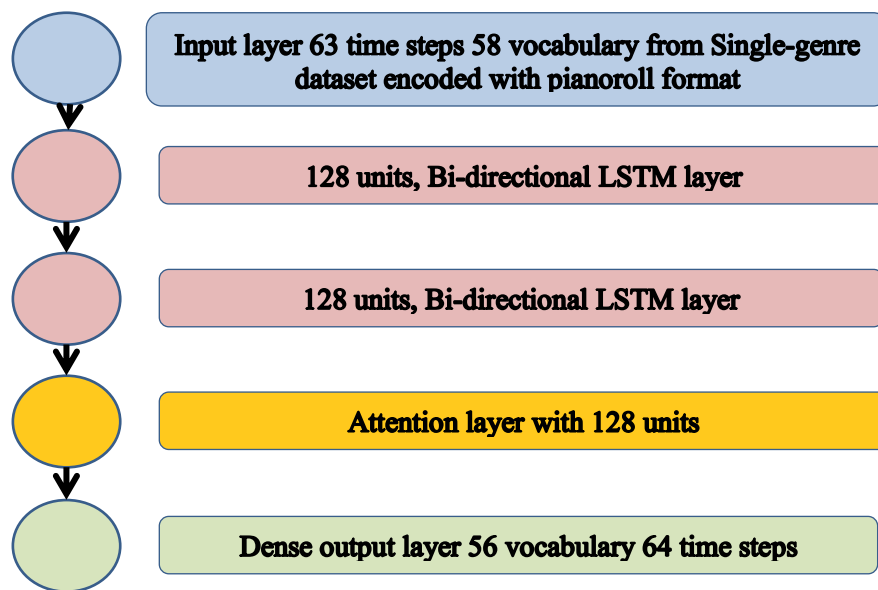


Figure 3.4: Architecture design of model No.2.

3.9.3 Model No.3: model_hook_melody_bi2lstm64_attention

The architecture as shown in Figure 3.5 is as same as model No. 1 except it is trained on the different dataset types; a multi-genre dataset that has a larger range vocabulary of 90 vocabularies. This model has been trained on 74933 files as a training set and 18734 files as a validation set.

'model_hook_melody_bi2lstm64_attention

Layer (type)	Output Shape	Param #
Input (InputLayer)	(None, 63, 90)	0
bi_0 (Bidirectional)	(None, 63, 128)	79360
bi_1 (Bidirectional)	(None, 63, 128)	98816
Attention (SeqWeightedAttent	(None, 128)	129
dense_outputs (Dense)	(None, 90)	11610
Total params: 189,915		
Trainable params: 189,915		
Non-trainable params: 0		

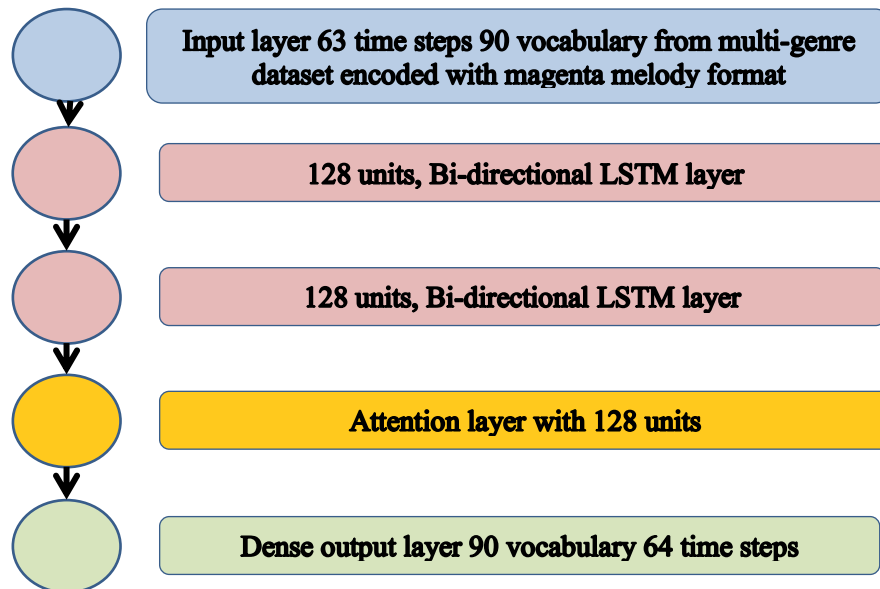


Figure 3.5: Architecture design of Model No.3.

3.9.4 Model No.4: model_hook_pianoroll_bi2lstm64_attention

The architecture as shown in Figure 3.6 is as same as model No. 1 except for the input layer property. Depending on the encoding format used, the input layer varies according to the number of units. A vocabulary of 88 is produced by the pianoroll, while the melody has a vocabulary of 90. So the input layer nodes consist of 88 units as well as the output nodes.

Layer (type)	Output Shape	Param #
Input (InputLayer)	(None, 63, 88)	0
bi_0 (Bidirectional)	(None, 63, 128)	78336
bi_1 (Bidirectional)	(None, 63, 128)	98816
Attention (SeqWeightedAttent	(None, 128)	129
dense_outputs (Dense)	(None, 88)	11352
Total params: 188,633		
Trainable params: 188,633		

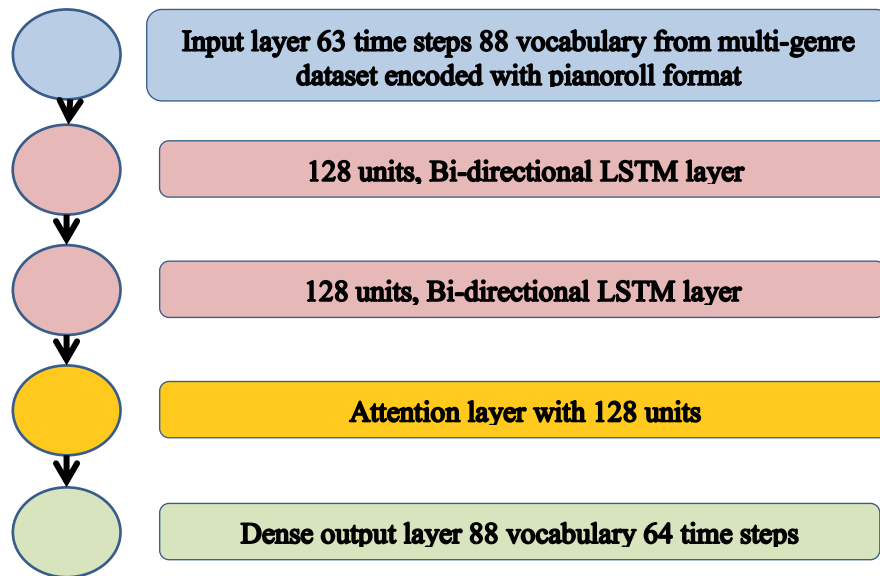


Figure 3.6: Architecture design of model No.4.

3.9.5 Model No.5: model_folk_melody_2lstm64_attention

The architecture as shown in Figure 3.7 is as follows:

- *Input layer:* the network is fed from this layer depending on the number of vocabulary symbols and the input seed melody sample time steps and the architecture type of RNN which is many-to-one for generating purpose. Here the network has been fed for each iteration with 63 input time steps to predict the next note with the vocabulary symbols number of 58; the exact number of used vocabulary (note ranges) by the dataset songs (this model trained on Single-genre dataset).
- *Hidden layers:* consist of 2 layers of LSTM, each with 64 cells. A dropout layer, with a 40% rate, follows every layer.
- *Attention layer:* this model is designed with the Attention layer on top of the last LSTM hidden layer, in other words, it is applied after the last dropout layer.
- *Output layer:* this is for the prediction time step with the same vocabulary number 58 as the input layer, in this case, the aim was to produce only one next prediction note or time step, also depending on network architecture type many-to-one.

```
(magenta) Aras-Mac-Pro:gm arasharif$ python3 main.py --new --layers '2'
--cells '64' --dataset 'folk_melody' --att

Using TensorFlow backend.
training a new model...
we have 88000 training files and 20000 validation files
generated model id from args: model_folk_melody_2lstm64_attention
model id: model_folk_melody_2lstm64_attention
model dir: /Users/arasharif/Desktop/Ara_Master_Thesis/model_folk_melody_2lstm64_attention
```

Layer (type)	Output Shape	Param #
Input (InputLayer)	(None, 63, 58)	0
LSTM_0 (LSTM)	(None, 63, 64)	31488
LSTM_1 (LSTM)	(None, 63, 64)	33024
Attention (SeqWeightedAttent	(None, 64)	65
dense_outputs (Dense)	(None, 58)	3770

```

=====
Total params: 68,347
Trainable params: 68,347
Non-trainable params: 0

```

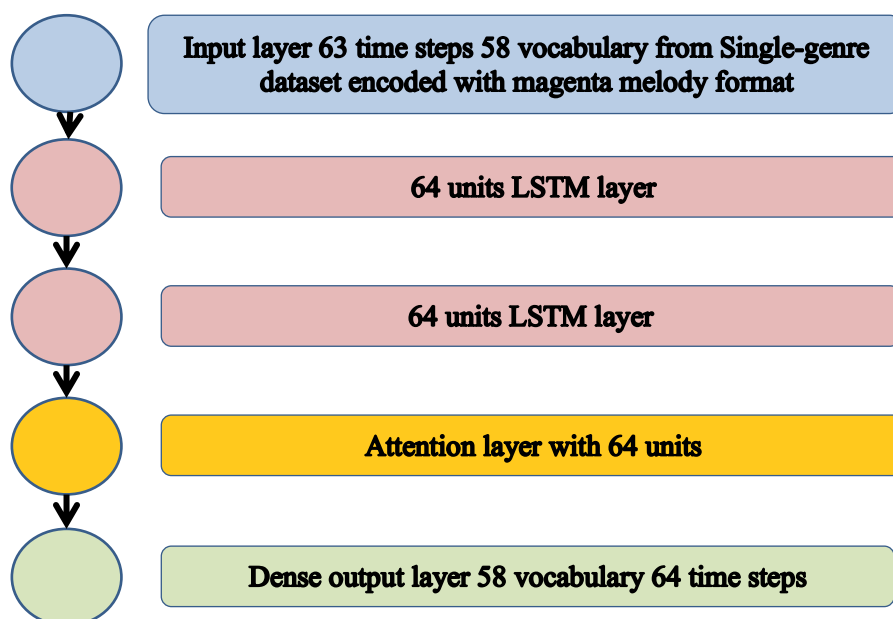


Figure 3.7: Neural Network architecture design of model No.5.

3.9.6 Model No.6: model_folk_melody_2lstm64_noattention

The architecture as shown in Figure 3.8 is as follows:

- *Input layer:* the network is fed from this layer depending on the number of vocabulary symbols and the input seed melody sample time steps and the architecture type of RNN which is many-to-one for generating purpose. Here the network has been fed with 63 input time steps in each iteration to predict the next note within the vocabulary range of 58 notes, model is trained on Single-genre.
- *Hidden layers:* consist of 2 layers of LSTM, each with 64 cells. A dropout layer, with a 40% rate, follows every layer.
- *Output layer:* this is for the prediction one time step depending on network architecture type many-to-one with the vocabulary number 58 as the input layer.

Layer (type)	Output Shape	Param #
Input (InputLayer)	(None, 63, 58)	0
LSTM_0 (LSTM)	(None, 63, 64)	31488
LSTM_1 (LSTM)	(None, 64)	33024
dense_outputs (Dense)	(None, 58)	3770

=====
Total params: 68,292
Trainable params: 68,282
Non-trainable params: 0

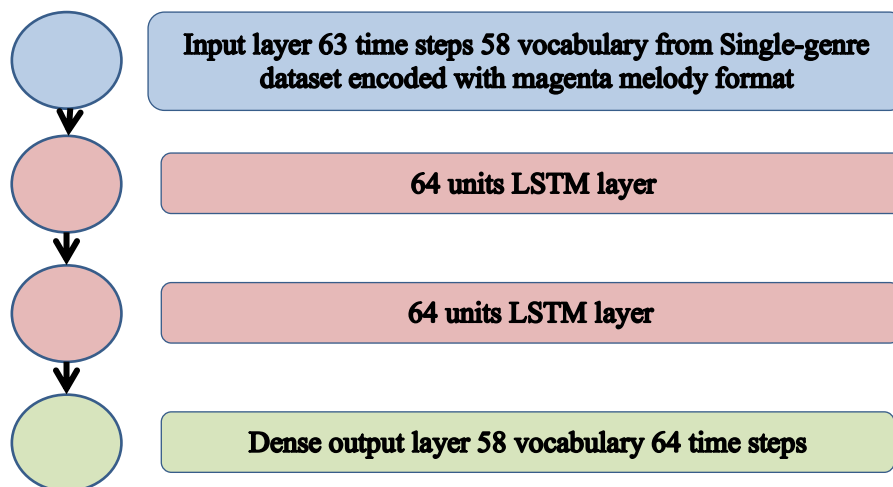


Figure 3.8: Neural Network architecture design of Model No.6.

3.9.7 Model No.7: model_folk_melody_bi2GRU64_attention

The architecture as shown in Figure 3.9 is as follows:

- *Input layer:* the network is fed from this layer depending on the number of vocabulary symbols and the input seed melody sample time steps and the architecture type of RNN which is many-to-one for generating purpose. Here the network has been fed for each iteration with 63 input time steps to predict the next note with the vocabulary symbols number of 58; the exact number of used vocabulary (note ranges) by the dataset songs (this model trained on Single-genre dataset).
- *Hidden layers:* consist of 2 layers of Bi-GRUs, each with 64 cells as described in the associated section of the mechanism of Bidirectionality in chapter 4; by adding a negative time path to Bi-GRUs, Bidirectionality doubles the number of cells. We, therefore, have 128 cells on each layer. A dropout layer, with a 40% rate, follows every layer.
- *Attention layer:* this model is designed with the Attention layer on top of the last Bi-GRUs hidden layer, in other words, it is applied after the last dropout layer.
- *Output layer:* this is for the prediction time step with the same vocabulary number 58 as the input layer, in this case, the aim was to produce only one next prediction note or time step, also depending on network architecture type many-to-one.

```
(magenta) Aras-Mac-Pro:gm arasharif$ python3 main_gru.py --new --layers '2'
```

```
--cells '64' --dataset 'folk_melody' --bi --att
```

Using TensorFlow backend.

training a new model...

we have 80000 training files and 20000 validation files

generated model id from args: model_folk_melody_bi2GRU64_attention

model id: model_folk_melody_bi2GRU64_attention

model dir: /Users/arasharif/Desktop/Aras_Master_Thesis/GRU/model_folk_melody_bi2GRU64_attention

Layer (type)	Output Shape	Param #
=====		
Input (InputLayer)	(None, 63, 58)	0

bi_GRU_0 (Bidirectional)	(None, 63, 128)	47232

bi_GRU_1 (Bidirectional)	(None, 63, 128)	74112

Attention (SeqWeightedAttent	(None, 128)	129

dense_outputs (Dense)	(None, 58)	7482
=====		
Total params: 128,955		
Trainable params: 128,955		
Non-trainable params: 0		

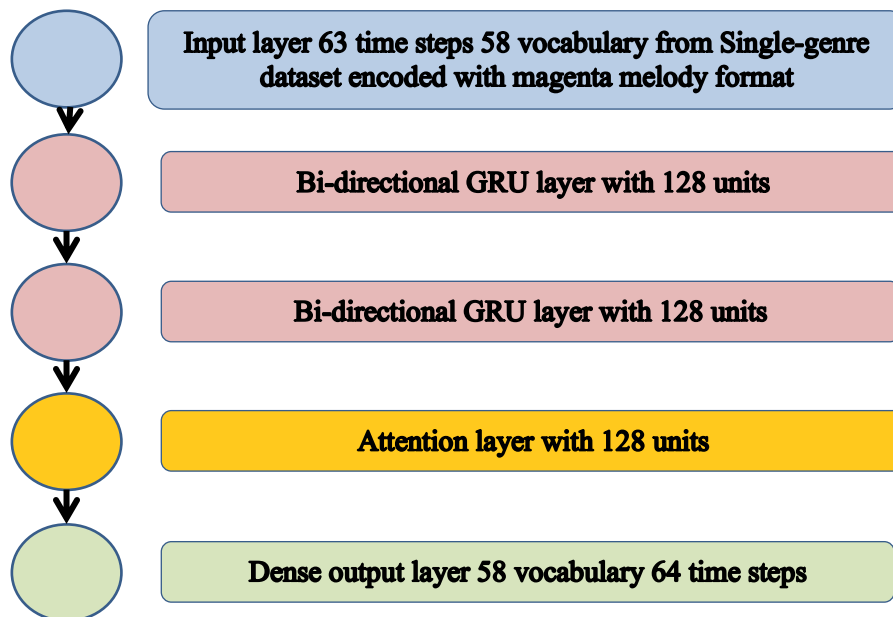


Figure 3.9: Neural Network architecture design of Model No.7.

CHAPTER 4

COLLECTING AND PROCESSING DATA

4.1 Dataset

For all machine learning problems, especially deep learning problems, it is clear that the quality of the dataset and its specifications are necessary to get a well-trained model. Gudivada et al. (2017) concluded in their paper that the central point for developing machine learning models is the Excellency of dataset quality.

We will address the datasets we have used to train the models in this chapter. We're going to include a description of their style and roots. In this thesis the data which have been used consists of two sets of data: the first contains Irish folklore songs assuming that it is a single-genre dataset and the second consists of a wide range of different styles as a multi-genre dataset, all of which contain MIDI file types. Obviously, for many causes, MIDI files are not ideal for being used strictly as an input to a machine learning algorithm. There is a lot of data in the files that are meaningless for producing music (De Coster, 2017). Besides, a MIDI file is a binary file that requires processes of transformation, there are many approaches for encoding MIID files and transform them to such a suitable file can be used by the RNNs; the pianoroll representation which is used by (Ycart et al., 2017; Mauthes, 2018; Dong et al., 2018). The Melody encoding format used by the Magenta google brain team (Waite, 2016) and Transformer encoding also used by (Huang et al., 2018) are some of these common approaches. As defined in Chapter 5 for each representation with varying results, a generative model is applied.

4.2 Multi-Genre Dataset

This set of data contains (19,877) divided songs into their parts (intro, verse, chorus, etc.), which is made up of songs of different styles such as Pop, Rock, jazz, metal, etc., was taken and downloaded from this address as a zip file with all meta-data information (https://drive.google.com/file/d/13iB5Brk1hypKsw9TSf8_d4Ka3xU0XmFZ/view Retrieved 03 March 2020), and it was mainly presented and collected from (<https://www.hooktheory.com/> Retrieved 01 March 2020) that contains 13,222 songs in the form of tabs lead sheets files at a time when we entered to it, the processing and conversion to MIDI files have been done through GitHub work by (Hsiao, 2019). It is worth noting that we consider this set of data as a multi-genre dataset. This type has been chosen to know the extent of its impact on learning our models and to compare it with the models learned from data of one specific genre like folk. This is explained in detail in Chapter 5.

4.3 Single-Genre Dataset

A single-style dataset consisting of (45,849) Irish folk songs in the form of a MIDI file was taken from GitHub prepared by (IraKurshunovava, 2019). CSV and JSON data files were originally taken from (Keith, 2015), by Kurshunova and were eventually cleaned and transformed from the ABC representation format to the MIDI file format. This website is the primary source of these folk tunes (<https://thesessiond.org/> Retrieved 02 March 2020). The aim of working on this type of dataset is to investigate the impact of RNNs architecture changes on the accuracy and the learning quality of generated models and melodies as result. This is also explained in detail in chapter 5.

4.4 Music Representation Formats


Traditional notation is too time-consuming to write down, too difficult to reproduce, and not suitable for electronic data processing. The need for international adoption of a basic code system for the notation of musical works is one of (Brook, 1965). Now a day there are different digital music formats such as ABC notation (Walshaw, 2020), Simplified

Plaine & Easie Code (Brook, 1965), Kern (Sapp, 2005), MusicXML (Good, 2001), and MIDI (Oliveira & Oliveira, 2017).

4.4.1 ABC format representation

Hundreds of thousands of tunes are now available on ABC, Figure 4.1, is an example of this representation. It was developed primarily for Western European folk and traditional tunes. It has become more popular, though, and has been used for many other kinds of music. One of the most important goals of ABC notation is that it is very clear, and this property distinguishes it from the most computer-based musical (Walshaw, 2020).

Recently this format was Used by many researchers work such as (Agarwala, 2017; Sturm et al., 2016).



X: 99
T: Short tune
C: K. Rettinghaus
M: 4/4
L: 1/4
K: G
D|G>ABG|A>BAD|G>ABA|G3|]

Figure 4.1: ABC format representation.

4.4.2 Simplified Plaine & Easie format representation

The original version of this format representation “Plaine and Easie Code System for Musike” was proposed in Brook and Murray in 1964. It was soon modified to make the

code more usable on an international basis for ordinary purposes, for detail (Brook, 1965). Figure 4.2, shows a simple example of this format.



4.4.4 MusicXML format representation

MusicXML (Markup languages) is an interchange and distribution format of digital sheet music. The purpose is to establish a standardized format for common Western music notation. It is designed to facilitate the exchange between applications for executing many processes such as musical notation, music information retrieval, instrument performing, and musical data analysis. It is designed for these applications to be adequate, not optimal. (Good, 2001). Figure 4.4, exhibits this Markup language format.



