

MUSIC GENERATION FROM NOISE USING MACHINE LEARNING

*Major project report submitted
in partial fulfillment of the requirement for award of the degree of*

**Bachelor of Technology
in
Computer Science & Engineering**

By

Tushar Kumar	(17UECN0064)
Amit Kumar Yadav	(17UECS0048)
B.Shiv Kumar	(17UECS0084)

*Under the guidance of
Ms.S.Abhinaya,M.Tech.,
ASSISTANT PROFESSOR*



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SCHOOL OF COMPUTING
VEL TECH RANGARAJAN Dr.SAGUNTHALA R&D
INSTITUTE OF SCIENCE AND TECHNOLOGY
(Deemed to be University Estd u/s 3 of UGC Act, 1956)**

**Accredited by NAAC with A Grade
CHENNAI 600 062, TAMILNADU, INDIA**

June,2021

CERTIFICATE

It is certified that the work contained in the project report titled "MUSIC GENERATION FROM NOISE USING MACHINE LEARNING" by "Tushar Kumar (17UECN0064), Amit Kumar Yadav (17UECS0048), B.Shiv Kumar (17UECS0084)" has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Signature of Supervisor

Ms. S. Abhinaya

Assistant Professor

Computer Science & Engineering

School of Computing

Vel Tech Rangarajan Dr.Sagunthala R&D

Institute of Science and Technology

June,2021

Signature of Head of the Department

Dr. V. Srinivasa Rao

Professor & Head

Computer Science & Engineering

School of Computing

Vel Tech Rangarajan Dr.Sagunthala R&D

Institute of Science and Technology

June,2021

DECLARATION

We declare that this written submission represents my ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

(Signature)

TUSHAR KUMAR

Date: / /

(Signature)

AMIT KUMAR YADAV

Date: / /

(Signature)

B. SHIV KUMAR

Date: / /

APPROVAL SHEET

This project report entitled “MUSIC GENERATION FROM NOISE USING MACHINE LEARNING” by ”Tushar Kumar (17UECN0064), Amit Kumar Yadav (17UECS0048), B.Shiv Kumar (17UECS0084)” is approved for the degree of B.Tech in Computer Science & Engineering.

Examiners

Supervisor

Ms. S. Abhinaya, M.Tech.,

Date: / /

Place:

ACKNOWLEDGEMENT

We express our deepest gratitude to our respected **Founder Chancellor and President Col. Prof. Dr. R. RANGARAJAN B.E. (EEE), B.E. (MECH), M.S (AUTO). DSc., Foundress President Dr. R. SAGUNTHALA RANGARAJAN M.B.B.S.** Chairperson Managing Trustee and Vice President.

We are very much grateful to our beloved **Vice Chancellor Prof. S. SALIVAHANAN**, for providing us with an environment to complete our project successfully.

We record indebtedness to our **Dean & Head, Department of Computer Science and Engineering Dr. V. SRINIVASA RAO, M.Tech., Ph.D.**, for immense care and encouragement towards us throughout the course of this project.

We also take this opportunity to express a deep sense of gratitude to Our Internal Supervisor **Ms. S. ABHINAYA, M.Tech.**, for her cordial support, valuable information and guidance, She helped us in completing this project through various stages.

A special thanks to our **Project Coordinators Mr. V. ASHOK KUMAR, M.Tech., Ms. C. SHYAMALA KUMARI, M.E., Ms.S.FLORENCE, M.Tech.**, for their valuable guidance and support throughout the course of the project.

We thank our department faculty, supporting staff and friends for their help and guidance to complete this project.

Tushar Kumar (17UECN0064)

Amit Kumar Yadav (17UECS0048)

B. Shiv Kumar (17UECS0084)

ABSTRACT

In this project we are going to implement a machine learning model which maps random sounds to musical notes which is further integrated in an harmonious ways according to the trained data set in the pipelining model. This ML model make uses of Pachyderm to handle the full pipeline for scaling and management. pachyderm makes it supremely simple to string together a bunch of loosely coupled frameworks into a smoothly scaling AI training platform. If you can package up your program in a Docker container you can easily run it in Pachyderm. This model takes a random noise note as input in mp3 format, which is further transformed to different formats for the machine learning model to recognize. This pipeline model is scaled according to the size of input data fed. This project will be consisting of a tool-box formatted container with the transformer model as the backend, where the user can feed the data and covert the noise notes into musical notes and gets an ambient music clip as output. Coming to the data set, we will be training the above-mentioned pipelining model with various noise notes which have been extracted from the data set. In here, each module will be commuting with other using JSON files. We will be integrating this whole file in a Docker container for an efficient implementation and use for the user.

Keywords: Pipelining, Machine learning, Transformer model .

LIST OF FIGURES

4.1	Architecture Diagram	13
4.2	Data Flow Diagram	14
4.3	UML Diagram	15
4.4	Use Case Diagram	16
4.5	Class Diagram	17
4.6	Sequence Diagram	18
4.7	Dataset	19
4.8	Data cleaning and pre-processing	20
4.9	Pachyderm work-space	21
4.10	Docker desktop	23
5.1	MIDI auto-formatted noise clips	24
5.2	MIDI auto-formatted music clip generated	25
5.3	Terminal execution	25
5.4	Independent Unit module execution	27
5.5	Integration in sub-modules	28
5.6	Testing via LLMS(synthesizer)	29
5.7	Execution of unit commands	30
5.8	Auto-formatted MIDI music file generated	31
6.1	Results of sample code execution	39
9.1	Poster Presentation	49

LIST OF ACRONYMS AND ABBREVIATIONS

abbr	Abbreviation
BS	Batch Script
GUI	Graphical User Interface
KNN	K-Nearest Neighbor
MIDI	Musical Instrument Digital Interface
WAV	Waveform Audio File Format
ML	Machine Learning
AI	Artificial Intelligence
TFR	Tensor Flow Records
UI	User Interface
LSTM	Long Short Term Memory

TABLE OF CONTENTS

	Page.No
ABSTRACT	v
LIST OF FIGURES	vi
LIST OF ACRONYMS AND ABBREVIATIONS	vii
1 INTRODUCTION	1
1.1 Introduction	1
1.2 Aim of the project	2
1.3 Project Domain	2
1.4 Scope of the Project	2
1.5 Methodology	3
2 LITERATURE REVIEW	4
2.1 Generating Music Using Reinforcement Learning . . .	4
2.2 Artificial Music Generation – A Survey	4
2.3 Music Generation Using Deep Learning	5
2.4 Music Generation Algorithms	5
2.5 Deep Learning Techniques for Music Generation . . .	6
2.6 Generation of music pieces using machine learning: long short-term memory neural networks approach . .	6

3	PROJECT DESCRIPTION	8
3.1	Existing System	8
3.2	Proposed System	9
3.3	Feasibility Study	10
3.3.1	Economic Feasibility	10
3.3.2	Technical Feasibility	10
3.3.3	Social Feasibility	10
3.4	System Specification	11
3.4.1	Hardware Specification	11
3.4.2	Software Specification	11
3.4.3	Standards and Policies	12
4	MODULE DESCRIPTION	13
4.1	General Architecture	13
4.2	Design Phase	14
4.2.1	Data Flow Diagram	14
4.2.2	UML Diagram	15
4.2.3	Use Case Diagram	16
4.2.4	Class Diagram	17
4.2.5	Sequence Diagram	18
4.3	Module Description	19
4.3.1	Dataset collection	19
4.3.2	Data Cleaning and Pre Processing	20
4.3.3	Data Loading & Analysing	21
4.3.4	ML model & Algorithm	21
4.3.5	Deployment of the ML model	23
5	IMPLEMENTATION AND TESTING	24
5.1	Input and Output	24