Figure 4.4: MusicXML format representation.

4.4.5 MIDI format representation

MIDI (Musical Instrument Digital Interface) is a standard music technology protocol that connects digital musical electronic instruments, and digital devices such as PC, tablets, and smartphones of many different organizations. The first version of MIDI was produced in 1983 and is used worldwide every day to create, perform, learn, and share music and creative works by musicians, DJs, developers, educators, and artists (Huber, 2007).

Music representation in this format includes any music data structure that focuses on music playback and editing as defined by one or more sequences of notes, with using many

operating systems MIDI programming interface, we can record a musician’s performance on a MIDI keyboard directly in MIDI and store it in data structures. MIDI describes musical note codes, as well as button key, dial, and pedal variations of digital instruments, and a sequence of synthesizers can be orchestrated by MIDI control messages (MMA, 2020). MIDI can also create a graphical interface such as the default pianoroll editor that represents the notes as horizontal bars as shown in Figure 4.5, that can be added, resized, transferred, removed, copied/pasted, transposed by the user. MIDI does not record sound waves that are analog or digital. It encodes keyboard features, including the beginning of a note, its pitch, duration, volume, and musical characteristics, such as vibrato (pcmag, 2020). All of these characteristics are in the 0-127 range (128 possible values), Figure 4.6, explains the relation between note names and their range number for example C4 has 21 of the range in a standard 88 keys MIDI keyboard.

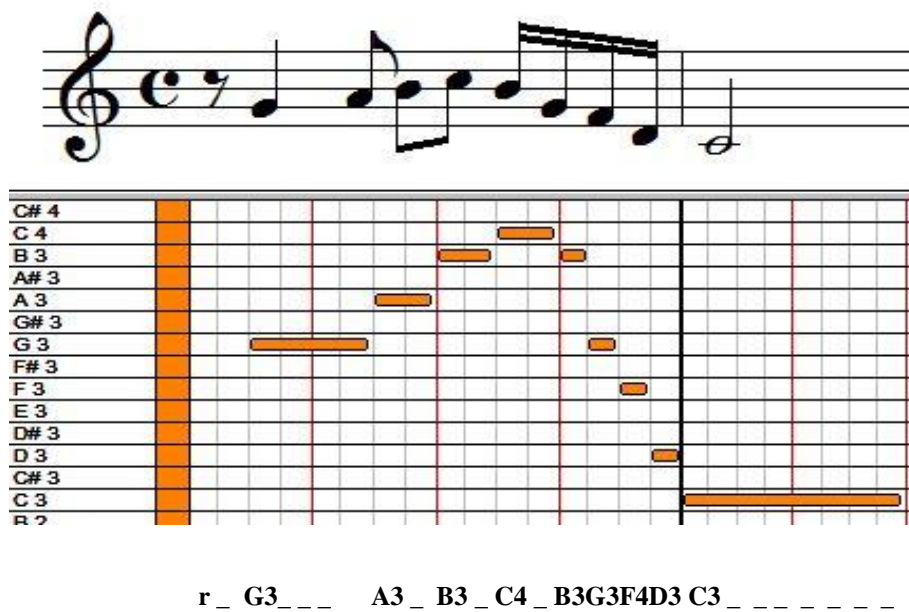


Figure 4.5: A combination of notation and its pianoroll representation (Vandenneucker, 2020).

Datasets with this format are used by so many researchers’ works related to music generation such as (Hilscher and Shahroudi, 2018).

	Frequency	Keyboard	Note name	MIDI number
	4186.0		C9	108
	3951.1		B8	107
	3729.3		A8	106
	3322.4		G8	104
	2960.0		F8	102
	2637.0		E8	100
	2489.0		D8	99
	2217.5		C8	97
	1975.5		B7	95
	1864.7		A7	94
	1661.2		G7	92
	1480.0		F7	90
	1318.5		E7	88
	1244.5		D7	87
	1108.7		C7	85
	987.77		B6	83
	932.33		A6	82
	830.61		G6	80
	739.99		F6	78
	659.26		E6	76
	622.25		D6	75
	554.37		C6	73
	493.88		B5	72
	466.16		A5	70
	415.30		G5	68
	369.99		F5	66
	349.23		E5	64
	311.13		D5	63
	277.18		C5	61
	246.94		B4	59
	233.08		A4	58
	207.65		G4	56
	185.00		F4	54
	164.81		E4	52
	155.56		D4	51
	138.59		C4	49
	123.47		B3	47
	116.54		A3	46
	103.83		G3	44
	92.499		F3	42
	87.307		E3	41
	82.407		D3	39
	77.782		C3	37
	69.296		B2	35
	61.735		A2	34
	58.270		G2	32
	51.913		F2	30
	46.249		E2	29
	41.203		D2	27
	38.891		C2	25
	34.648		B1	23
	32.703		A1	22
	30.868			
	29.135			
	27.500			

Figure 4.6: MIDI Note Numbers for Different Octaves.

4.5 Format Encoding

The method of the format encoding of musical inputs into the one that comprehensible by neural networks is one of the most important issues affecting the efficiency of the models that are produced and, accordingly, melody samples that generate from them (Laden and Keefe, 1989). Music that we listen to it directly from the sound sources in life or players when recorded is always in the form of sound waves. But in the case of digital electronic devices, the music in its waveform is inappropriate and undefined and must be converted into a suitable digital form. MIDI format representation which has been chosen in this work is a common digital format used by so many digital audio workstations; electronic devices or application software. Our dataset MIDI files need to be transformed and encoded using two format encoding; pianoroll and melody for our neural network inputs for comparing the effect of these encoding on the overall processes and the quality of generated models, detail is in chapter 5

4.5.1 Pianoroll format encoding

Pianoroll is a format for music representation that describes a piece of music via a score-like matrix. The vertically and horizontally axis, respectively, reflect note pitch and time. In this thesis, we set symbolic timing and the temporal resolution to 16 per beat to cover common temporal patterns till 16th notes such as setting do not include triplets and 32nd notes and over. The note pitch has 128 possibilities, covering from C-0 to G10 see Figure 4.7. A 4/4 time signature bar monophonic melody with one instrument the same as used in our experiments has a matrix with $(64 * 128)$ dimensions. For dealing with pianoroll encoding, we used Pypianoroll which is an open-source Python library. It offers an important multitrack pianoroll handling tool, including powerful I / O as well as a tool for simulation, analysis, and assessment (Dong et al., 2018).

Octave		Notes											
Number	C	C#	D	D#	E	F	F#	G	G#	A	A#	B	
0	0	1	2	3	4	5	6	7	8	9	10	11	
1	12	13	14	15	16	17	18	19	20	21	22	23	
2	24	25	26	27	28	29	30	31	32	33	34	35	
3	36	37	38	39	40	41	42	43	44	45	46	47	
4	48	49	50	51	52	53	54	55	56	57	58	59	
5	60	61	62	63	64	65	66	67	68	69	70	71	
6	72	73	74	75	76	77	78	79	80	81	82	83	
7	84	85	86	87	88	89	90	91	92	93	94	95	
8	96	97	98	99	100	101	102	103	104	105	106	107	
9	108	109	110	111	112	113	114	115	116	117	118	119	
10	120	121	122	123	124	125	126	127	-	-	-	-	

Figure 4.7: Each MIDI number is equivalent to an octave in the left-hand column and a note in the top row (Nicholsonr & Kim, 2016).

4.5.2 Magenta melody format encoding

This format is suggested in MelodyRNN models by Google Magenta which was started by some researchers and engineers from the Google Brain team with their motto “Make machines intelligent. Improve people’s lives”. Magenta is a research study that investigates the function of machine learning as a system in innovative processes. Driven by TensorFlow, Magenta as an open-source is distributed Python libraries to manipulate and train machine learning models with music pieces or image data, for using them to create new content (Waite, 2016). The python library that is used by Magenta is (melodies_lib.py), and it converts the file into melody format, considering all 128 equivalent numbers in pianoroll representation as shown in Figure 4.6, 0 to 127 = note-on even, -2 = no event, and -1 = note-off event. The feature that distinguishes Melody encoding from its pianoroll format counterpart is holding the (note-on) for the entire duration of the note; melody does not hold while pianoroll does (Roberts, 2019).

4.6 Preprocessing

All processes and techniques that are applied to the dataset and precede the training and generating processes can be measured as a vital part of any ML problem, especially its DL main branch techniques problems. It is clear that preparing the data and identifying its diagnostics will have an effective effect on the models that we gain from the training process and in turn on generated melodies. So it is important to have sufficient and abundant information about the characteristics of the data we have used. Any changes in these specifications, even if they are minor, will greatly affect the overall results (De Coster, 2017).

For preprocessing the datasets we have preceded the following operation steps:

1. Files with (.mid) extensions have been extracted from the dataset. The single genre (Folk) dataset contains (45,849) midi file songs and the multi-genre (Hook) dataset has (19,877) multi-structure midi song tracks
2. MIDI files with three and more staves (instruments) have been eliminated, only those with two instruments remained and we have assumed that the first staff contains a melody part and we deleted the second staff which usually contains chords, that's for selecting only the monophonic melodies (music that is written for only one voice or part) to train and obtain our models, this represented as an essential part of our project problem statement. Choosing only the melody part is to ensure to train our systems only on melodies to generate later monophonic music samples in the sampling processing stage. Concerning the hook multi-genre dataset, we obtained (17,954) songs after applying these criteria, but the Folk single dataset had (45,849) songs.
3. The remained dataset has been filtered and only those melodies have been selected which have a four-by-four (4/4) key signature, to simplify the learning. In this case, our systems will focus on learning only one type of rhythm, which is the most common time signature in popular and traditional music genres, especially in western music. In this stage, the remaining 4/4 key signature songs from the Hook dataset were (16,386) songs, and (24,234) four by four songs remained from the Folk dataset.

4. With the quantization process, we have converted all notes with triple duration to the nearest eighth or sixteenth note length.
5. We have transposed the keys of all chosen monophonic melodies to only “C” major and its related minor, “A” minor Key (Do Major and La Minor) as done in, this step is also proposed to let our systems focus on only one scale major type and its relative minor key instead of learning twelve existent keys in western music (Ycar et al., 20017; Hadjeres and Nielsen, 2017; Simon et al., 2018). Transposed dataset melodies to all keys and the results were lower learning with the same amount of time as described by (De Coster, 2017). It was because the network now wants to learn musicality patterns at the same time across multiple keys.
6. By detecting the number of used notes in songs and determining the vocabulary ranges (minimum and maximum notes) we have fined that the Folk dataset has a vocabulary of (58) used notes while the Hook dataset has (90) used pitches. We know that the standard MIDI pitches number is (127) pitches. Through this preprocessing step we let the systems focus on the only used notes which are less than the standard midi notes in both datasets. This means maximizing the total use of data memory, speeding up the learning process, as the model does not need to learn about the pitches that are not used.
7. Musical works with 4 bars are generally felt by listeners as either an ending or a turning point in the music (Kitagawa, 1999). Then by moving a 4-bar window at a time across each file, with a 1-bar step length, we have created multiple 4-bar sequences. This implies, for example, that a 5 to 8 bar MIDI file will have two 4-bar sequences, and a 9 to 12 bar file will have 3 bars, and so on. In this stage with the use of the two common encoding formats; pianoroll and Melody which have been explained earlier in this chapter, (93,667) 4-bar sequences from the multi-genre Hook dataset and (100,000) sequences from single genre Folk dataset have been obtained.
8. Five songs randomly as original songs from the single-genre dataset have been selected for comparison with five generated samples in the Human Evaluation section in chapter 5.

All codes related to the dataset preprocessing are settled in Appendix 1, and the workflow of the preprocessing can be summarized in Figure 4.8.

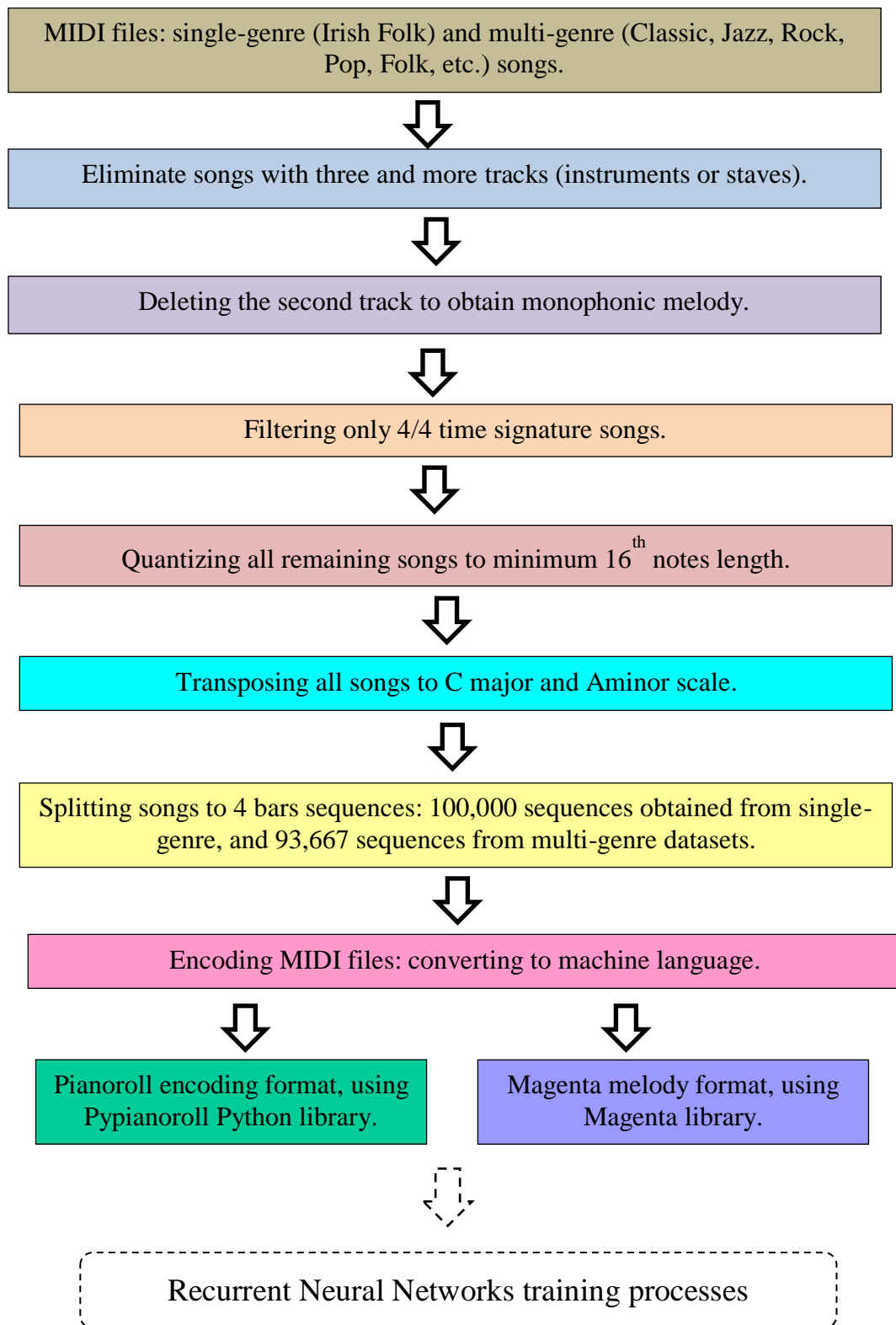


Figure 4.8: Preprocessing workflow.

CHAPTER 5

RESULTS AND DISCUSSION

A methodology and the architecture of models for monophonic music generation and methods of evaluation have been presented in chapter 3 with the thorough detail of all seven trained models architecture, each with its own meta-parameters and preprocessed approaches as well its dataset type which trained on it. In this chapter, the results of these methods and their related tables and plots have been presented and discussed. A series of trials to train the models are performed, a workflow of processes and some important codes related to these issues are settled in Appendix 1, and screenshot samples of implementation have been fixed in Appendix 2. The results are given and evaluated, and the quality of the generated samples is measured and defined, as well as the accuracy of the models is discussed.

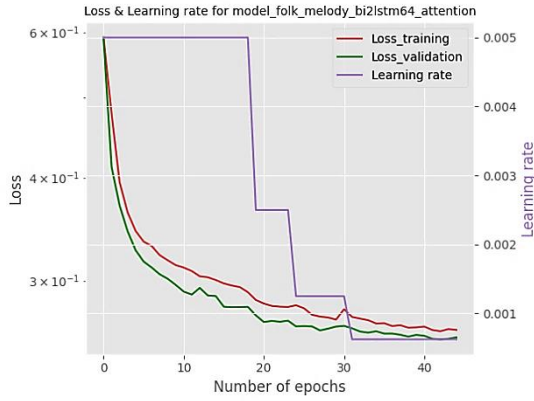
5.1 Results

Through the models analysis and objectively comparing them to each other via their generated melodies with the assist of experts in the field of music and composing melodies, as well as through analysis of the results of the charts and the values of the losses functions and the models' accuracies which obtained from the implementation of the codes for each model, the good investigation can be obtained, and the impacts of changing each independent variables in the models on the generated melodies can be felt and have been verified as follow.

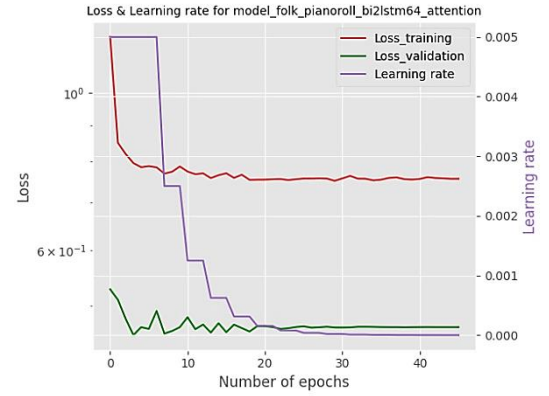
5.1.1 Format encoding impacts

The comparison has been applied between two models, No.1 with No.2 and No.3 with No.4 for revealing the influence of format encoding on the overall processes and the quality of the melody samples that are generated. The only difference between these two models architecture is the encoding format, all things are the same except the ways have been used for encoding the MIDI songs from the dataset, two approaches have been used

for this purpose, the first is the same as proposed by Google magenta which is melody format encoding applied to model No.1 (Waite, 2016), and the second approach is pianoroll format encoding which is applied to model No.2. By comparing loss graphs of these two models as shown in Figure 5.1, the important result has been noticed; the stagnation of validation loss at (0.45508) after five epochs and did not reduce due to the overfitting early when the model with pianoroll encoding trained, while the model with melody format achieves a lower validation loss of (0.27749), this means that this model has learned more than the other one.



(a) Loss graph for melody encoding



(b) Loss graph for pianoroll encoding

Figure 5.1: Loss graphs for models No.1 and No.2.

Thus, by making a comparison between the tunes that have been gotten from these two models themselves as well as with the seed sample, as shown in Figure 5.2, and through the analysis of music experts, the following inferences as shown in Table 5.1 have been grasped, and the conclusion is that the magenta melody format encoding has better learning ability at the sequencing of melodic steps. It is better than the pianoroll format encoding. In other words, model No.1 is better.

Table 5.1: Experts objectively comparison for revealing the encoding influences.

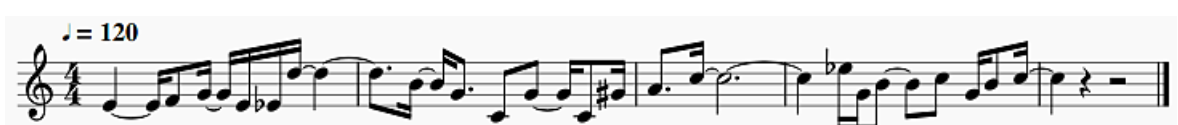
	Samples generated with trained model No.1, melody encoding	Samples generated with trained model No.2, pianoroll encoding
Tempo	Same as the seed sample and training set 120 Beats/minute	Same as the seed sample and training set 120 Beats/minute
Time signature	Same as the seed sample and training set, 4/4	Same as the seed sample and training set, 4/4
Vocabulary range	A smaller range of notes has been used, hence can be said it is closer to the seed sample.	A larger range of notes is used, hence can be said it is further from the seed sample.
Note time range	The smaller range has been used; dotted quarter, quarter, and eighth notes.	The larger range between a whole note and sixteenth notes has been used.
Key signature / Scale	Same as the seed sample or training set songs (C major or A minor)	Differ from the seed sample or training set songs there are some notes out of the scale range of (C major or A minor)
Melodically	Melodically samples are nearer to the seed sample and training set songs (Irish folk).	Less melodic characteristic samples.
Rhythmically	Less rhythmic characteristic samples.	More rhythmical samples. They are closer to classical music than simple folk songs.



(a) The seed sample used in sampling processes



(b) Melody samples generated by the use of trained Model No.1 (melody encoding).

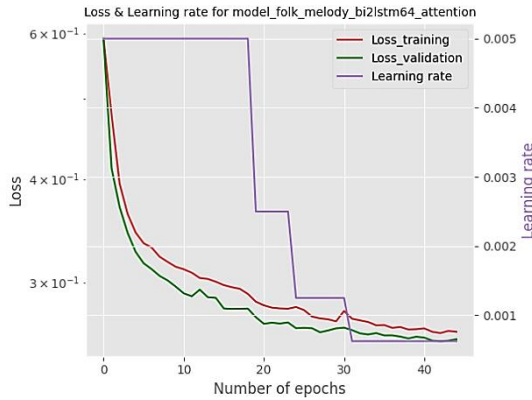


(c) Melody samples generated by the use of trained Model No.2 (pianoroll encoding).

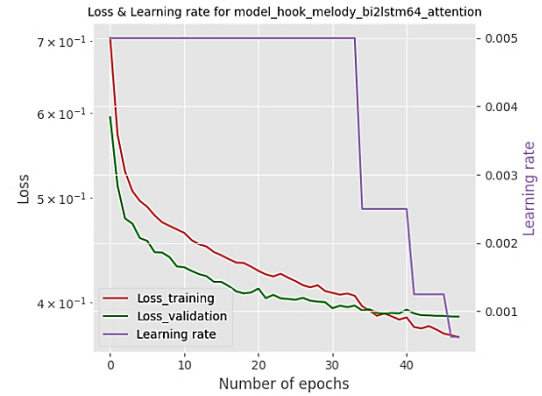
Figure 5.2: Seed and generated samples for showing format encoding impacts.

5.1.2 Dataset nature impacts (Single-genre vs Multi-genre)

The distinction between two trained models, No.1 and No.3, has been extended to expose the effect of the style of the dataset on the overall processes and the nature of the produced melody samples. They have exactly the same architecture design but trained on two different datasets stylistically. The significant and predicted outcome was found by comparing loss graphs of these two models No.1 and No.3 as seen in Figure 5.3 due to the mixture of songs in the multi-genre dataset; stagnation of validity loss at (0.38790) and did not decrease due to early overfitting. But the model trained on the homogenous property single-genre dataset achieves a lower validity loss of (0.27749), which suggests that more has been learned from this model.



(a) Loss graph for Single-genre



(b) Loss graph for Multi-genre

Figure 5.3: Loss graphs for models No.1 and No.3.

Therefore, in Table 5.2, and through the music experts study, the following inferences have been contrasted among the tunes generated by the models themselves and the seed sample which is shown in Figure 5.4. Hence we can conclude that model No.1 learned more, and has better results due to training on the single-genre dataset, it is better at mimicking the characteristics associated with the pitch if compared with model No.3 which is trained on a

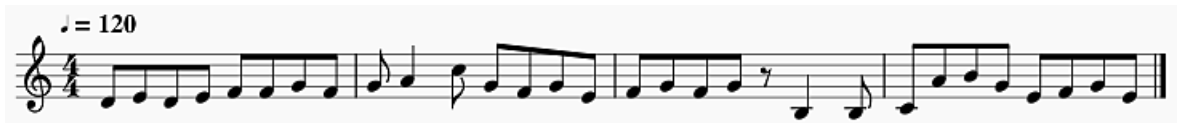
multi-genre dataset. Clearly, we can see that the model has a weak ability to mimic both the dataset's pitch and rhythmic aspects. This is the same as humans learning to compose a melody. The one who studies and attempts to learn a type of music will learn better than the one who tries to learn several styles in the same period.

Table 5.2: Experts objectively comparison for revealing the dataset nature influences.

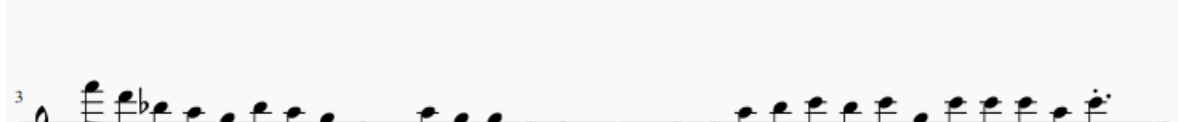
	Samples generated with trained model No.1 on the Single-genre dataset	Samples generated with trained model No.3 on the Multi-genre dataset
Tempo	Same as the seed sample and training set 120 Beats/minute	Same as the seed sample and training dataset 120 Beats/minute
Time signature	Same as the seed sample and training set, 4/4	Same as the seed sample and training dataset, 4/4
Vocabulary range	A small range of notes have been used, hence can be said it is closer to the seed sample and the training set.	A larger range of notes is used same as the training set which has a larger vocabulary range, but far from the seed sample nature.
Note time range	The smaller range has been used; dotted quarter, quarter, and eighth notes.	The larger range has been used; the sixteenth notes can be seen as well.
Key signature / Scale	Same as the seed sample or training set songs (C major or A minor).	There are a few false notes out of the scale of the seed sample.
Melodically	Melodically the samples are closer to the seed sample and training set songs (Irish folk; single-genre).	Melodically the samples have both characteristics of the seed sample and training set songs (multi-genre).
Rhythmically	Less rhythmic.	More rhythmic.



(a) The seed sample used in sampling processes



(b) Melody samples generated by the use of trained Model No.1 (Single-genre).

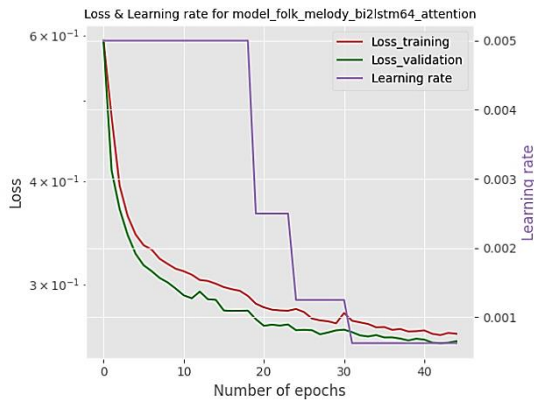


(c) Melody samples generated by the use of trained Model No.3 (Multi-genre).

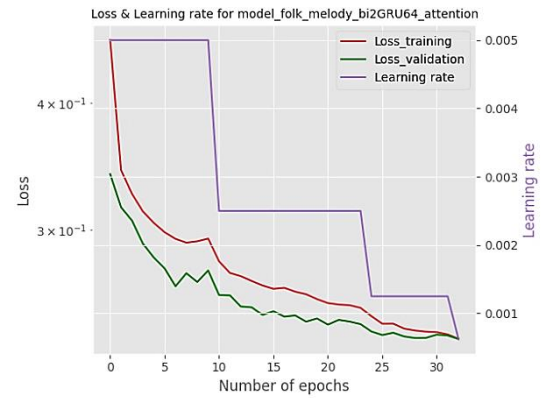
Figure 5.4: Seed and generated samples to show dataset nature impacts.

5.1.3 RNN type impacts (LSTM vs GRUs)

The aim of the comparison between trained models, No.1 and No.7, is to determine the influences of the accuracy of the two main modified and improved types of Recurrent Neural Network; LSTM and GRUs, both of them trained on the same dataset and with the same computation resources. Via the investigation and comparing their loss graphs as shown in Figure 5.5, this fact can be noticed; There is no significant difference between the results for the two models both models validation loss reached nearly 0.2, but the model with GRUs architecture needs less time to lean same as the model with LSTM design.



(a) Loss graph for LSTM



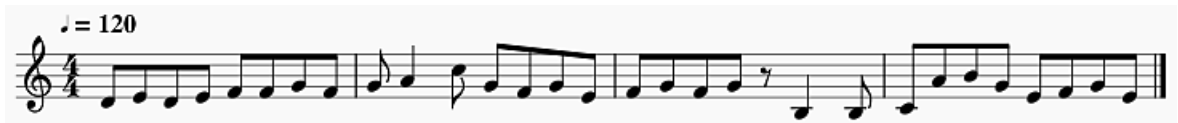
(b) Loss graph for GRUs

Figure 5.5: Loss graphs for models No.1 and No.7.

Thus by comparing the tunes derived from these two models themselves as well as the seed melody, as seen in Figure 5.6, and by evaluating the music experts, the music samples generated by both models are so closer to each other and has the same structure as the seed sample as well as the training set songs, except the existing the one false tone out of the chosen scales (C major and A minor) in the second sample of the model No.7 (GRUs). So we can conclude that Model No.1 (LSTM) has results of generated melody slightly better.



(a) The seed sample used in sampling processes



(b) Melody samples generated by the use of trained Model No. 1.

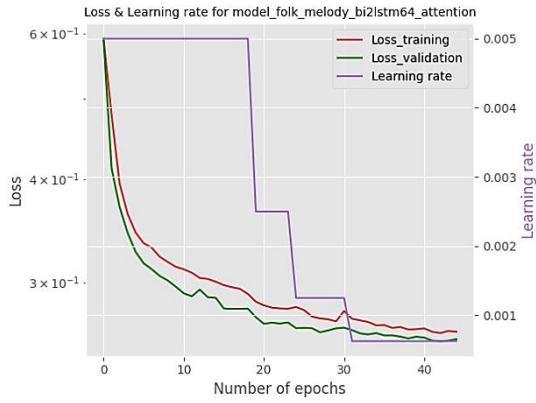


(c) Melody samples generated by the use of trained Model No.7.

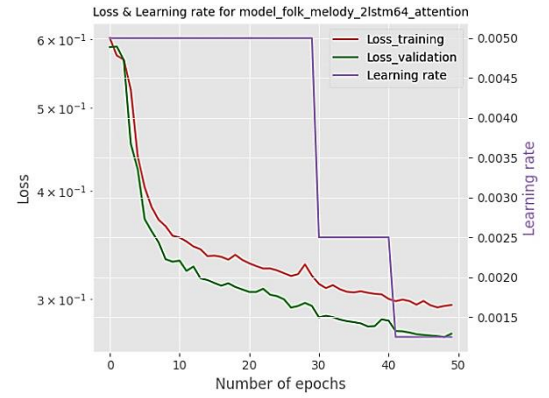
Figure 5.6: Seed and generated samples to demonstrate the RNN type impacts.

5.1.4 Bi-directional mechanism impacts

The only existing difference in these two trials between both models No.1 and No.5 is that the former has a Bi-directional mechanism but the latter's design is without a Bi-directional mechanism. By looking closely at their loss graphs in Figure 5.7, we settle that they both approximately reached the same value of validation loss, knowing that the first model has 128 cells for each layer of its hidden layers, and that is because the cells are counted in both directions from the beginning of the hidden layer to the end and back in the opposite direction, while the second has just 64 cells.



(a) Loss graph for Bi-directional



(b) Loss graph for Non-Bi-directional

Figure 5.7: Loss graphs for models No.1 and No.5.

Here the goal is to compare the effect of the Bidirectionality on the generated melody results that shown in Figure 5.8, via the experts' assessment and from their summaries fixed in Table 5.3, we can infer that the melodies produced by model No.1 with Bi-directional mechanism are better than those generated by model No.5, which has no Bi-directional mechanism. The Bi-directional model generates pieces that tend to match the melodic structure of the seed sample structure more effectively. More detail on this

mechanism in chapter 3, section 3.4. The key inference is that, with the use of Bi-directionality, doubling the number of cells increases the consistency of the samples.

Table 5.3: Experts objectively comparison for revealing the Bidirectionality influences.

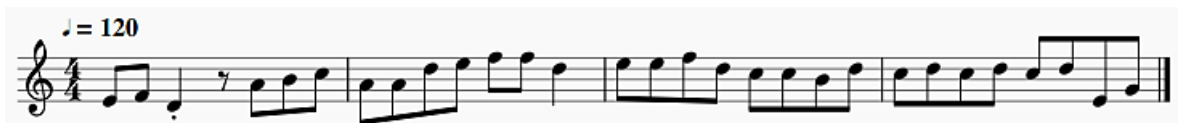
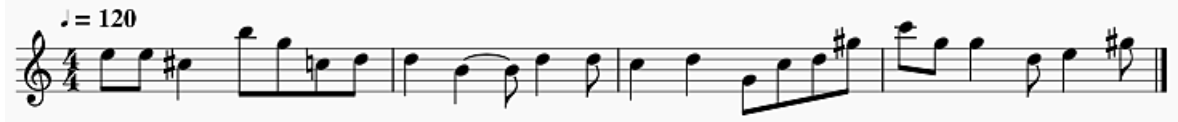
	Samples generated with trained model No.1 using the Bi-directional mechanism	Samples generated with trained model No.5 without using the Bi-directional mechanism
Tempo	Same as the seed sample and training dataset 120 Beats/minute	Same as the seed sample and training dataset 120 Beats/minute
Time signature	Same as the seed sample and training dataset, 4/4	Same as the seed sample and training dataset, 4/4
Vocabulary range	Approximately one octave (12-13 notes) has been used	One and have octaves(20-21 notes has been used
Note time range	So close to the seed sample as well as the training set.	So close to the seed sample as well as the training set.
Key signature / Scale	Same as the seed sample or training set songs (C major or A minor).	There are some notes out of the scale range of (C major or A minor).
Melodically	They have an emotional effect, musically more close to monophonic folk songs.	They have an emotional effect musically but not much as model No.1 results. More repeating tones can be seen.
Rhythmically	Less rhythmic character melodies like the properties of the seed melody and training set folk songs.	Same as results in model No.1



(a) The seed sample used in sampling processes



(b) Melody samples generated by the use of trained Model No.1 (Bi-directional).

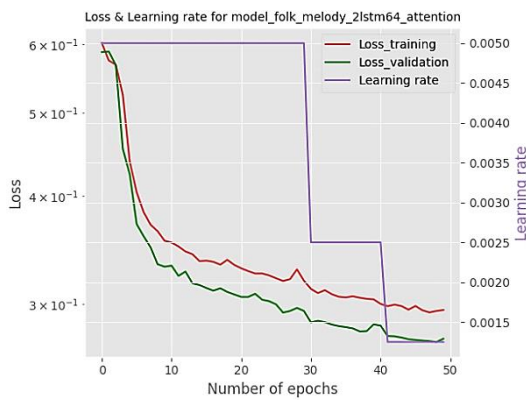


(c) Melody samples generated by the use of trained Model No.5 (Non-Bi-directional).

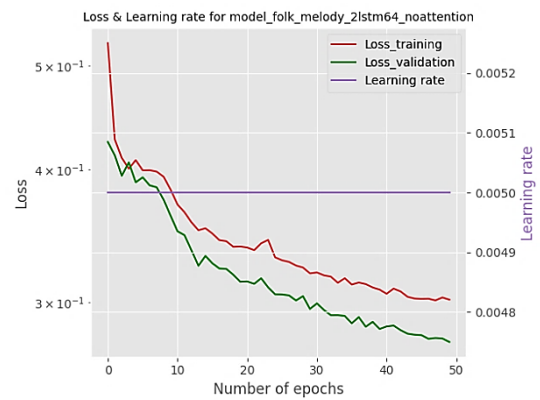
Figure 5.8: Seed and generated samples to demonstrate Bidirectionality impacts.

5.1.5 Attention mechanism impacts

Models No.5 differs from model No.6 in the architecture design, the former has an Attention mechanism, but the latter is without it. By observing their loss graphs in Figure 5.9, we determine that both models achieved the same amount of validation loss roughly. But their generated samples as shown in Figure 5.10 did not have the same musical quality as analyzed and compared by the experts. We can notice that model No.6 did not learn to mimic the key signature of the trained set and the seed sample (C major or A minor), due to lack of Attention, each sample has a different scale and dissimilar key signature, They are similar to atonal music, not melodically the same as the seed sample and training set songs (Irish folk). “Atonality is simply the absence of tonality, tonality being the musical system based on major and minor keys” (Miles Hoffman, 2018). The samples were generated through model No.5 melodically they are more similar to the seed sample and training set songs (Irish folk), they are more emotional and closer to tonal music with a few dissonance tones. In conclusion, we can decide that the model with the Attention mechanism has better-generated samples and the model has been learned more than the other one that has no Attention mechanism. It keeps the seed sequence composition much better. For more detail about this mechanism see chapter 3, section 3.5, and chapter 2 section 2.1.2.9



(a) Loss graph for Attention

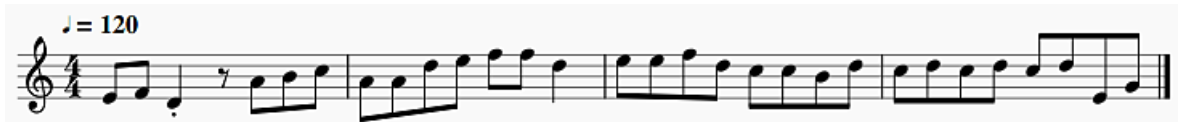
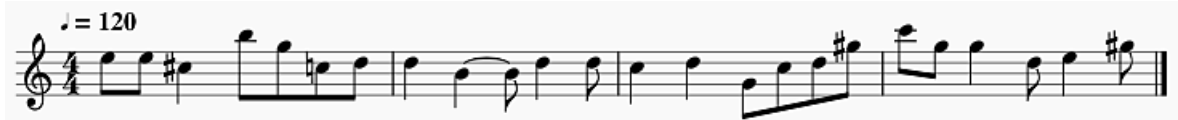


(b) Loss graph for Non-Attention

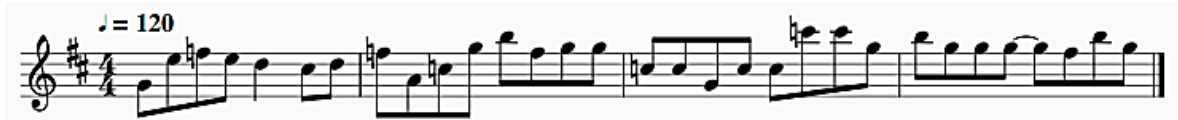
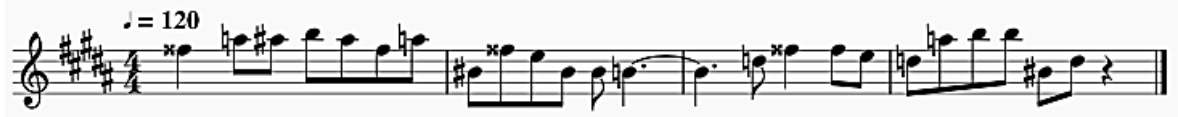
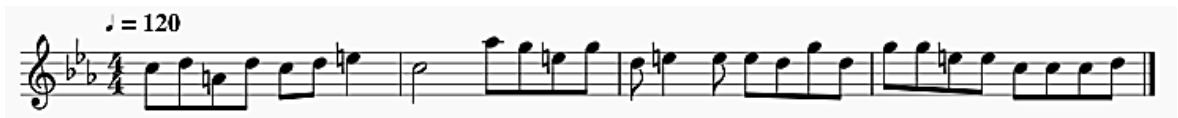
Figure 5.9: Loss graphs for models No.5 and No.6



(a) The seed sample used in sampling processes



(b) Melody samples generated by the use of trained Model No.5 (Attention).

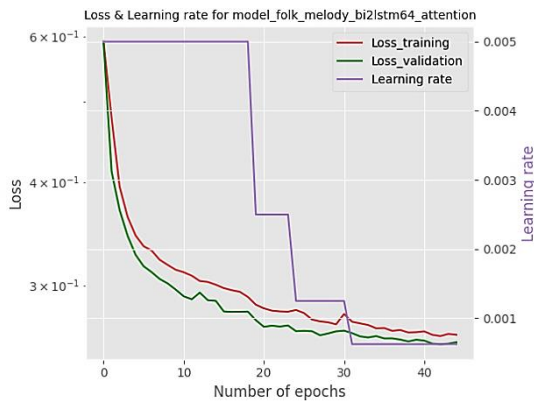


(c) Melody samples generated by the use of trained Model No.6 (Non-Attention).

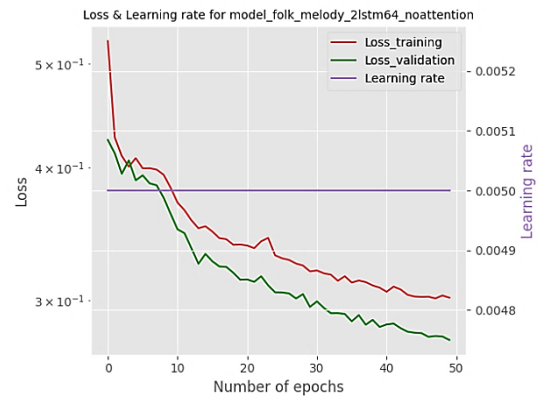
Figure 5.10: Seed and generated samples to demonstrate Attention mechanism impacts.

5.1.6 Bi-directional & Attention impacts

The comparison has been applied between two models, No.1 and No.6 for revealing the influence of two important mechanisms together on the overall processes and the quality of the melody samples that are generated. From comparing their loss graphs as shown in Figure 5.11, we cannot observe a big distinction except the relation between the learning rate and the change of losses. Hence the fixed learning rate can be noticed in the case of model No.6 while losses decrease. Both trained models tend to reduce validation loss from infinity nearly to 0.2. Therefore with the aid of expert analysis, many differences can be observed: the samples related to the model without using both mechanisms have Atonal properties, the scale is not the same as the seed or the training set. The trained model could not mimic the melodic structure of the training set; the results have different key signatures, as well as rhythmically the samples are not related to the original dataset or the seed sample, see Figure 5.12. In conclusion, we can adopt that the model with the Attention mechanism and Bidirectionality keeps the seed sequence composition much better and has better generated samples. The model has been learned more than the other one that has no these two mechanisms.



(a) Loss graph for Bi-directional & Attention



(b) Loss graph for Non-Bi-directional & Non-Attention

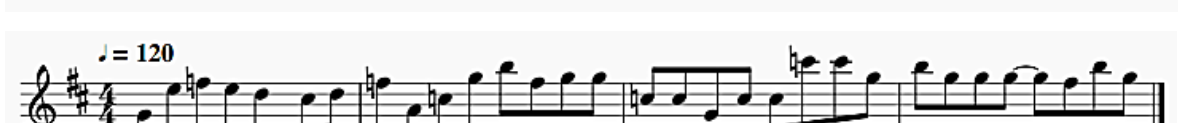
Figure 5.11: Loss graphs for models No.1 and No.6.



(a) The seed sample used in sampling processes



(b) Melody samples generated by the use of trained Model No. 1 (Bi-directional & Attention).



(c) Melody samples generated by the use of trained Model No. 6 (Non-Bi-directional & Non-Attention).

Figure 5.12: Seed and generated samples to show Bi-directional & Attention impacts.

5.2 Human Evaluation

For the purpose of audience qualitatively assessment of melody samples generated via the proposed model, as well as to determine the participant's background musically knowledge and expertise. We prepared a specific survey form for assessing melodies subjectively and defining the assessor listener's grade in the music industry. The form which is shown in Appendix 3, attached with the mixture of (10) four bars length melodies made up of (5) melodies from the original training set selected randomly and (5) melodies generated from model No. 1 which is trained on the folk dataset that encoded with magenta melody format as advanced by the Magenta team (Waite, 2016), assuming that this model is the best model which assessed its generated melodies by proficiencies, and it has the following architecture:

- Input Layer has (63) time steps and 58 vocabulary sizes from the melody encoding format of the folk data set.
- 2 layers of Bi-directional LSTM, each with (64) cells and (0.4) dropout rate.
- On top of the second layer, the attention layer has been applied.
- (One) time step and (58) vocabulary size of the output layer.

It's important to note that the model was optimized with an Adam optimizer with a (0.005) learning rate, and was trained for 50 epochs. For generating samples with the temperature equals to 1(the effect of temperature values have been explained in Section 3.7), we have selected 4 bars of the melody from the same genre that the model trained with as a seed sequence. This helps the model to generate samples of greater quality. The seed was in D major with the name "The Banks of the Ilan" (<https://thesession.org/> Retrieved 11 June 2020).

Participants were asked to listen to the mixture of the samples and to answer the questions. Of course, the first one is about the rhythmic stability of melodies as they inquire if they should tap along with the melodies. Aniruddh et al., (2009) identified that humans have a high degree of flexibility in synchronizing Beat Perception; they can easily synchronize

with music rhythm through a variety of body parts movements such as head bobbing, foot, hand and finger tapping, side-to-side tilting, etc.

Also, the audience was asked to answer the latter question about how they affect emotionally just after listening to the melodies. This is for the aim of targeting the melody content itself. The overall grades, as done by (Huang et al., 2020) were on a Likert scale of 5 points. In chapter 3 there is a detailed explanation of this survey methodology.

5.2.1 Survey analysis and results:

We will try to discuss and give appropriate graphs and plots of the results in this section after collecting (50) answered forms (see Appendix 3) and implementing the respective codes (see Appendix 1). From Figure 5.13, due to their skill rate, we can see the distribution of participants, for each level we have 10 persons. 1 is representing the lowest level and 5 is the highest musical experience level.

Participant distribution by musical experience level

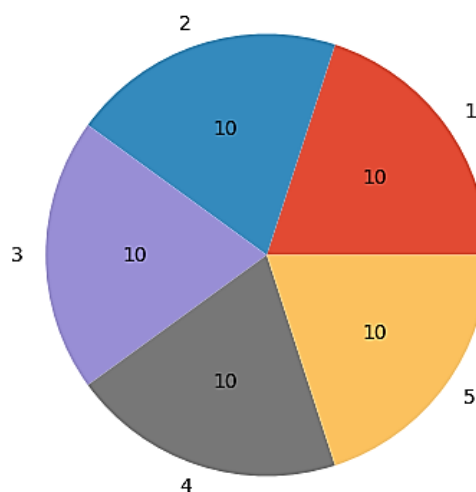


Figure 5.13: Distribution of participants according to experience levels.

The difference in the assessment of the generated samples can be perceived in terms of participant's musical expertise when the overall collected forms have been classified into two main groups; group1 consists of the participants with level 1 and level 2 which demonstrate in general the low expertise level. Thus the remaining levels from 3 to 5 are categorized as group2 and it represents the high-level expertise level participants. Statistically, table 5.4 has been shown the results, as well as Figure 5.14, represents the results in a bar plot of mean and standard deviation. Both groups felt the difference between the original and produced models, albeit with a slight difference. They do not give high rates to all samples. This is due to the influence of eastern culture on the participants' taste for music; we know that both groups of the samples have a western musical character. In high-level group 2, the standard deviation for the assessment is lower than in the low-level group, explaining that the score is accurate.

Table 5.4: Evaluation results of the participants per group experience level.

participants per low & high groups	Mean	SD
Group 1 per original rhythm stability	3.950	1.052
Group 1 per generated rhythm stability	3.770	1.112
Group 1 per original melody pleasing	3.840	1.037
Group 1 per generated melody pleasing	3.650	1.081
Group 2 per original rhythm stability	3.960	0.965
Group 2 per generated rhythm stability	3.693	1.064
Group 2 per original melody pleasing	3.840	1.020
Group 2 per generated melody pleasing	3..667	1.056

Participant evaluation results per group level (low & high)

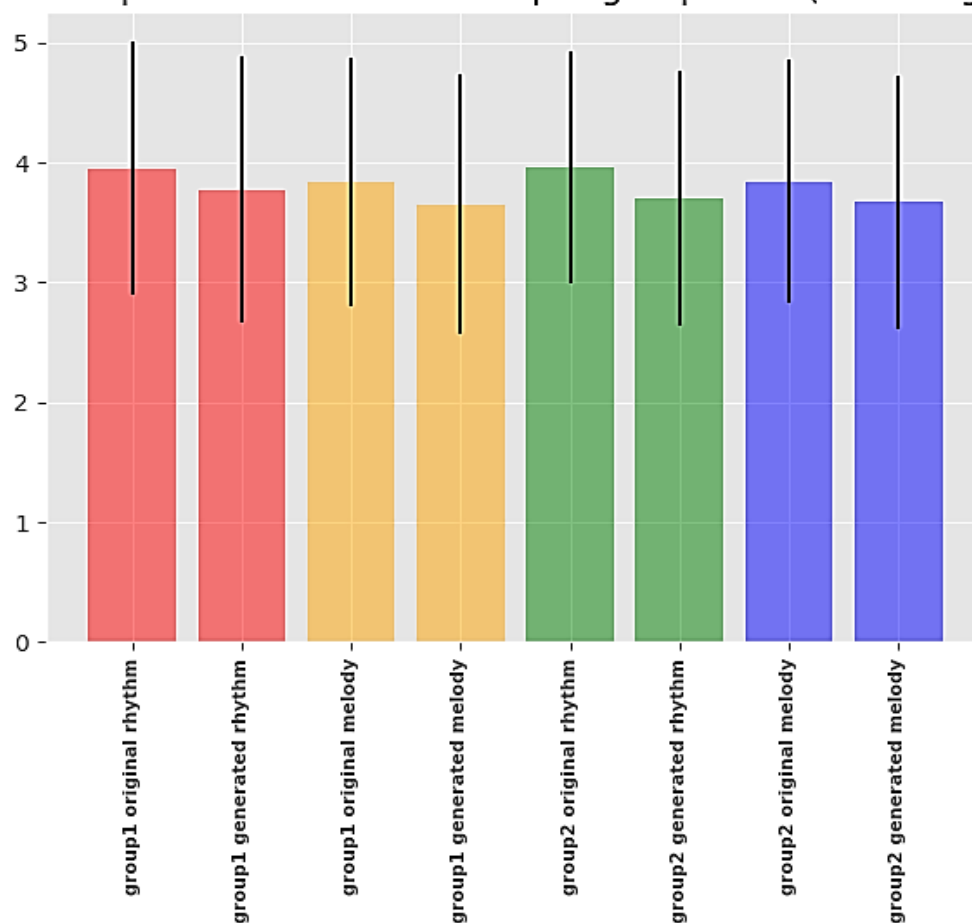


Figure 5.14: Participants' evaluation results per low and high experience level.

The overall assessment by the participants, rhythmically and melodically, also represents the convergence of the results of mean and standard deviation values as shown in both Table 5.5 and Figure 2.15; there is a slight distinction between the results, this means that the generated samples from the model are so close to the original songs.

Table 5.5: Evaluation results of all participants.

All participants	Mean	SD
Rhythm stability per original samples	3.956	1.001
Rhythm stability per generated samples	3.724	1.084
Melody pleasing per original samples	3.840	1.027
Melody pleasing per generated samples	3.660	1.066

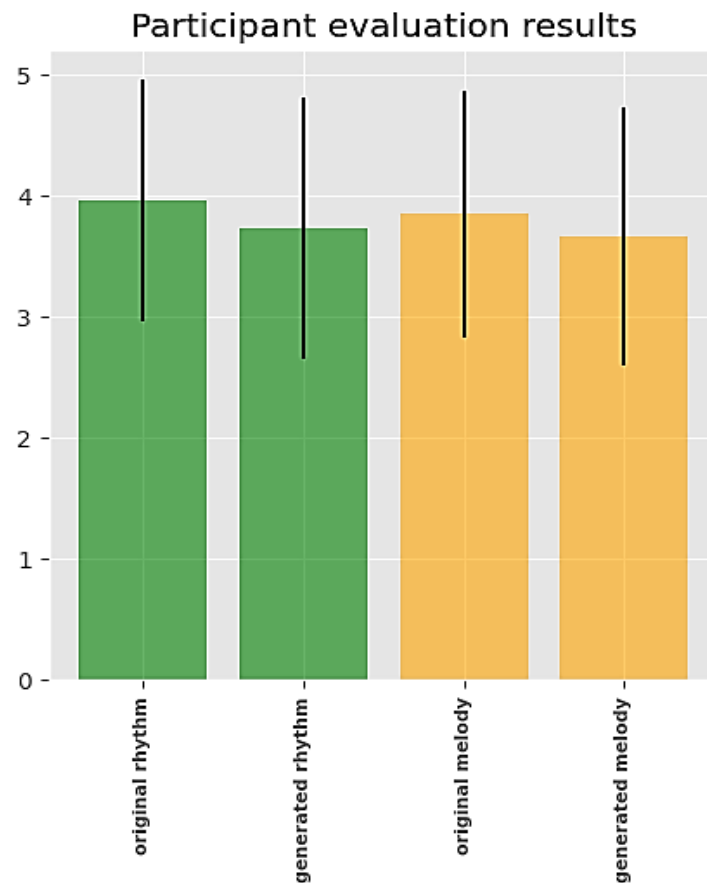


Figure 5.15: The overall participant evaluation results

By looking at Table 5.6, we infer that by comparing the mean values of the generated sample with original training set samples for both evaluation, rhythmically (3,820 to 3,420) and melodically (3,780 to 3,420), the highest level participants assessed the greater discrepancy between generated and original samples see Figure 5.16.

Table 5.6: Evaluation results of all participants per each experience level.

Participants per experience level	Mean	SD
Level 1 per original rhythm stability	3.780	1.082
Level 1 per generated rhythm stability	3.560	1.169
Level 1 per original melody pleasing	3.700	0.964
Level 1 per generated melody pleasing	3.300	1.100
Level 2 per original rhythm stability	4.120	0.993
Level 2 per generated rhythm stability	3.980	1.010
Level 2 per original melody pleasing	4.00	1.086
Level 2 per generated melody pleasing	4.100	0.938
Level 3 per original rhythm stability	3.600	0.922
Level 3 per generated rhythm stability	3.940	1.114
Level 3 per original melody pleasing	3.760	1.047
Level 3 per generated melody pleasing	3.960	1.011
Level 4 per original rhythm stability	3.960	1.019
Level 4 per generated rhythm stability	4.060	0.925
Level 4 per original melody pleasing	3.800	1.058
Level 4 per generated melody pleasing	3.820	1.033
Level 5 per original rhythm stability	3.820	0.931
Level 5 per generated rhythm stability	3.420	1.041
Level 5 per original melody pleasing	3.780	0.944
Level 5 per generated melody pleasing	3.420	1.079

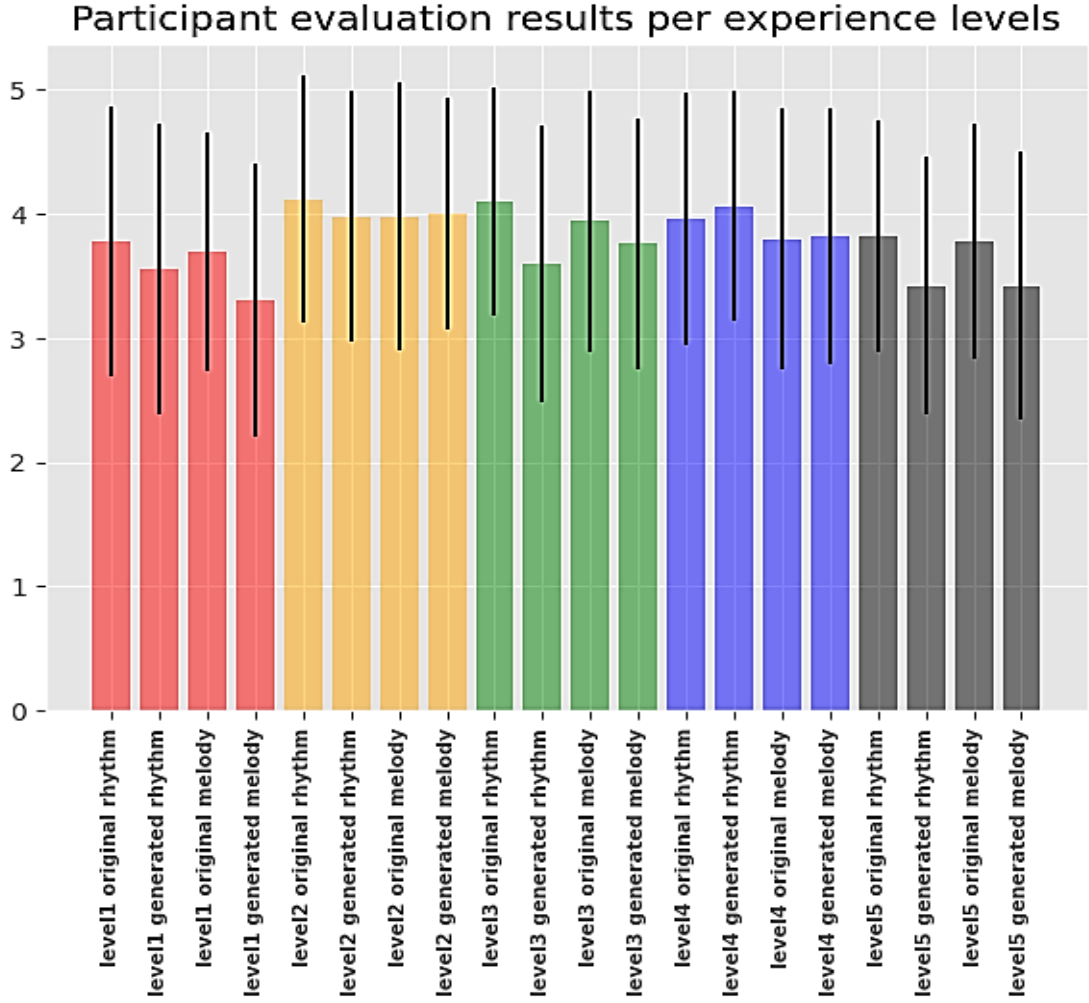


Figure 5.16: Participants' evaluation results for each experience level.

As a whole conclusion giving the small rates to all generated as well as original melodies, in melodically and rhythmically assessment is due to the influence of eastern culture on the participants' taste for music; we know that both groups of the samples have a western musical character. The final judgment would be that the samples generated are slightly less musically enjoyable and also slightly less rhythmically balanced for audiences. This leads to infer that model No.1 of two layers of Bi-directional LSTM, 64 cells for each with applying Attention mechanism on single-genre dataset has satisfactory results in generating monophonic music. This was concluded also in both expert's evaluation and through the analysis of the training and validation losses previously in this section.

CHAPTER 6

CONCLUSION

The main goal of this project is to delve into the study of how to use Artificial Intelligence deep learning techniques, for the purpose of generating and composing monophonic music, by designing models using the Python programming language and its own libraries in this field and trying to teach the models on different datasets in terms of the style (single-genre and multi-genre) to determine More efficient and effective techniques, and also to demonstrate effects of some independent variables such as (magenta melody and pianoroll format encoding, Bidirectionality and Attention mechanism, RNN types, and datasets nature) on the dependent variable (the generated melody). For these purposes, seven models, each with a different architecture design have been used and trained on a specific type of dataset to learn and to generate new melody samples, the results have been analyzed and comparison has been done between samples of the models with the assist of academic musicians and experts composers, as well as a survey has been conducted to evaluate subjectively the generated music samples by the best model selected via the musician experts assessment.

In conclusions, In case of using the same independent variables except for encoding format of dataset MIDI songs which trained the model on it, the generated melodies outcome with pianoroll encoding format has more rhythmically enhancement melodies comparing with magenta melody encoding format result samples, but melodically, the latter has better emotional melodies and so closer to trained dataset and the seed sample.

For studying the influence of the dataset specifications on the generated melody samples quality, dealing with two sets of data that varied in stylistic homogeneity (single-genre and multi-genre) for training has been tackled. So we can conclude that the produced melodies have a more melodic and sensitive personality when the model trained on single-genre folk.

As a result, the fact which has been formulated is that the use of 2 layers Bi-directional LSTM with attention mechanism is a very promising candidate for generating emotional

melodies similar to the original dataset and the used seed sample and relatively close to GRUs results. In the case of using GRUs, the training process has been stopped at 35 epochs judging by the validation loss behavior compared with the same design with LSTM; this means GRUs needs less implementation time for achieving relatively same results. Conclude point here is that the LSTM outperforms the GRUs with similar global architecture meta-parameters but more training time and epoch's number.

Bidirectionality and attention mechanisms have a positive impact on models learning capability, significantly the better melodic structure samples have been produced by models with the use of these two great mechanisms. Exactly as deduced and proven by Magenta, the results with the model which is used attention makes it possible for the model to learn longer-term dependencies and can produce melodies with longer themes effortlessly. This is because, without storing information in the RNN cells; Attention helps the model to handle past information more efficiently and effectively.

Future Project Expansions

Here is the list of possible future works of this thesis project:

- Designing models using a combination of RNN hidden layers, for example combining BLSTM with BGRUs.
- Training models with different augmentation of different key signature melodies to improve generated sample results harmonically.
- Training models on multiple tracks/instruments and generating the same melodies (Homophonic, Polyphonic, and Heterophonic)
- Designing models for dealing with Eastern musical scales.
- Training using Eastern melody datasets.
- Using encoder-decoder architecture with sequence to sequence methodology.
- Using a dataset with other music notations and representations than MIDI.
- Incorporating dynamics and emotional expression of notes to expand this project to deal with human feelings.
- Testing models with different Temperature values in the generating phase.

REFERENCES

- Agarwala N., Inoue, Y., & Sly. A. (2017). Music Composition using Recurrent Neural Networks. Retrieved 10 April 2020 from <https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1174/reports/2762076.pdf>
- AI Wiki, Recurrent Neural Network (RNN). Retrieved 02 May 2020 from <https://docs.paperspace.com/machine-learning/wiki/recurrent-neural-network-rnn>
- AIVA, (2020), Artificial Intelligence Virtual Artist. Retrieved 12 March 2020 from <https://www.aiva.ai/>
- Amidi, (2019) Recurrent Neural Networks. Retrieved 20 Feb. 2020. <https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks>.
- Ammer, (2004). *The facts on file dictionary of music*. Infobase Publishing. P.16
- Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural Machine Translation by Jointly Learning to Align and Translate. *CoRR*, *abs/1409.0473*. arxiv.org/abs/1409.0473
- Ben Dunnett (2019), Musical Structures. Retrieved: 10 May 2020 from <https://www.musictheoryacademy.com/understanding-music/musical-structures/>
- Boulanger-Lewandowski, N., Bengio, Y., & Vincent, P. (2012). Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *Computer Science, Machine Learning* <https://arxiv.org/abs/1206.6392>
- Briot, J., & Pachet, F. (2017). Music Generation by Deep Learning - Challenges and Directions. *Computer Science, Engineering*. <https://arxiv.org/abs/1712.04371>
- Briot, J., Hadjeres, G., & Pachet, F. (2017). Deep Learning Techniques for Music Generation - A Survey. *ArXiv*, *abs/1709.01620*. <https://arxiv.org/abs/1709.01620>
- Brook, B. (1965). The Simplified Plaine and Easie Code System for Notating Music. In *A Proposal for International Adoption. Fontes Artis Musicae*, 12(2/3), (PP 156-160). Retrieved October 11, 2020, from <http://www.jstor.org/stable/23504707>

- Brownlee Jason, (2017). *Master Machine Learning Algorithms*. (Chapter 9, P 30)
- Byrd, Donald & Isaacson, Eric. (2003). A Music Representation Requirement Specification for Academia. *Computer Music Journal - COMPUT MUSIC J.* 27. (PP 43-57). 10.1162/014892603322730497.
- Chen Frank, (2016), AI, Deep Learning, and Machine Learning: A Primer. Retrieved 10 Feb.2020 from <https://a16z.com/2016/06/10/ai-deep-learning-machines/>
- Cho, K., Merrienboer, B.V., Bahdanau, D., & Bengio, Y. (2014). On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. *SSST@EMNLP*.
- Cho, K., Merrienboer, B.V., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *Computer Science, Mathematics*. <https://arxiv.org/pdf/1406.1078.pdf>
- Choi, K., Fazekas, G., & Sandler, M. (2016). Text-based LSTM networks for Automatic Music Composition. *Computer Science, Artificial Intelligence*. <https://arxiv.org/abs/1604.05358>
- Chollet et al., (2020). Keras: Deep Learning for Python. Retrieved April 2020 from <https://github.com/keras-team/keras>
- Walshaw Chris. About ABC Notation, Retrieved 10 Jan. 2020, <http://abcnotation.com>.
- Chung, J., Gülçehre, Ç. Cho, K., & Bengio, Y. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *Computer Science*. <https://arxiv.org/abs/1412.3555>
- Colab, Making Music with Magenta, Creating New Sequences. retrieved March 2020 https://colab.research.google.com/notebooks/magenta/hello_magenta/hello_magenta.ipynb#scrollTo=SIYDyTA0-UJT
- Sapp. Craig Stuart, (2005). Online Database of Scores in the Humdrum File Format. *Royal Holloway, University of London*, <http://ismir2005.ismir.net/proceedings/3123.pdf>.

- De Coster, M. (2017). Polyphonic music generation with style transitions using recurrent neural networks. Retrieved 20 March 2020 from <https://lib.ugent.be/en/catalog/rug01:002367003>
- DiPietro, R., & Hager, G. D. (2019). Deep learning: RNNs and LSTM. In Handbook of Medical Image Computing and Computer Assisted. *The Elsevier and MICCAI Society Book Series 2020*, (PP 503-519). <https://doi.org/10.1016/B978-0-12-816176-0.00026-0>
- Dong, (2018). SAM-bach: A deep generative model for bach chorale generation. Retrieved 08 May 2020, from <https://escholarship.mcgill.ca/concern/theses/8910jw92v>
- Dong, H., Hsiao, W., Yang, L., & Yang, Y. (2018). MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment. *AAAI*. <https://arxiv.org/abs/1709.06298>
- Eck D. and Schmidhuber J, (2002). A first look at music composition using lstm recurrent neural networks, *Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale*, vol. 103. <http://www.iro.umontreal.ca/~eckdoug/blues/IDSIA-07-02.pdf>
- Felbo, B., Mislove, A., Søgaaard, A., Rahwan, I., & Lehmann, S. (2017). Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm. *EMNLP*. <https://arxiv.org/abs/1708.00524>
- Gal, Y., & Ghahramani, Z. (2016). A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. *NIPS*.29,(PP 1019-1027). arxiv.org/abs/1512.05287
- Globant, (2017), Artificial Intelligence As a Music Composer. Retrieved 10 March 2020 from <https://stayrelevant.globant.com/en/artificial-intelligence-composing-original-music/>
- Good M., 2001 MusicXML for Notation and Analysis. *The Virtual Score: Representation, Retrieval, Restoration, Volume 12 of computing in musicology*, MIT Press (P. 113-124) <https://doi.org/10.7551/mitpress/2058.003.0010>
- Goodfellow, Bengio, and Courville, (2016). *Deep learning*. MIT Press.
- Graves, A. (2013). Generating Sequences with Recurrent Neural Networks. *Computer Science, Neural and Evolutionary Computing*, <https://arxiv.org/abs/1308.0850>

- Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural networks: the official journal of the International Neural Network Society*, 18(5-6), (P. 602–610). <https://doi.org/10.1016/j.neunet.2005.06.042>
- Gregor, K., Danihelka, I., Graves, A., Rezende, D., & Wierstra, D. (2015). DRAW: A Recurrent Neural Network For Image Generation. *ICML*. <https://arxiv.org/abs/1502.04623>
- Gudivada, V., Apon, A., & Ding, J. (2017). Data quality considerations for big data and machine learning: Going beyond data cleaning and transformations. *International Journal on Advances in Software 10.1*, (pp. 1 – 20).
- Hadjeres, G., & Nielsen, F. (2017). Interactive Music Generation with Positional Constraints using Anticipation-RNNs. *ArXiv*, *abs/1709.06404*. <https://arxiv.org/abs/1709.06404>
- Hao-Wen Dong, Wen-Yi Hsiao, and Yi-Hsuan Yang, (2018.) .Pypianoroll: Open Source Python Package for Handling Multitrack Pianorolls. *In Late-Breaking Demos of the 19th International Society for Music Information Retrieval Conference (ISMIR)*, <https://salu133445.github.io/pypianoroll/>
- Henry Earl; Snodgrass Jennifer; and Piagentini Susan, (2018), *Fundamentals of Music: Rudiments, Musicianship, and Composition*. Published by Pearson ISBN: 0134491386,9780134491387.
- Hilscher, M., & Shahroudi, N. (2018). Music Generation from MIDI datasets. Retrieved 08 Feb. 2020.
- Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation* 9(8) (PP. 1735–1780). <https://doi.org/10.1162/neco.1997.9.8.1735>
- Hosken, 2010, *An Introduction to Music Technology*, First ed. Taylor & Francis e-Library. Routledge. <https://doi.org/10.4324/9780203849514>
- Hsiao, (2019). Lead Sheet Dataset. Retrieved 05 January 2020. <https://github.com/wayne391/Lead-Sheet-Dataset>

- Huang, C.A., Vaswani, A., Uszkoreit, J., Shazeer, N., Simon, I., Hawthorne, C., Dai, A.M., Hoffman, M., Dinculescu, M., & Eck, D. (2018). Music Transformer. *arXiv: Learning*. Retrieved 11 May 2020. <https://arxiv.org/abs/1809.04281>
- Huber, D. (2007). The MIDI Manual, Third Edition: A Practical Guide to MIDI in the Project Studio. *What Is MIDI?* (PP. 1-12). Focal Press. ISBN 9780240807980. <https://www.sciencedirect.com/science/article/pii/B9780240807980500032>
- IraKurshunovava (2019). Folk music style modeling using LSTMs. Retrieved February 2020. <https://github.com/IraKorshunova/folk-rnn>.
- Itoh, K., Sakata, H., Igarashi, H., & Nakada, T. (2019). Automaticity of pitch class-color synesthesia as revealed by a Stroop-like effect. *Consciousness and Cognition*, 71, (PP 86-91). <https://doi.org/10.1016/j.concog.2019.04.001>.
- Johnson, D. D. (2017). Generating polyphonic music using tied parallel networks. *In International conference on evolutionary and biologically inspired music and art* (pp. 128-143). Springer, Cham.
- Jordan, (1997) Serial order: A parallel distributed processing approach. *Advances in psychology*, vol. 121, pp. 471–495.
- Karpathy, (2015). The Unreasonable Effectiveness of Recurrent Neural Networks. Retrieved 20 March 2020 from <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>
- Karpathy, et al., (2016). CS231n: Convolutional Neural Networks for Visual Recognition. Retrieved 15 March 2020 from <https://cs231n.github.io/neural-networks-3/>
- Keith, (2015). The Session data. Retrieved February 2020 <https://github.com/adactio/TheSession-data>.
- Kennedy, M., & Kennedy, J. (2013). *The Oxford dictionary of music*. Oxford University Press. eISBN: 9780191744518, DOI: 10.1093/acref/9780199578108.001.0001.
- Kingma, D.P., & Ba, J. (2015). Adam: A Method for Stochastic Optimization. *CoRR*. <https://arxiv.org/abs/1412.6980>
- Kitagawa, Y. (1999), Handbook of Music Theory. *RittorMusic, Tokyo, Japan*

- Kostadinov Simeon, (2017) Understanding GRU Networks. Retrieved 19 June 2020 from <https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be>
- Laden, B., & Keefe, D. (1989). The Representation of Pitch in a Neural Net Model of Chord Classification. *Computer Music Journal*, 13(4), (PP 12-260. doi:10.2307/3679550
- Levitin, Daniel J., (2006) *This Is Your Brain on Music: The Science of a Human Obsession*. New York, N.Y.: Dutton.
- Liu, I., & Ramakrishnan, B. (2014). Bach in 2014: Music Composition with Recurrent Neural Network. *Computer Science, Artificial Intelligence*. <https://arxiv.org/abs/1412.3191>
- Marinescu (2019). Bach 2.0. Retrieved 10 March 2020 from <https://amarinescu.ro/bach-prev/>
- Marr Bernard, (2019) *Artificial Intelligence in Practice: How 50 Successful Companies Used AI and Machine Learning to Solve Problems*, JohnWiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, United Kingdom ISBN: 978-1-119-54896-6 (ePDF).
- Mauthes, N. (2018). VGM-RNN: Recurrent Neural Networks for Video Game Music Generation. *San Jose State University*. <https://doi.org/10.31979/etd.87bh-zeeh>
- McClelland Calum, (2017). The Difference Between Artificial Intelligence, Machine Learning, and Deep Learning. Retrieved 10 March 2020 from <https://medium.com/iotforall/the-difference-between-artificial-intelligence-machine-learning-and-deep-learning-3aa67bff5991>
- Miles Hoffman (2018), A Minute with Miles. *In a production of South Carolina Public Radio, made possible by the J.M. Smith Corporation*. <https://www.southcarolinapublicradio.org/post/atonality-vs-dissonance>
- Mitroi, (2019), Symbolic Music Generation with RNNs, retrieved 02 Feb. 2020 from https://github.com/cristianmtr/master_thesis_symbolic_music_generation
- MMA, *The MIDI Manufacturers Association*, (2020). Retrieved 12 Jan 2020 <https://www.midi.org/>.

- Mogren, O. (2016). C-RNN-GAN: Continuous recurrent neural networks with adversarial training. *ArXiv, abs/1611.09904*. <https://arxiv.org/abs/1611.09904>
- Mozer, M. C., (1994.) Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing, *Connection Science*, vol. 6, no. 2, (p. 247–280). <https://doi.org/10.1080/09540099408915726>
- Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.
- Musicmap (2020), The Genealogy and History of Popular Music Genres from Origin till Present (1870-2016). Retrieved Apr. 2020 from <https://musicmap.info>
- Ng Andrew, (2019), Bi-directional RNN. Retrieved 03 May 2020 from <https://www.coursera.org/lecture/nlp-sequence-models/Bi-directional-rnn-fyXnn>.
- Nicholson, Schuyler & Kim, Eun-Jin. (2016). Structures in Sound: Analysis of Classical Music Using the Information Length. *Entropy*. 18. 258. 10.3390/e18070258.
- Olah & Carter, (2016). Attention and Augmented Recurrent Neural Networks. Retrieved 10 Feb. 2020 from <https://distill.pub/2016/augmented-rnns>
- Oliveira, H.M., & Oliveira, R.D. (2017). Understanding MIDI: A Painless Tutorial on Midi Format. *ArXiv, abs/1705.05322*. <https://arxiv.org/abs/1705.05322>
- Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., & Kavukcuoglu, K. (2016). WaveNet: A Generative Model for Raw Audio. *Computer Science, Sound*. <https://arxiv.org/abs/1609.03499>
- Patel, A., Iversen, J., Bregman, M., & Schulz, I. (2009). Studying synchronization to a musical beat in nonhuman animals. *Annals of the New York Academy of Sciences*, 1169(1), (PP 459-469). DOI: 10.1111/j.1749-6632.2009.04581.x.
- Picheny Michael, Ramabhadran Bhuvana, Stanley F. Chen, Markus Nussbaum-Thom, (2016). Advanced Neural Networks, Lecture 14. Watson Group IBM T.J. Watson Research Center Yorktown Heights, New York, USA. Retrieved 06 April 2020 from <http://www.ee.columbia.edu/~stanchen/spring16/e6870/slides/lecture14.pdf>

- Anwla P. K. (2020). Recurrent Neural Network (RNN) architecture explained in detail. Retrieved 20 Dec. 2019. <https://towardsmachinelearning.org/recurrent-neural-network-architecture-explained-in-detail/>
- Abiyev R, Arslan M., Idoko J.B.,Sekeroglu B., Ilhan A., (2020) Identification of Epileptic EEG Signals Using Convolutional Neural Networks. *Appl. Sci.* 10(12). <https://doi.org/10.3390/app10124089>
- Abiyev R.H., Arslan M., (2020) Head mouse control system for people with disabilities. *Expert Systems*, 37. <https://doi.org/10.1111/exsy.12398>
- Abiyev R.H., Arslan M., Idoko J.B., (2020) Sign Language Translation Using Deep Convolutional Neural Networks. *KSII Transactions on Internet and Information Systems*, 14(2). <https://doi.org/10.3837/tiis.2020.02.009>.
- Idoko J.B., Abiyev R.H., Arslan M., (2019) Impact of Machine Learning Techniques on Hand Gesture Recognition. *Journal of Intelligent & Fuzzy Systems*, 37(3), 4241-4252. <https://doi.org/10.3233/JIFS-190353>
- Pinkerton, R. C. (1956). *Information theory and melody*. *Scientific American*, 194(2), (P.77–86). <https://doi.org/10.1038/scientificamerican0256-77>.
- Roberts A., (2019) Music and Art Generation with Machine Intelligence. Retrieved 21 Jan. 2020. https://github.com/magenta/magenta/blob/master/magenta/models/melody_rnn
- Ruder, S. (2016). An overview of gradient descent optimization algorithms. *ArXiv*, *abs/1609.04747*. <https://arxiv.org/abs/1609.04747>
- Schuster, M. and Paliwal, K. (1997). Bi-directional recurrent neural networks. *In IEEE Transactions on Signal Processing*, vol. 45, no. 11, (pp. 2673-2681). doi: 10.1109/78.650093.
- Sigurgeirsson (2019), Classical Piano Composer, retrieved 03 April 2020 from <https://github.com/Skuldur/Classical-Piano-Composer>
- Simon and Oore, (2017). Performance RNN: Generating music with expressive timing and dynamics. Retrieved May 2020 from <https://magenta.tensorflow.org/performance-rnn>

- Simon, I., Roberts, A., Raffel, C., Engel, J., Hawthorne, C., & Eck, D. (2018). Learning a Latent Space of Multitrack Measures. *ArXiv*, *abs/1806.00195*. <https://arxiv.org/abs/1806.00195>
- Sloboda, J. A. (2010). *Music in everyday life: The role of emotions*. In P. N. Juslin & J. A. Sloboda (Eds.), *Series in affective science. Handbook of music and emotion: Theory, research, applications* (p. 493–514). Oxford University Press.
- Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *JMLR. org 15*, (P. 1929-1958). <https://jmlr.org/papers/volume15/srivastava14a.old/srivastava14a.pdf>
- Sturm, B.L., Santos, J.F., Ben-Tal, O., & Korshunova, I. (2016). Music transcription modelling and composition using deep learning. *ArXiv*, *abs/1604.08723*. <https://arxiv.org/abs/1604.08723>
- Summerville, A., & Mateas, M. (2016). Super Mario as a String: Platformer Level Generation Via LSTMs. *Computer Science, Neural and Evolutionary Computing*. <https://arxiv.org/abs/1603.00930>
- Sutskever, Martens, and Hinton, (2011). Generating text with recurrent neural networks. *In Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. (pp. 1017–1024). Bellevue, WA, USA.
- Sutskever, Vinyals, and Le, (2014), Sequence to sequence learning with neural networks. In NIPS, *Computer Science*. (pp. 3104– 3112). <https://arxiv.org/abs/1409.3215>
- Tyler, Rinker. (2014). On the Treatment of Likert Data. *Research Gate* https://www.researchgate.net/publication/262011454_Likert
- Vandenneucker Dominique, All You Need for Music Software Development. 2020, Retrieved 05 Feb. 2020. <http://www.music-software-development.com/music-data-structures.html>
- Velardo (2020), Generating Melodies with RNN LSTM, retrieved 11 May 2020 from https://github.com/cristianmtr/master_thesis_symbolic_music_generation.
- Waite E., (2016). Generating long-term structure in songs and stories. Retrieved 02 June 2020 from <https://magenta.tensorflow.org/2016/07/15/lookback-rnn-attention-rnn>

- Walder, C. (2016). Modeling Symbolic Music: Beyond the Piano Roll. *ACML*.
<https://arxiv.org/abs/1606.01368>
- Weng Lilian, (2018). Attention? Attention! Retrieved 21 March 2020 from
<https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html>.
- White John, (2019), Differences between AI, Machine learning and Deep learning.
 Retrieved 05 April 2020 from <https://www.msystaining.com/articles/ai-machine-learning/differences-between-ai-machine-learning-and-deep-learning/>
- Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., Zemel, R., & Bengio, Y. (2015). Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. *Computer Science, Machine Learning*. <https://arxiv.org/abs/1502.03044>
- Yang Z., Yang D., Dyer C., He X., Smola A., and Hovy E., (2016). Hierarchical attention networks for document classification. *In Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: Human language technologies*, (pp. 1480–1489).
- Ycart, A., & Benetos, E. (2017). A study on LSTM networks for polyphonic music sequence modelling. In *18th International Society for Music Information Retrieval Conference*. <https://qmro.qmul.ac.uk/xmlui/handle/123456789/24946>

APPENDICES

APPENDIX 1
Ethical Approval Document



ETHICAL APPROVAL
DOCUMENT

Date:03/05/2021

To the Graduate School of Applied Sciences

For the thesis project entitled “Monophonic Music Generation using Artificial Intelligence through Deep Learning Techniques”, the researchers declare that they did not collect any data from human/animal or any other subjects. Therefore, this project does not need to go through the ethics committee evaluation.

Title: Prof. Dr.

Name Surname: Rahib Abiyev

Signature:

A handwritten signature in black ink, appearing to read "Rahib Abiyev".

Role in the Research Project: Supervisor

APPENDIX 2

Similarity Report

The screenshot shows a web browser window with the Turnitin interface. The page title is 'turnitin.com/t_inbox.asp?r=39.9921965170847&svr=38&lang=en_us&aid=55438470'. The Turnitin logo is visible at the top left. Below the logo, there are tabs for 'Assignments', 'Students', 'Grade Book', 'Libraries', 'Calendar', 'Discussion', and 'Preferences'. The main content area shows 'NOW VIEWING: HOME > CLIMATE > CLIMATE1'. Under 'About this page', it states: 'This is your assignment inbox. To view a paper, select the paper's title. To view a Similarity Report, select the paper's Similarity Report icon in the similarity column. A ghosted icon indicates that the Similarity Report has not yet been generated.'

The submission is titled 'climate1' and is in the 'INBOX | NOW VIEWING: NEW PAPERS' section. A 'Submit File' button is present. The table below lists the submission details:

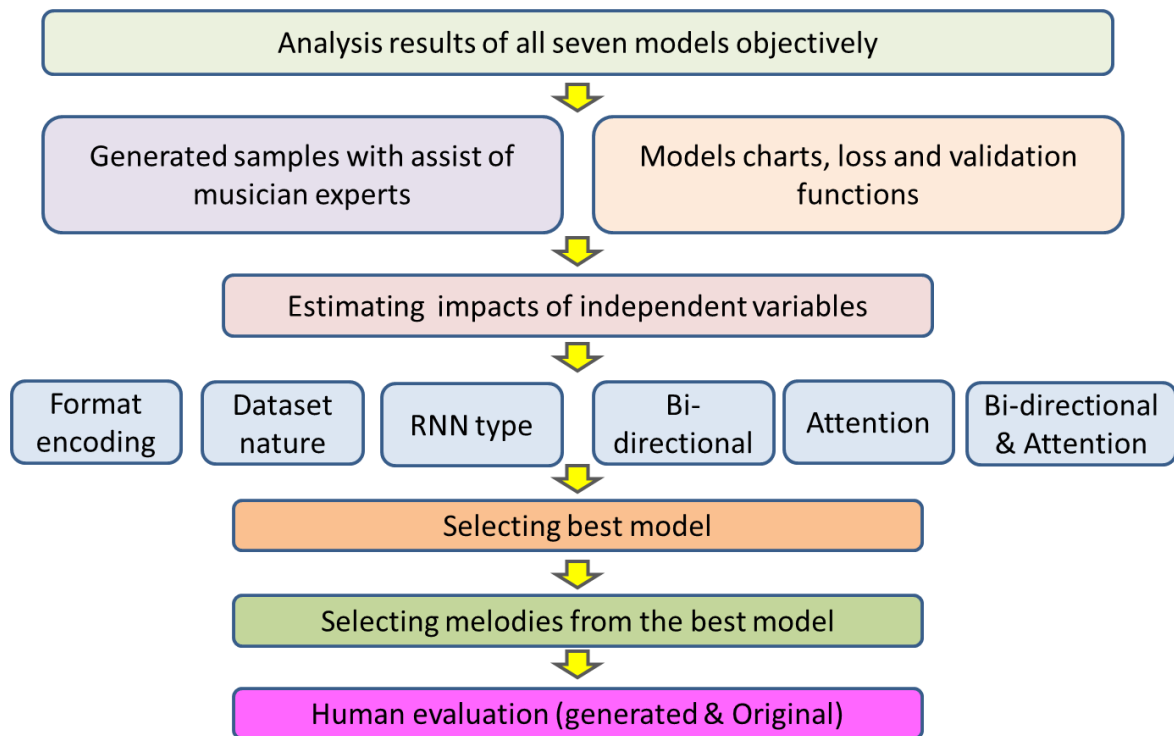
	AUTHOR	TITLE	SIMILARITY	GRADE	RESPONSE	FILE	PAPER ID	DATE
<input type="checkbox"/>	Ara Ahmed Sharif	Abstract	0%	---	---		1553502763	08-Apr-2021
<input type="checkbox"/>	Ara Ahmed Sharif	Ch.6. Conclusion and Future Expansion	0%	---	---		1553506946	08-Apr-2021
<input type="checkbox"/>	Ara Ahmed Sharif	Ch3. Methodology and Architecture	1%	---	---		1553504838	08-Apr-2021
<input type="checkbox"/>	Ara Ahmed Sharif	Ch5. Results and Discussion	1%	---	---		1553506284	08-Apr-2021
<input type="checkbox"/>	Ara Ahmed Sharif	Ch1. Introduction	2%	---	---		1553503238	08-Apr-2021
<input type="checkbox"/>	Ara Ahmed Sharif	ch4. Collecting and Processing Data	3%	---	---		1553505830	08-Apr-2021
<input type="checkbox"/>	Ara Ahmed Sharif	Ch2. Background and State of Art	6%	---	---		1562293224	18-Apr-2021

At the bottom of the page, there is a taskbar showing the Windows logo, search icon, and several application icons. The system clock shows 12:41 PM on 4/18/2021.

Supervisor: Prof.Dr.Rahib Abiyev

APPENDIX 3

Workflow & Implemented Codes Samples



Preprocessing codes:

filter_4/4.py

```

import os
import tqdm
import pretty_midi
import glob

def filter_midis(files, dst):
    """for each file, check if timing is only 4/4 and there is only one
    track
    Then save the ones in dst folder"""
    print('filtering to 4/4...')
    for fpath in tqdm.tqdm(files):

```

```

pm = pretty_midi.PrettyMIDI(fpath)
if len(pm.time_signature_changes) == 1:
    ts = pm.time_signature_changes[0]
    if ts.numerator == 4 and ts.denominator == 4:
        # single track
        if len(pm.instruments) == 1:
            fname = fpath.split(os.path.sep)[-1]
            pm.write(os.path.join(dst, fname))

def main(files, dst):
    filter_midis(files, dst)

```

transpose.py to C-major and A-minor

```

import os
import sys
from glob import glob

import music21
import numpy as np
from tqdm import tqdm

majors = dict([("A-", 4), ("G#", 4), ("A", 3), ("A#", 2), ("B-", 2), ("B", 1), ("C", 0), ("C#", -1), ("D-", -1), ("D", -2), ("D#", -3), ("E-", -3), ("E", -4), ("F", -5), ("F#", 6), ("G-", 6), ("G", 5)])
minors = dict([("G#", 1), ("A-", 1), ("A", 0), ("A#", -1), ("B-", -1), ("B", -2), ("C", -3), ("C#", -4), ("D-", -4), ("D", -5), ("D#", 6), ("E-", 6), ("E", 5), ("F", 4), ("F#", 3), ("G-", 3), ("G", 2)])

def main(files, dst_dir):
    print('transposing...')
    for file in tqdm(files):
        # transpose
        score = music21.converter.parse(file)
        key = score.analyze('key')
        if key.mode == "major":
            halfSteps = majors[key.tonic.name]

        elif key.mode == "minor":
            halfSteps = minors[key.tonic.name]

        newscore = score.transpose(halfSteps)

        file = os.path.abspath(file)
        unique_name = ''.join(file.split(os.path.sep)[-1].split(".mid")[:-1])
        new_file_path = os.path.join(dst_dir, unique_name + "_transposed.mid")
        newscore.write("midi", new_file_path)

```

procedure.py

```
import subprocess
from glob import glob
import os
import filter_four_four
import numpy as np
import encode
import transpose
import monophonize
import sys
import encode_pianoroll
import comparisons

sys.path.append("/Users/arasharif/opt/anaconda3/envs/magenta/lib/python3.7/site-packages/note_seq")

from melody_encoder_decoder import MelodyOneHotEncoding

def main(top_dir, max_sequences, nr_bars, bar_len, max_seq_len):
    # original midi files
    orig_dir = os.path.join(top_dir, '1_Origin')
    orig_dir_glob = os.path.join(orig_dir, "*.mid")

    # contains 4/4 only midis
    four_four_dir = os.path.join(top_dir, "2_fourfour")
    if not os.path.exists(four_four_dir):
        os.mkdir(four_four_dir)

    # contains the transposed midi files
    transposed_dir = os.path.join(
        top_dir, "3_transposed")
    if not os.path.exists(transposed_dir):
        os.mkdir(transposed_dir)

    # contains the transposed, monophonic melodies files
    monophonic_dir = os.path.join(
        top_dir, "4_mono"
    )
    if not os.path.exists(monophonic_dir):
        os.mkdir(monophonic_dir)

    # 100 random files to be used in evaluation
    comparison_dir = os.path.join(top_dir, "7_comparison")
    if not os.path.exists(comparison_dir):
        os.mkdir(comparison_dir)

    # contains the transposed, split, magenta one hot encoded dataset
    # as a big mmap file
    magenta_dir = os.path.join(top_dir, "5_encoded")
    if not os.path.exists(magenta_dir):
        os.mkdir(magenta_dir)

    pianoroll_dir = os.path.join(top_dir, "6_pianoroll")
    if not os.path.exists(pianoroll_dir):
        os.mkdir(pianoroll_dir)
```

```

magenta_dataset_file = os.path.join(top_dir, "dataset.dat")

pianoroll_dataset_file = os.path.join(top_dir, "pianoroll.dat")

# FILTER TO 4/4 ONLY
four_four_dir_files = glob(os.path.join(four_four_dir, "*.mid"))
if len(four_four_dir_files) == 0:
    orig_files = glob(orig_dir_glob)
    filter_four_four.main(orig_files, four_four_dir)
else:
    print('skipping filtering to 4/4 as directory %s is not empty' %
          four_four_dir)

# TRANPOSE THE 4/4 MIDI FILES
transposed_files = glob(os.path.join(transposed_dir, "*.mid"))
if len(transposed_files) == 0:
    transpose.main(
        glob(os.path.join(four_four_dir, "*.mid")),
        transposed_dir
    )
else:
    print("skipping transposing as %s is not empty" % transposed_dir)

# monophonize
mono_files = glob(os.path.join(monophonic_dir, "*.mid"))
if len(mono_files) == 0:
    monophonize.main(
        glob(os.path.join(transposed_dir, "*.mid")),
        monophonic_dir
    )
else:
    print("skipping monophonize as %s is not empty" % monophonic_dir)

# choose 100 random samples, take first 4 bars
mono_files = glob(os.path.join(monophonic_dir, "*.mid"))
comparisons.main(
    mono_files,
    comparison_dir
)

encoder = None
min_note = None
max_note = None
# ENCODING AND SPLITTING THE TRANPOSED MIDIS INTO MAGENTA FORMAT AND
THEN
# CREATING ONE BIG MMAP FILE OF ONE-HOT ENCODED SEQUENCES
if len(glob(os.path.join(magenta_dir, '*.npy')) == 0:
    encoder = encode.main(
        glob(os.path.join(monophonic_dir, "*.mid")),
        magenta_dir,
        nr_bars,
        max_seq_len,
        bar_len
    )
else:
    print('skipping encoding into melody as directory %s was not
empty' %magenta_dir)

```

```

min_note, max_note = np.load(os.path.join(top_dir, 'min_max.npy'))
encoder = MelodyOneHotEncoding(min_note, max_note+1)

if not os.path.exists(magenta_dataset_file): # dat file doesnt exist
    encode.dat_file(
        glob(os.path.join(magenta_dir, "*.npy")),
        max_sequences,
        magenta_dataset_file,
        max_seq_len,
        encoder
    )
    print('dataset at ', magenta_dataset_file)
else:
    print('skipping creating dataset file as %s exists'
          %(magenta_dataset_file))

## encode into pianoroll
if len(glob(os.path.join(pianoroll_dir, "*.npy"))) == 0:
    encode_pianoroll.main(
        glob(os.path.join(monophonic_dir, "*.mid")),
        pianoroll_dir,
        nr_bars,
        max_seq_len,
        bar_len,
        min_note,
        max_note
    )
else:
    print('skipping encoding into pianoroll as directory %s was not
empty' %pianoroll_dir)

if not os.path.exists(pianoroll_dataset_file): # dat file doesnt
exist
    encode_pianoroll.dat_file(
        glob(os.path.join(pianoroll_dir, "*.npy")),
        max_sequences,
        pianoroll_dataset_file,
        max_seq_len,
        min_note,
        max_note
    )
    print('dataset at ', pianoroll_dataset_file)
else:
    print('skipping creating dataset file as %s exists'
          %(pianoroll_dataset_file))

```

Training Model Codes:

config.py for configuration

```
import os

EPOCHS = 50
BATCH_SIZE = 128

datasets = {
    "folk_melody": {
        "path":
"/Users/arasharif/Desktop/Ara_Master_Thesis/Folk_Dataset/dataset.dat",
        "shape": (100000, 64, 58)
    },
    "folk_pianoroll": {
        "path":
"/Users/arasharif/Desktop/Ara_Master_Thesis/Folk_Dataset/pianoroll.dat",
        "shape": (100000, 64, 56)
    },
    "hook_melody": {
        "path":
"/Users/arasharif/Desktop/Ara_Master_Thesis/hooktheory_dataset/dataset.dat",
        "shape": (93667, 64, 90)
    },
    "hook_pianoroll": {
        "path":
"/Users/arasharif/Desktop/Ara_Master_Thesis/hooktheory_dataset/pianoroll.dat",
        "shape": (95661, 64, 88)
    },
}

# main.py for single-genre dataset

import subprocess
from glob import glob
import os
import filter_four_four
import encode
import transpose

import procedure

if __name__ == "__main__":
    max_sequences = 100000
    nr_bars = 4
    bar_len = 16 # based on steps_per_quarter=4 in midi_file_to_melody
in / Users/arasharif/opt/anaconda3/envs/magenta/gm/melodies_lib.py
    max_seq_len = nr_bars * bar_len

    # top dir
    top_dir = "/Users/arasharif/Desktop/Ara_Master_Thesis/Folk_Dataset"
    procedure.main(top_dir, max_sequences, nr_bars, bar_len, max_seq_len)
```

main.py for designing & training LSTM model.

```
import sys
import pickle
import argparse
from utils import *
from callbacks import get_callbacks, delete_epoch_counters
from glob import glob
import shutil
from generator import BatchGenerator
from keras_self_attention import SeqWeightedAttention
from sklearn.model_selection import train_test_split
from build_dataset import min_max_from_folder
import keras
import os
import config
from architecture import new_architecture

def get_data(dataset):
    dpath = dataset['path']
    dataset = np.memmap(dpath, mode="r",
                        dtype="uint8", shape=dataset['shape'])
    x = dataset[:, :-1]
    y = dataset[:, -1]
    X_train, X_val, y_train, y_val = train_test_split(
        x, y, test_size=0.2, random_state=42, shuffle=True)
    print('we have %s training files and %s validation files' %
          (len(y_train), len(y_val)))

    return X_train, X_val, y_train, y_val

def get_model_id(args):
    # model_multi-genre_melody_bi2lstm64_attention
    model_id = None
    if args.new:
        model_id = "model_"
        model_id += args.dataset
        if args.bi:
            model_id += "_bi"
        else:
            model_id += "_"

        model_id += "%slstm%s_" % (args.layers, args.cells)
        if args.att:
            model_id += "attention"
        else:
            model_id += "noattention"
        print("generated model id from args: %s" % model_id)
    else:
        model_id = args.id
        print("using existing model id %s" % model_id)
    return model_id

def get_model_dir(args):
    model_id = get_model_id(args)
    model_dir =
os.path.abspath(os.path.join('/Users/arasharif/Desktop/Ara_Master_Thesis',
model_id))
    if os.path.exists(""):
        model_dir = os.path.join("", model_id)
```

```

        if not os.path.exists(model_dir):
            os.mkdir(model_dir)
    else:
        if not os.path.exists(model_dir):
            os.mkdir(model_dir)
    print('model id: ', model_id)
    print('model dir: ', model_dir)
    return model_dir

def get_model(args, dshape):
    model_dir = get_model_dir(args)

    model = None
    loss = 'categorical_crossentropy'
    optimizer = keras.optimizers.Adam(lr=0.005)

    if args.new:
        print('generating NEW model...')
        model = new_architecture(
            dshape[1]-1,
            dshape[2],
            args.layers,
            args.bi,
            args.att,
            args.cells
        )
        # copy arch to folder
        shutil.copy('architecture.py', model_dir)
        model_json = model.to_json()
        model_json_path = os.path.join(model_dir, "model.json")
        print('storing model json in %s' % model_json_path)
        with open(model_json_path, "w") as json_file:
            json_file.write(model_json)
        # delete epoch counters
        delete_epoch_counters(model_dir)
        model.compile(
            loss=loss,
            optimizer=optimizer
        )

    else:
        print('using existing model...')
        model_json_path = os.path.join(model_dir, "model.json")
        model = keras.models.Model_from_json(open(model_json_path, "r").read(),
            custom_objects=SeqWeightedAttention.get_custom_objects())

        model_weights_path = os.path.join(model_dir, "model.h5")
        print('loading existing weights from %s...' % model_weights_path)
        model.load_weights(model_weights_path)
        model.compile(
            loss=loss,
            optimizer=optimizer
        )

    print(model.summary())

    return model, model_dir

def get_dataset_name(args):
    if args.dataset:
        return config.datasets[args.dataset]
    elif args.id:

```



```

        for name in config.datasets.keys():
            if name in args.id:
                print("found name of dataset in model id : %s" %name)
                return config.datasets[name]
    else:
        print("Dataset could not be deduced...")
        sys.exit(1)

def main(args):
    dataset = get_dataset_name(args)
    X_train, X_val, y_train, y_val = get_data(dataset)
    model, model_dir = get_model(args, dataset['shape'])

    verbosity, callbacks = get_callbacks(model_dir, args, model)

    model.fit(
        X_train, y_train,
        epochs=config.EPOCHS,
        batch_size=config.BATCH_SIZE,
        callbacks=callbacks,
        validation_data=(X_val, y_val),
        verbose=verbosity
    )

if __name__ == "__main__":
    parser = argparse.ArgumentParser(
        description="train LSTM model for music generation")
    parser.add_argument('--id', metavar='id', type=str,
        help='model id to load weights if continuing training')
    parser.add_argument('--new', action='store_true', default=True,
        help='whether to load existing weights for model or \
        create new one')
    parser.add_argument('--tqdm', action='store_true', default=False,
        help='whether this is running in a Jupyter environment')

    parser.add_argument('--dataset', type=str,
        default="folk100k_melody",
        help='what dataset to use. Check "config.py" for
options')
    parser.add_argument('--layers', type=int, help="nr of layers")
    parser.add_argument('--bi', action="store_true", help="include
Bidirectionality wrapper for each layer")
    parser.add_argument('--att', action="store_true", help="add attention
mechanism on top of last layer")
    parser.add_argument('--cells', type=int, help="nr of cells in each layer")

    args = parser.parse_args()

    if not args.new and not args.id:
        print('either continue training a model by using "--id" or train a new
one by using "--new"')
        sys.exit(1)

    if args.id and args.new:
        print('either continue training a model by using "--id" or train a new
one by using "--new"')
        sys.exit(1)

    if args.id and not args.new:
        print('continue training of model %s' %args.id)

```

```

if args.new and not args.id:
    if not args.layers or not args.cells:
        print('need to specify nr of layers and nr of cells per layer')
        sys.exit(1)
    else:
        print('training a new model...')

main(args)

```

Melodies Generating Codes:

```

import tensorflow as tf
import argparse
import os
import sys
from glob import glob

import keras
import matplotlib.pyplot as plt
import numpy as np
import pypianoroll
from keras_self_attention import *
from keras_self_attention import SeqWeightedAttention
from tqdm import tqdm

import config
import keras.backend as K

from config import datasets
from generator import *
from utils import *

sys.path.append("/Users/arasharif/Desktop/Ara_Master_Thesis/")
sys.path.append("/Users/arasharif/opt/anaconda3/envs/magenta/gm/")
sys.path.append("/Users/arasharif/opt/anaconda3/envs/magenta/lib/python3.7/site-packages/note_seq")

import melodies_lib
import midi_io
import transpose

from melody_encoder_decoder import MelodyOneHotEncoding

def attention_loss(factor=1e-6):
    def attention_regularizer(y, y_pred):
        input_len = K.shape(y_pred)[-1]
        return factor * K.square(K.batch_dot(y_pred, K.permute_dimensions(y_pred,
(1, 0))))
        - tf.eye(input_len))
    return attention_regularizer

def att_model(cells, bi, layers, att):

    # cells = 64
    vocab_size=58
    # bi = True

```

```

# att=True
inputs = keras.layers.Input(
    shape=(63, 58,), name='Input')

prev = inputs
for i in range(layers):
    ret_seq = True
    if i == layers-1 and att == False:
        ret_seq = False

    this_layer = keras.layers.LSTM(
        cells,
        dropout=0.4,
        name='LSTM_%s' %i,
        return_sequences=ret_seq
    )
    if bi:
        this_layer = keras.layers.Bi-directional(
            this_layer,
            name='bi_%s' %i
        )
    prev = this_layer(prev)

attention = SeqWeightedAttention(
    return_attention=True,
    name='Attention'
)
attention_layer = attention(prev)
attention_layer, attention = attention_layer

dense = keras.layers.Dense(
    vocab_size, activation='softmax', name="dense_outputs")(attention_layer)

outputs = [dense, attention]
model = keras.Model(inputs=inputs, outputs=outputs)
model.compile(
    optimizer='adam',
    loss={
        'dense_outputs': 'categorical_crossentropy',
        'Attention': attention_loss(1e-4)
    }
)
return model

```

```

def get_model(args):
    model = None
    modelname = args.model_id

    if modelname == "model_folk_melody_2lstm32_attention":
        # (100000, 64, 58)
        model = att_model(32, False, 2, True)
    elif modelname == "model_folk_melody_bi3lstm64_attention":
        model = att_model(64, True, 3, True)
    else:
        json_model = open(os.path.join(modelname, "model.json"), "r").read()
        model = keras.models.model_from_json(
            json_model, custom_objects=SeqWeightedAttention.get_custom_objects())
    model.load_weights(os.path.join(modelname, "model.h5"))
    print(model.summary(line_length=100))
    return model

```

```

def get_dataset(args):
    dataset = None

    for dset in config.datasets.keys():
        if dset in args.model_id:
            dataset = config.datasets[dset]
            break
    dshape = dataset['shape']
    print('dataset : %s' % dataset)
    print(dshape)
    input_seq_len = dshape[1] - 1
    print(input_seq_len)
    ## for folk dataset (Ara)
    #min_note, max_note = np.load(
        #os.path.abspath(os.path.join(dataset['path'],
        '/Users/arasharif/Desktop/Ara_Master_Thesis/Folk_Dataset', 'min_max.npy')))
    ##for hook dataset (Ara)
    min_note, max_note = np.load(
        os.path.abspath(os.path.join(dataset['path'],
        '/Users/arasharif/Desktop/Ara_Master_Thesis/hooktheory_dataset', 'min_max.npy')))

    return input_seq_len, dshape, min_note, max_note

def transpose_seed(args):
    file = args.seed
    unique_name = ''.join(file.split(os.path.sep)[-1].split(".mid")[:-1])

    if not os.path.exists('midi_seeds_transposed'):
        os.mkdir('midi_seeds_transposed')

    transpose.main([file], os.path.abspath("midi_seeds_transposed/"))

    transposed_seed = os.path.abspath(
        glob(os.path.join("midi_seeds_transposed", unique_name) + "*")[0])
    print(transposed_seed)
    return transposed_seed

def from_trim_pianoroll_to_full(seq, min_note, max_note):
    zero_sequence = np.zeros((seq.shape[0], 128))
    zero_sequence[:, min_note:max_note + 1] = seq
    zero_sequence[zero_sequence == 1] = 127
    return zero_sequence

def save_trim_pianoroll_seq(seq, min_note, max_note, thepath):
    pypianoroll.Multitrack(
        tracks=[
            pypianoroll.Track(
                from_trim_pianoroll_to_full(
                    seq,
                    min_note,
                    max_note,
                ))
        ],
        beat_resolution=4).write(thepath)

def read_encode_pad_sequence_melody(filepath, min_note, max_note, input_seq_len):
    print("loading encoder...")
    encoder = MelodyOneHotEncoding(min_note, max_note+1)
    seed_melody = melodies_lib.midi_file_to_melody(filepath)
    seed_melody.squash(min_note, max_note)

```

```

seed_sequence = [encoder.encode_event(ev) for ev in list(seed_melody)]

print("padding...")
if len(seed_sequence) > input_seq_len:
    seed_sequence = np.array(seed_sequence[:input_seq_len])
else:
    zero_padded_seq = np.repeat(0, input_seq_len)
    zero_padded_seq[input_seq_len - len(seed_sequence):] = seed_sequence
    seed_sequence = zero_padded_seq
print("size after padding: ", seed_sequence.shape)

seed_sequence = keras.utils.to_categorical(seed_sequence,
num_classes=encoder.num_classes, dtype='uint8')
print('shape of seed sequence after 1h encoding: ', seed_sequence.shape)
return seed_sequence, encoder

def read_encode_pad_sequence_pianoroll(filepath, min_note, max_note,
input_seq_len):
    multitrack = pypianoroll.Multitrack(filepath, beat_resolution=4)
    sequence_full = multitrack.tracks[0]
    sequence_full.binarize()
    sequence_full = sequence_full.pianoroll
    seed_sequence = sequence_full[:, min_note:max_note + 1]
    print(seed_sequence.shape)

    print("padding...")
    if len(seed_sequence) > input_seq_len:
        seed_sequence = np.array(seed_sequence[:input_seq_len])
    else:
        zero_padded_seq = np.repeat(0, input_seq_len)
        zero_padded_seq[input_seq_len - len(seed_sequence):] = seed_sequence
        seed_sequence = zero_padded_seq
    print("size after padding: ", seed_sequence.shape)
    return seed_sequence

def build_template_for_generated_pianoroll(dshape, seed_sequence, model,
min_note, max_note, input_seq_len, seedfilename, model_dir):
    generated = np.zeros((2*dshape[1], seed_sequence.shape[1]))
    print('shape of generated ', generated.shape)
    generated[:input_seq_len] = seed_sequence
    seed_filename = seedfilename.split(os.path.sep)[-1].split(".mid")[0]

    if not os.path.exists(os.path.join(model_dir, "samples")):
        os.mkdir(os.path.join(model_dir, "samples"))

    samples_dir = os.path.abspath(os.path.join(model_dir, "samples",
seed_filename))
    if not os.path.exists(samples_dir):
        os.mkdir(samples_dir)

    seed_dir = os.path.join(samples_dir, "seed")
    if not os.path.exists(seed_dir):
        os.mkdir(seed_dir)

    seedpath = os.path.join(seed_dir, "1seed.mid")

    print("saving seed...")
    save_trim_pianoroll_seq(seed_sequence, min_note, max_note, seedpath)
    print('seed saved at ', seedpath)
    return generated, samples_dir

```

```

def save_trim_melody_seq(seed_sequence, encoder, seedpath):
    midi_io.note_sequence_to_midi_file(melodies_lib.Melody(
        [
            encoder.decode_event(ev) for ev in
np.trim_zeros(np.argmax(seed_sequence, axis=1), 'f')
        ]
    ).to_sequence(), seedpath)

def build_template_for_generated_melody(dshape, seed_sequence, model, min_note,
max_note, input_seq_len, seedfilename, model_dir, encoder):
    generated = np.zeros((2*dshape[1], seed_sequence.shape[1]))
    print('shape of generated ', generated.shape)
    generated[:input_seq_len] = seed_sequence
    seed_filename = seedfilename.split(os.path.sep)[-1].split(".mid")[0]

    if not os.path.exists(os.path.join(model_dir, "samples")):
        os.mkdir(os.path.join(model_dir, "samples"))

    samples_dir = os.path.abspath(os.path.join(model_dir, "samples",
seed_filename))
    if not os.path.exists(samples_dir):
        os.mkdir(samples_dir)

    seedpath = os.path.join(samples_dir, "1seed.mid")

    print("saving seed...")
    save_trim_melody_seq(seed_sequence, encoder, seedpath)
    print('seed saved at ', seedpath)
    return generated, samples_dir

def plot_midifile(filepath, samples_dir, name):
    roll = None
    try:
        roll =
pypianoroll.Multitrack(filepath, beat_resolution=4).tracks[0].pianoroll
    except Exception as _:
        return None
    plt.figure(figsize=(14, 8))
    ax = plt.gca()
    pypianoroll.plot_pianoroll(ax, roll)
    plt.title(name)
    pathtopng = os.path.join(samples_dir, name)
    print('plotting pianoroll to %s' % pathtopng)
    plt.savefig(pathtopng, bbox_inches='tight')
    return True

def generate_pianoroll(args, input_seq_ln, model, generated, samples_dir,
min_note, max_note,):
    temperature = float(args.temp)

    nr_samples = int(args.nr)

    for i in tqdm.tqdm(list(range(nr_samples))):
        for timestep in range(input_seq_ln, len(generated)):
            start_index = timestep - (input_seq_ln)
            sequence_for_prediction = generated[start_index:timestep]
            # next_step, att = sample(model, sequence_for_prediction,
temperature, withatt=True)
            next_step, _ = sample(model, sequence_for_prediction, temperature,

```

```

withatt=args.att)
#         print(att.argsort() [-10:] [::-1])
        generated[timestep] = next_step

        generated_noseed = generated[input_seq_len:]

        new_path = os.path.join(samples_dir, "temp_%s_%s.mid" %(temperature, i))
        save_trim_pianoroll_seq(generated_noseed,min_note,max_note,new_path)
        plot_midifile(new_path,samples_dir,"temp_%s_%s.png" %(temperature, i))

def pianoroll_sampling(filepath, min_note, max_note, model,
                        input_seq_len, dshape, model_dir):
    print('shape of sequence from pypianoroll...')
    seed_sequence = read_encode_pad_sequence_pianoroll(filepath, min_note,
max_note, input_seq_len)
    generated, samples_dir = build_template_for_generated_pianoroll(dshape,
seed_sequence, model, min_note, max_note, input_seq_len, filepath, model_dir)

    # plot seed and save in folder
    plot_midifile(filepath, samples_dir, '1seed.png')
    generate_pianoroll(args, input_seq_len, model, generated, samples_dir,
min_note, max_note)

def generate_melody(args, input_seq_len, model, generated, samples_dir, min_note,
max_note, encoder):
    temperature = float(args.temp)
    to_generate = int(args.nr)
    nr_empty = 0
    nr_generated = 0

    progress = tqdm.tqdm(total=to_generate)
    atts = []
    softmax_es = []
    tokens_low = []
    tokens_high = []
    while nr_generated != to_generate:
        for timestep in range(input_seq_len, len(generated)):
            start_index = timestep - (input_seq_len)
            sequence_for_prediction = generated[start_index:timestep]
            #         next_step, att = sample(model, sequence_for_prediction,
temperature, withatt=True)
            next_step = None
            if args.att:
                next_step, att, softmax_preds = sample(model,
sequence_for_prediction, temperature, withatt=args.att)
                if args.no_zero:
                    input_tokens = np.argmax(sequence_for_prediction,axis=1)
                    mask = np.where(input_tokens==0)
                    att[mask] = 0
                if np.argmax(att) < 6:
                    # print('focusing on token',
np.argmax(sequence_for_prediction[np.argmax(att)]), 'at time step index',
np.argmax(att))
                    tokens_low.append(
                        np.argmax(sequence_for_prediction[np.argmax(att)])
                    )
                if args.debug_print:
                    print(
                        'window around focused token ',
                        np.argmax(
                            sequence_for_prediction[0:np.argmax(att)+3],

```

```

        axis=1)
    )
    print('softmax pointing at ', np.argmax(softmax_preds), '
actual prediction is ', np.argmax(next_step))
    elif np.argmax(atts) > 30:
        tokens_high.append(
            np.argmax(sequence_for_prediction[np.argmax(atts)])
        )
    atts.append(atts)
    else:
        next_step, softmax_preds = sample(model, sequence_for_prediction,
temperature, withatt=args.atts)
        softmax_es.append(softmax_preds)
        # print(atts.argsort() [-10:] [::-1])
        generated[timestep] = next_step

generated_noseed = generated[input_seq_len:]
unique_pitches = np.unique(np.argmax(generated_noseed,axis=1))
if len(unique_pitches) == 1 and unique_pitches[0] == 0:
    nr_empty += 1
else:
    new_path = os.path.join(samples_dir, "temp_%s_%s.mid" %(temperature,
nr_generated))

    save_trim_melody_seq(generated_noseed, encoder, new_path)
    if not plot_midifile(new_path,samples_dir,"temp_%s_%s.png"
%(temperature, nr_generated)):
        nr_empty += 1
    else:
        nr_generated += 1
        progress.update(1)

print('generated %s empty rolls' %nr_empty)
if args.atts:
    atts = np.array(atts)
    atts = atts.reshape(to_generate, -1, atts.shape[-1])
    np.save(os.path.join(samples_dir, 'atts.npy'), atts)

softmax_es = np.array(softmax_es)
softmax_es = softmax_es.reshape(to_generate, -1, softmax_es.shape[-1])
np.save(os.path.join(samples_dir, 'softmax.npy'), softmax_es)
with open(os.path.join(samples_dir, '%s empty.txt' %nr_empty), 'w') as f:
    f.writelines('\n')

def melody_sampling(filepath, min_note, max_note, model,
                    input_seq_len, dshape, model_dir):
    print('shape of sequence from pypianoroll...')
    seed_sequence, encoder = read_encode_pad_sequence_melody(filepath, min_note,
max_note, input_seq_len)
    generated, samples_dir = build_template_for_generated_melody(dshape,
seed_sequence, model, min_note, max_note, input_seq_len, filepath, model_dir,
encoder)

    # plot seed and save in folder
    plot_midifile(filepath, samples_dir, '1seed.png')
    generate_melody(args, input_seq_len, model, generated, samples_dir, min_note,
max_note, encoder)

def main(args):
    input_seq_len, dshape, min_note, max_note = get_dataset(args)

```



```

model = get_model(args)
transposed_seed = transpose_seed(args)
print("min, max:")
print(min_note, max_note)
model_dir = os.path.abspath(args.model_id)

if "pianoroll" in args.model_id:
    # pianoroll encoding
    pianoroll_sampling(transposed_seed, min_note, max_note, model,
                       input_seq_len, dshape, model_dir)

elif "melody" in args.model_id or args.melody:
    # melody encoding
    # melody_sampling(transposed_seed, min_note, max_note, model,
    #                  input_seq_len, dshape)
    melody_sampling(transposed_seed, min_note, max_note, model,
                    input_seq_len, dshape, model_dir)

else:
    print("unknown encoding in model name : %s" % args.model_id)
return

if __name__ == "__main__":
    parser = argparse.ArgumentParser(
        description="sample from model using a specific seed")
    parser.add_argument('model_id', metavar='id', type=str, help='model id')
    parser.add_argument(
        'seed', metavar='seed', type=str, help='path to seed midi file')
    parser.add_argument(
        '--nr',
        type=str,
        default="10",
        help='how many samples to generate. default = 10')
    parser.add_argument(
        '--temp',
        type=float,
        default="1.0",
        help='temperature for sampling. default = 1.0')
    parser.add_argument(
        '--att',
        action='store_true'
    )
    parser.add_argument(
        '--melody',
        action='store_true',
    )
    parser.add_argument(
        '--debug_print',
        action='store_true',
        help='whether to print info about attention tokens'
    )
    parser.add_argument(
        '--no_zero',
        action='store_true',
        help='in plotting attention remove all zeros'
    )

    args = parser.parse_args()

    print('generating %s samples, at %s temperature, using %s, from seed %s'
          %(args.nr, args.temp, args.model_id, args.seed))
    main(args)

```

Participants Survey codes:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import ttest_ind

plt.style.use('ggplot')

# path = "/Users/arasharif/Desktop/Ara_Master_Thesis/survey
# results/Thesis_Monophonic_Music_Survey.csv"
path = "/Users/arasharif/Desktop/Ara_Master_Thesis/survey
results/Thesis_Monophonic_Music_Survey.csv"
data = np.loadtxt(path, skiprows=1, delimiter=',', dtype=object)
data = data[:, 1:] # eliminate names column

user_experiences, u_x_counts = np.unique(data[:-2, 0], return_counts=True)
total = np.sum(u_x_counts)

plt.figure(figsize=(5, 5))
plt.pie(
    u_x_counts,
    labels=user_experiences,
    shadow=False,
    autopct=lambda p: '{:.0f}'.format(p * total / 100)
)
plt.title('Participant distribution by musical experience level')
plt.savefig('* Participant_distributions.png', bbox_inches='tight')
# plt.show()
users = {
    1: {
        'train': {
            'r': [], 'm': []
        },
        'gen': {
            'r': [], 'm': []
        }
    } \
    for l in user_experiences if len(l.strip()) > 0
}
print(users)

train = {'r': [], 'm': []}

gen = {'r': [], 'm': []}

for row in data[:-2]:
    u_x = row[0]
    scores = row[1:]
    for i in range(len(scores)):
        score = int(scores[i])
        if i % 2 == 0:
            label = data[-2, i + 2]
        else:
            label = data[-2, i + 1]
        if label == '1':
            # generated sample
            if i % 2 == 0:
                # rhythm
                gen['r'].append(score)
            users[u_x]['gen']['r'].append(score)
        else:
```

```

        # melody
        gen['m'].append(score)
        users[u_x]['gen']['m'].append(score)
    elif label == '0':
        # training sample
        if i % 2 == 0:
            # rhythm
            train['r'].append(score)
            users[u_x]['train']['r'].append(score)
        else:
            # melody
            train['m'].append(score)
            users[u_x]['train']['m'].append(score)

def plot_overall(train, gen):
    g_r_mean = np.mean(gen['r'])
    g_r_std = np.std(gen['r'])

    g_m_mean = np.mean(gen['m'])
    g_m_std = np.std(gen['m'])

    t_r_mean = np.mean(train['r'])
    t_r_std = np.std(train['r'])

    t_m_mean = np.mean(train['m'])
    t_m_std = np.std(train['m'])

    names = ['original rhythm', 'generated rhythm', 'original melody', 'generated
melody']
    x_pos = np.arange(len(names))
    means = [
        t_r_mean,
        g_r_mean,
        t_m_mean,
        g_m_mean
    ]
    stds = [
        t_r_std,
        g_r_std,
        t_m_std,
        g_m_std
    ]

    plt.figure(figsize=(5, 5))
    plt.bar(
        x_pos,
        means,
        yerr=stds,
        align='center',
        alpha=0.6,
        ecolor='black',
        color=np.concatenate([[ 'green' ] * 2, [ 'orange' ] * 2])
    )
    # plt.xticks(x_pos, names)
    plt.xticks(x_pos, names, fontsize=8, fontweight='bold', rotation=90)
    plt.title('Participant evaluation results')
    plt.savefig('* participant_evaluation_results.png', bbox_inches='tight')

    table = np.vstack([
        ['%.3f' % m for m in means],
        ['%.3f' % s for s in stds]
    ])
    table = pd.DataFrame(table, columns=names).transpose()

```

```

print(table.to_latex())

# t tests
# rhythm
r_ttest = ttest_ind(gen['r'], train['r'])
print('rhythm t-test p: %.4f' % r_ttest[1], r_ttest[1])

# melody
m_ttest = ttest_ind(gen['m'], train['m'])
print('melody t-test p: %.4f' % m_ttest[1], m_ttest[1])

def plot_separate(users):
    group_1_train_r = np.concatenate([
        users['1']['train']['r']
    ])
    group_1_train_m = np.concatenate([
        users['1']['train']['m']
    ])
    group_1_gen_r = np.concatenate([
        users['1']['gen']['r']
    ])
    group_1_gen_m = np.concatenate([
        users['1']['gen']['m']
    ])

    group_2_train_r = np.concatenate([
        users['2']['train']['r']
    ])
    group_2_train_m = np.concatenate([
        users['2']['train']['m']
    ])
    group_2_gen_r = np.concatenate([
        users['2']['gen']['r']
    ])
    group_2_gen_m = np.concatenate([
        users['2']['gen']['m']
    ])

    group_3_train_r = np.concatenate([
        users['3']['train']['r']
    ])
    group_3_train_m = np.concatenate([
        users['3']['train']['m']
    ])
    group_3_gen_r = np.concatenate([
        users['3']['gen']['r']
    ])
    group_3_gen_m = np.concatenate([
        users['3']['gen']['m']
    ])

    group_4_train_r = np.concatenate([
        users['4']['train']['r']
    ])
    group_4_train_m = np.concatenate([
        users['4']['train']['m']
    ])

```

```

group_4_gen_r = np.concatenate([
    users['4']['gen']['r']
])
group_4_gen_m = np.concatenate([
    users['4']['gen']['m']
])

group_5_train_r = np.concatenate([
    users['5']['train']['r']
])

group_5_train_m = np.concatenate([
    users['5']['train']['m']
])

group_5_gen_r = np.concatenate([
    users['5']['gen']['r']
])
group_5_gen_m = np.concatenate([
    users['5']['gen']['m']
])
### abcd
group_6_train_r = np.concatenate([
    users['1']['train']['r'], users['2']['train']['r']
])
group_6_train_m = np.concatenate([
    users['1']['train']['m'], users['2']['train']['m']
])

group_6_gen_r = np.concatenate([
    users['1']['gen']['r'], users['2']['gen']['r']
])
group_6_gen_m = np.concatenate([
    users['1']['gen']['m'], users['2']['gen']['m']
])

group_7_train_r = np.concatenate([
    users['3']['train']['r'],
    users['4']['train']['r'],
    users['5']['train']['r']
])
group_7_train_m = np.concatenate([
    users['3']['train']['m'],
    users['4']['train']['m'],
    users['5']['train']['m']
])

group_7_gen_r = np.concatenate([
    users['3']['gen']['r'],
    users['4']['gen']['r'],
    users['5']['gen']['r']
])
group_7_gen_m = np.concatenate([
    users['3']['gen']['m'],
    users['4']['gen']['m'],
    users['5']['gen']['m']
])

names = [
    'level1 original rhythm',
    'level1 generated rhythm',
    'level1 original melody',
    'level1 generated melody',

```

```

'level2 original rhythm',
'level2 generated rhythm',
'level2 original melody',
'level2 generated melody',
'level3 original rhythm',
'level3 generated rhythm',
'level3 original melody',
'level3 generated melody',
'level4 original rhythm',
'level4 generated rhythm',
'level4 original melody',
'level4 generated melody',
'level5 original rhythm',
'level5 generated rhythm',
'level5 original melody',
'level5 generated melody',
]

x_pos = np.arange(len(names))
groups = [
    group_1_train_r,
    group_1_gen_r,
    group_1_train_m,
    group_1_gen_m,
    group_2_train_r,
    group_2_gen_r,
    group_2_train_m,
    group_2_gen_m,
    group_3_train_r,
    group_3_gen_r,
    group_3_train_m,
    group_3_gen_m,
    group_4_train_r,
    group_4_gen_r,
    group_4_train_m,
    group_4_gen_m,
    group_5_train_r,
    group_5_gen_r,
    group_5_train_m,
    group_5_gen_m,
]

means = [
    np.mean(g) for g in groups
]
stds = [
    np.std(g) for g in groups
]

table = np.vstack([
    ['%.3f' % m for m in means],
    ['%.3f' % s for s in stds]
])
table = pd.DataFrame(table, columns=names).transpose()
print(table.to_latex())

plt.figure(figsize=(7, 5))
plt.bar(x_pos, means, yerr=stds, align='center', alpha=0.5, ecolor='black',
        color=np.concatenate([['red'] * 2, ['red'] * 2, ['orange'] * 2,
                                ['orange'] * 2, ['green'] * 2,
                                ['blue'] * 2, ['blue'] * 2, ['black'] * 2,
                                ['black'] * 2]))
plt.xticks(x_pos, names, fontsize=8, fontweight='bold', rotation=90)
plt.title('Participant evaluation results per experience levels')

```

```

plt.savefig('* Participant_evaluation_results2.png', bbox_inches='tight')

names = [
    'group1 original rhythm',
    'group1 generated rhythm',
    'group1 original melody',
    'group1 generated melody',
    'group2 original rhythm',
    'group2 generated rhythm',
    'group2 original melody',
    'group2 generated melody',
]

x_pos = np.arange(len(names))
groups = [
    group_6_train_r,
    group_6_gen_r,
    group_6_train_m,
    group_6_gen_m,
    group_7_train_r,
    group_7_gen_r,
    group_7_train_m,
    group_7_gen_m,
]

means = [
    np.mean(g) for g in groups
]
stds = [
    np.std(g) for g in groups
]

plt.figure(figsize=(7, 5))
plt.bar(x_pos, means, yerr=stds, align='center', alpha=0.5, ecolor='black',
        color=np.concatenate([[ 'red' ]*2, [ 'orange' ]*2, [ 'green' ]*2,
[ 'blue' ]*2]))
plt.xticks(x_pos, names, fontsize=8, fontweight='bold', rotation=90)
plt.title('Participant evaluation results per group level (low & high)')
plt.savefig('* Participant_evaluation_results3.png', bbox_inches='tight')

table = np.vstack([
    ['%.3f' % m for m in means],
    ['%.3f' % s for s in stds]
])
table = pd.DataFrame(table, columns=names).transpose()
print(table.to_latex())

plot_overall(train, gen)

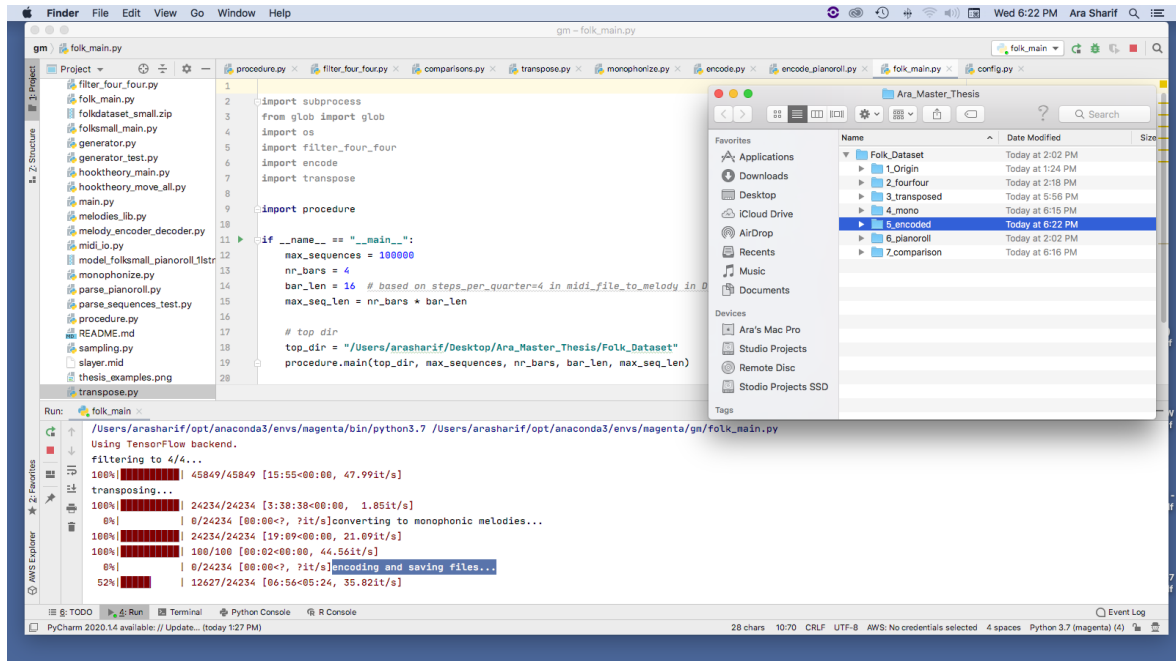
plot_separate(users)
plt.show()

```

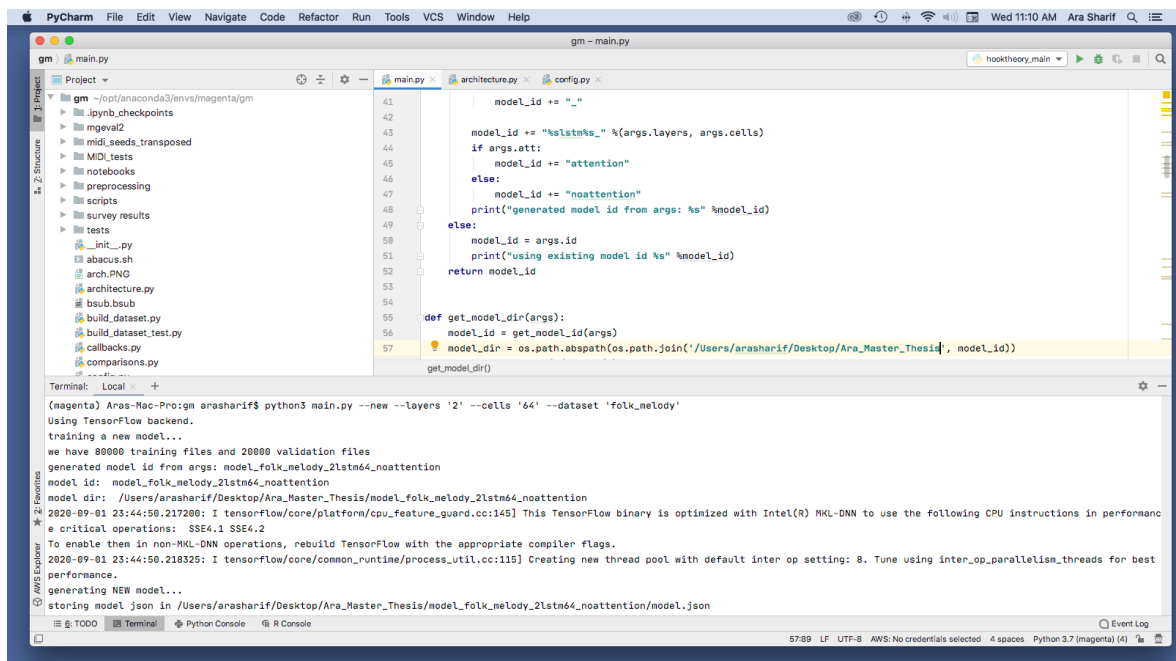
APPENDIX 4

Screenshot Samples for Implementations

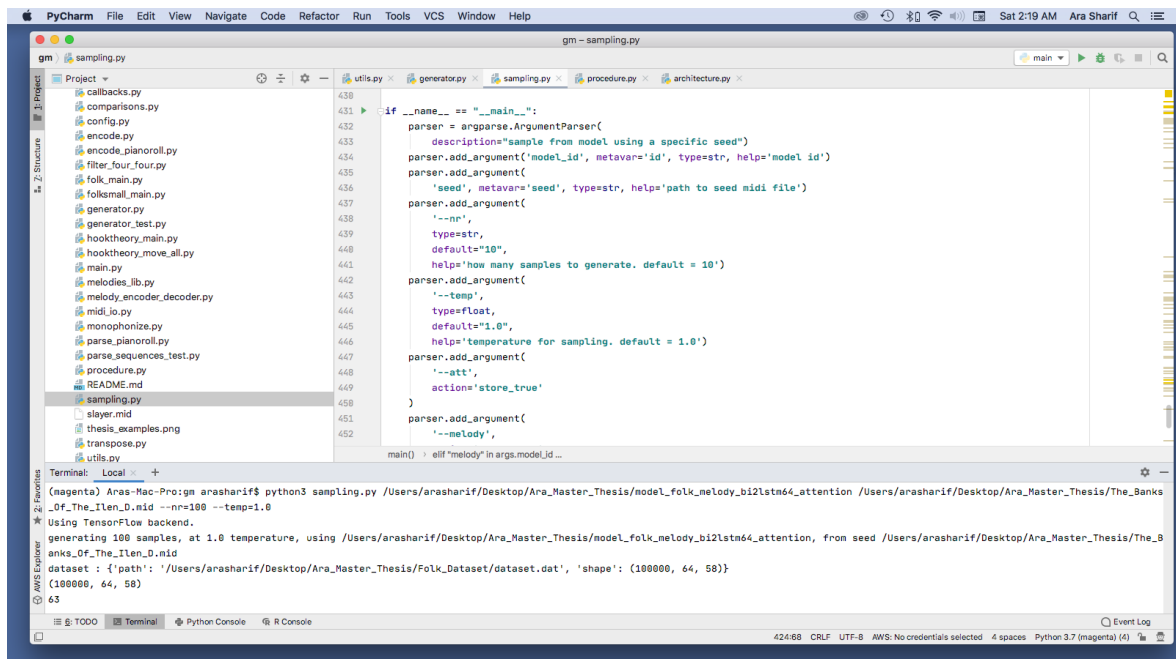
Preprocessing Screenshot Sample:



Training Screenshot Sample:



Generating Screenshot Sample:

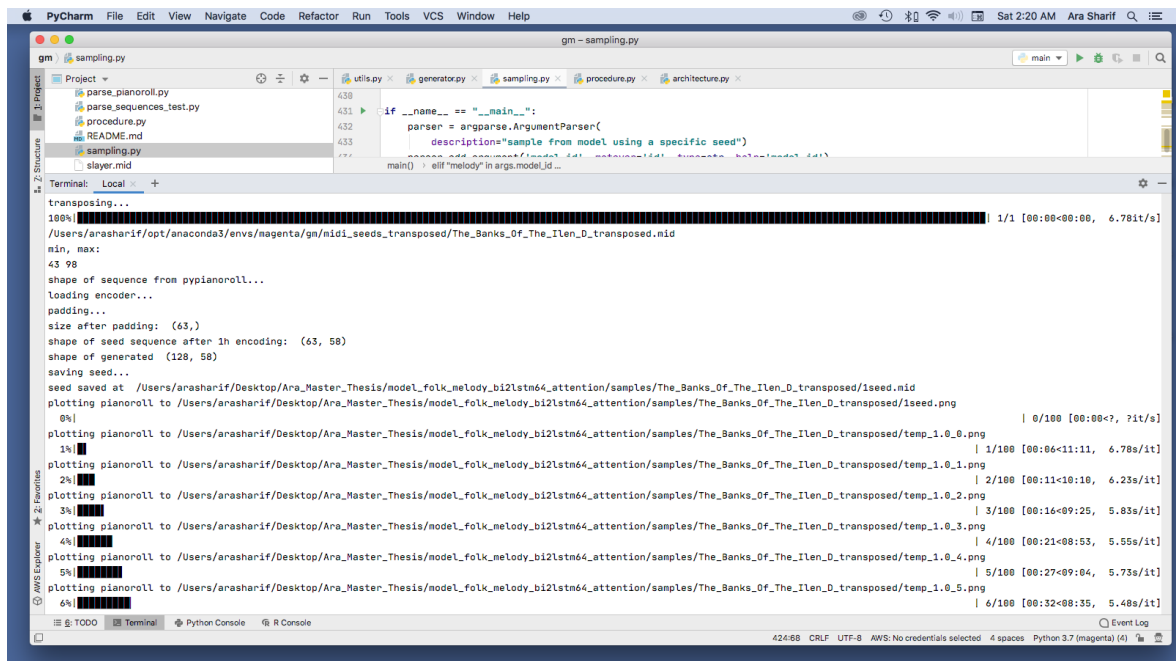


The screenshot shows the PyCharm IDE with the `sampling.py` file open. The code defines an argument parser for a sampling process. The terminal output shows the execution of the script with the following parameters: `--nr=100 --temp=1.0`. The output indicates that 100 samples are generated at 1.0 temperature, using a specific seed and dataset.

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser(
        description="sample from model using a specific seed"
    )
    parser.add_argument('model_id', metavar='id', type=str, help='model id')
    parser.add_argument(
        'seed', metavar='seed', type=str, help='path to seed midi file'
    )
    parser.add_argument(
        '--nr',
        type=str,
        default="10",
        help='how many samples to generate. default = 10'
    )
    parser.add_argument(
        '--temp',
        type=float,
        default=1.0,
        help='temperature for sampling. default = 1.0'
    )
    parser.add_argument(
        '--att',
        action='store_true'
    )
    parser.add_argument(
        '--melody',
    )

    main()
    elif "melody" in args.model_id...
```

```
Terminal: Local
(magenta) Ara-Mac-Pro:gm arasharif$ python3 sampling.py /Users/arasharif/Desktop/Ara_Master_Thesis/model_folk_melody_b12lstm64_attention /Users/arasharif/Desktop/Ara_Master_Thesis/The_Banks_Of_The_Ilen_D.mid --nr=100 --temp=1.0
Using TensorFlow backend.
generating 100 samples, at 1.0 temperature, using /Users/arasharif/Desktop/Ara_Master_Thesis/model_folk_melody_b12lstm64_attention, from seed /Users/arasharif/Desktop/Ara_Master_Thesis/The_Banks_Of_The_Ilen_D.mid
dataset : {'path': '/Users/arasharif/Desktop/Ara_Master_Thesis/Folk_Dataset/dataset.dat', 'shape': (100000, 64, 58)}
(100000, 64, 58)
63
```



The screenshot shows the PyCharm IDE with the `sampling.py` file open. The code defines an argument parser for a sampling process. The terminal output shows the execution of the script with the following parameters: `--nr=100 --temp=1.0`. The output indicates that 100 samples are generated at 1.0 temperature, using a specific seed and dataset.

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser(
        description="sample from model using a specific seed"
    )
    parser.add_argument('model_id', metavar='id', type=str, help='model id')
    parser.add_argument(
        'seed', metavar='seed', type=str, help='path to seed midi file'
    )
    parser.add_argument(
        '--nr',
        type=str,
        default="10",
        help='how many samples to generate. default = 10'
    )
    parser.add_argument(
        '--temp',
        type=float,
        default=1.0,
        help='temperature for sampling. default = 1.0'
    )
    parser.add_argument(
        '--att',
        action='store_true'
    )
    parser.add_argument(
        '--melody',
    )

    main()
    elif "melody" in args.model_id...
```

```
Terminal: Local
(magenta) Ara-Mac-Pro:gm arasharif$ python3 sampling.py /Users/arasharif/Desktop/Ara_Master_Thesis/model_folk_melody_b12lstm64_attention /Users/arasharif/Desktop/Ara_Master_Thesis/The_Banks_Of_The_Ilen_D.mid --nr=100 --temp=1.0
Using TensorFlow backend.
generating 100 samples, at 1.0 temperature, using /Users/arasharif/Desktop/Ara_Master_Thesis/model_folk_melody_b12lstm64_attention, from seed /Users/arasharif/Desktop/Ara_Master_Thesis/The_Banks_Of_The_Ilen_D.mid
dataset : {'path': '/Users/arasharif/Desktop/Ara_Master_Thesis/Folk_Dataset/dataset.dat', 'shape': (100000, 64, 58)}
(100000, 64, 58)
63
```

APPENDIX 5

Survey Resources

A Sample of human evaluation form:



NEAR EAST UNIVERSITY

Graduated School of Applied Science

Computer Engineering

MONOPHONIC MUSIC GENERATION USING ARTIFICIAL INTELLIGENCE THROUGH DEEP LEARNING TECHNIQUES

Human evaluation for monophonic melodies

prepared by: Ara Ahmed Sharif

October 2020

Participant Name: <u>Zaher Shwan</u>	Musical experience level	5
--------------------------------------	--------------------------	---

	Read the scores & play them with your instrument	Rate for rhythm stability 1-5	Rate for melody pleasing 1-5	Listen to Melody Samples & tap with them	Rate for rhythm stability 1-5	Rate for melody pleasing 1-5
1	<u>Monophonic Samples-Score</u>	4	5	<u>Monophonic Sample1</u>	4	5
2	<u>Monophonic Samples-Score</u>	4	4	<u>Monophonic Sample2</u>	3	4
3	<u>Monophonic Samples-Score</u>	4	5	<u>Monophonic Sample3</u>	4	4
4	<u>Monophonic Samples-Score</u>	4	5	<u>Monophonic Sample4</u>	5	5
5	<u>Monophonic Samples-Score</u>	5	5	<u>Monophonic Sample5</u>	3	5
6	<u>Monophonic Samples-Score</u>	4	3	<u>Monophonic Sample6</u>	4	2
7	<u>Monophonic Samples-Score</u>	4	3	<u>Monophonic Sample7</u>	3	2
8	<u>Monophonic Samples-Score</u>	3	3	<u>Monophonic Sample8</u>	4	5
9	<u>Monophonic Samples-Score</u>	2	2	<u>Monophonic Sample9</u>	3	3
10	<u>Monophonic Samples-Score</u>	4	4	<u>Monophonic Sample10</u>	3	4

Survey Result:

Participant Name	Performance Metrics (1-5 Scale)																				
	Rate for Music Experience 1-5	Rate for rhythm stability 1-5	Rate for melody pleasing 1-5	Rate for rhythm stability 1-5	Rate for melody pleasing 1-5	Rate for rhythm stability 1-5	Rate for melody pleasing 1-5	Rate for rhythm stability 1-5	Rate for melody pleasing 1-5	Rate for rhythm stability 1-5	Rate for melody pleasing 1-5	Rate for rhythm stability 1-5	Rate for melody pleasing 1-5	Rate for rhythm stability 1-5	Rate for melody pleasing 1-5	Rate for rhythm stability 1-5	Rate for melody pleasing 1-5	Rate for rhythm stability 1-5	Rate for melody pleasing 1-5	Rate for rhythm stability 1-5	
Zhila yahst	1	3	4	2	3	5	4	2	2	4	3	4	2	2	4	3	3	2	5	3	2
Shene Sirwan	1	4	2	1	1	4	4	3	2	5	5	1	1	2	2	1	1	4	4	2	1
Ani Kamaran	1	3	4	4	4	3	4	5	4	3	2	2	3	2	2	1	2	3	2	4	3
Frishta Ahmed	1	3	5	4	5	4	4	5	5	4	5	4	4	4	4	5	4	5	3	5	5
Rand Dilman	1	3	4	4	3	3	4	4	4	2	3	4	3	5	2	4	4	3	3	4	4
Kamal Aerin	1	4	3	2	2	2	3	2	4	3	2	4	4	3	2	3	5	3	4	4	3
Barin Jamal	1	4	4	4	4	5	5	3	4	5	5	3	3	3	3	4	5	5	4	3	3
Yad Hidayat	1	5	4	5	4	5	4	5	4	5	3	5	3	4	3	4	4	4	4	5	4
Lanya Maeiwan	1	4	4	5	4	5	5	3	3	5	4	5	4	5	5	5	3	4	3	4	4
Rasan Najib	1	4	4	3	4	5	5	4	3	5	4	2	2	5	5	5	5	4	4	3	3
Aya Aram	2	5	5	5	3	5	5	5	4	5	5	5	4	5	5	5	4	5	5	5	4
Roza Walid	2	3	3	5	5	5	5	3	3	5	5	2	2	5	5	3	4	3	2	5	4
Aland Muhamad	2	4	5	5	5	5	5	4	4	5	5	3	4	3	4	3	4	4	4	3	4
Miran Unis	2	5	5	5	2	5	5	5	4	5	4	4	4	5	5	5	5	5	5	4	3
Naz Mahmud	2	4	5	5	5	5	5	5	5	5	5	4	5	5	5	5	5	5	5	5	5
Mhemed Baqi	2	3	3	3	2	3	2	4	4	4	4	3	3	3	3	2	3	2	3	3	3
Lawin Jamal	2	3	3	3	4	5	2	5	1	5	3	3	4	5	5	4	4	5	5	5	3
Namo Jamal	2	3	5	5	4	5	5	4	4	3	4	4	3	4	5	4	3	4	5	3	4
Sonya Ahmad	2	3	3	4	3	2	5	3	5	4	3	5	5	4	4	5	5	5	4	3	2
Baraham Saman	2	3	3	2	4	2	3	4	3	3	5	4	3	5	4	2	2	3	5	3	4
Shabar Qadir	3	3	4	4	4	3	3	5	5	5	5	4	4	3	3	2	2	2	3	5	5
Monako Ibrahim	3	5	3	5	4	3	2	2	4	2	2	4	2	1	4	1	2	4	2	3	5
Safin Mahmud	3	5	5	5	5	5	5	5	5	5	5	5	5	5	4	5	5	5	5	5	5
Avan Sabri	3	4	5	3	3	3	4	5	5	5	5	3	5	2	3	4	4	3	3	5	5
Tara Ahmad	3	4	3	4	4	4	5	4	2	3	3	4	3	3	5	3	2	3	4	3	3
Helin Majid	3	4	5	4	5	4	3	3	2	5	3	4	4	4	3	3	3	3	3	4	2
Vanya Ara	3	4	5	5	5	3	4	2	2	4	4	3	5	2	4	1	2	3	4	4	5
Hasaw Amir	3	5	4	5	5	5	4	5	5	4	5	4	4	4	4	4	5	4	4	5	4
Harez Hersh	3	4	3	3	5	3	3	4	2	5	4	4	4	3	4	4	5	5	3	5	3
Danar Ayar	3	5	3	4	5	5	5	4	4	4	3	3	4	5	4	3	4	4	5	4	3
Zhulia Shwan	4	5	5	5	4	3	4	4	5	5	5	4	5	3	5	4	4	5	5	4	4
Bawan Akram	4	5	5	5	4	5	4	5	5	5	4	5	5	4	4	4	5	5	5	4	5
Sivan Jamal	4	4	3	2	3	4	5	3	5	2	4	3	3	4	3	5	4	3	5	4	3
Las Azad	4	4	4	3	3	5	4	3	2	3	4	5	4	3	3	3	4	4	3	4	3
Zhulia Hussen	4	4	3	5	4	5	4	5	5	5	4	5	3	5	5	5	5	5	5	5	4
Diya Diyar	4	4	4	3	2	4	2	5	3	5	5	2	2	4	3	5	5	5	5	3	3
Chapk Amin	4	4	3	5	4	2	1	3	2	1	1	2	2	3	3	5	5	2	1	5	5
Ara Kamaran	4	4	4	3	4	5	4	5	5	4	5	3	2	4	4	5	5	4	5	3	4
Vesan Ayar	4	3	4	4	5	3	4	4	3	5	4	4	3	5	4	4	4	3	4	3	3
Akar Ari	4	4	5	4	3	3	5	3	3	4	3	4	2	5	3	4	4	3	4	5	4
Bilind Zahid	5	4	4	4	3	4	4	3	3	2	2	2	3	5	4	3	4	4	4	4	3
Hemin Husen	5	2	2	2	3	2	2	1	1	2	3	1	1	3	2	1	1	1	1	2	1
Ako Aziz	5	5	4	5	5	5	4	5	5	5	5	4	5	4	5	5	5	4	4	4	4
Baxan Aso	5	4	5	5	3	3	4	4	3	4	4	4	3	5	4	3	2	2	4	3	3
Zulya Shwan	5	4	5	3	4	4	4	5	5	3	5	4	2	3	2	4	5	3	3	3	4
Helin Ara	5	5	4	4	3	4	5	4	4	5	3	3	2	2	3	4	4	3	3	3	4
Diler Husen	5	4	4	4	4	4	5	4	3	3	4	3	4	4	3	4	5	4	5	4	4
Twana Faraj	5	4	4	4	3	3	4	5	5	4	3	5	4	3	4	4	3	2	3	4	4
Diya Ayar	5	4	4	3	4	4	4	3	3	4	3	2	4	4	5	4	3	4	4	4	4
Diyarı Muhamad	5	4	4	4	5	4	3	4	5	5	4	5	3	4	5	4	4	3	4	4	3
Origin_Generate		0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
Sample No.		1	1	2	2	3	3	4	4	5	5	6	6	7	7	8	8	9	9	10	10