5.1.1	Input Design	24
5.1.2	Output Design	25
5.2	Testing	26
5.3	Types of Testing	26
5.3.1	Unit testing	26
5.3.2	Integration testing	27
5.3.3	Functional testing	28
5.3.4	White Box Testing	29
5.3.5	Black Box Testing	30
5.4	Testing Strategy	31
6	RESULTS AND DISCUSSIONS	32
6.1	Efficiency of the Proposed System	32
6.2	Comparison of Existing and Proposed System	32
6.3	Advantages of the Proposed System	32
6.4	Sample Code	33
7	CONCLUSION AND FUTURE ENHANCEMENTS	40
7.1	Conclusion	40
7.2	Future Enhancements	40
8	PLAGIARISM REPORT	41
9	SOURCE CODE & POSTER PRESENTATION	42
9.1	Source Code	42
9.2	Poster Presentation	49
	References	49

Chapter 1

INTRODUCTION

1.1 Introduction

Composing music is a very interesting challenge that tests the composer's creative capacity, whether its a human or a computer. Although there have been many arguments on the matter, almost all of music is some regurgitation or alteration of a sonic idea created before. Thus, with enough data and the correct algorithm, deep learning should be able to make music that would sound human. The use of deep learning to solve problems in literary arts has been a recent trend that has gained a lot of attention and automated generation of music has been an active area. This project deals with the generation of music using some form of music notation relying on various LSTM(Long Short Term Memory) architectures. Fully connected and convolutional layers are used along with LSTM's to capture rich features in the frequency domain and increase the quality of music generated. The work is focused on unconstrained music generation and uses no information about musical structure such as notes or chords to aid learning.

Our task here is to take some existing music data then train a model using this existing data. The model has to learn the patterns in music that we humans enjoy. Once it learns this, the model should be able to generate new music for us. It cannot simply copy-paste from the

training data. It has to understand the patterns of music to generate new music. We here are not expecting our model to generate new music from the noise input fed, which is of professional quality, but we want it to generate a decent quality music which should be melodious and good to hear.

1.2 Aim of the project

1. To develop a software which is able to convert noise notes into musical notes and further harmonies it to yield a music.
2. To generate music in real time with almost nil production cost.

1.3 Project Domain

The project falls under the domain of Machine learning. Machine learning algorithms can apply what has been learned in the past to new data using labeled examples to predict future events. Starting from the analysis of a known training dataset, the learning algorithm produces an inferred function to make predictions about the output values. The system is able to provide targets for any new input after sufficient training.

1.4 Scope of the Project

The scope of the project is generate fragmented musical notes, cinematic sounds and ambient music through corresponding noises which would bring a great convivence to the music industry and to the artists. And will let non-professional to get started with least setup.

1.5 Methodology

We will be developing a transformer model using magenta, which would be trained on a data set which consists of a particular genre of music, inhere we are using the ambient genre. The data set is cleaned and pre-processed and converted into the transformer readable format i.e MIDI. The transofrmer model is basically a machine learning model which makes uses of KNN algorithm to match the input data to the nearest trained dataset. After long hours of training on pachyderm, the transformer model is then deployed on docker for testing.

Chapter 2

LITERATURE REVIEW

2.1 Generating Music Using Reinforcement Learning

A model of music needs to have the ability to recall past details and have a clear, coherent understanding of musical structure. Detailed in the paper is a deep reinforcement learning architecture that predicts and generates polyphonic music aligned with musical rules. The probabilistic model presented is a Bi-axial LSTM trained with a “kernel” reminiscent of a convolutional kernel. To encourage exploration and impose greater global coherence on the generated music, a deep reinforcement learning (DQN) approach is adopted. When analyzed quantitatively and qualitatively, this approach performs well in composing polyphonic music.

2.2 Artificial Music Generation – A Survey

The quest for autonomous music-making systems may be an interesting perspective for exploring the process of composition⁸ and it also serves as an evaluation method. An example of a musical Turing test⁹ will be introduced in Section 6.14.2. It consists in presenting to various

members of the public (from beginners to experts) chorales composed by J. S. Bach or generated by a deep learning system and played by human musicians¹⁰. As we will see in the following, deep learning techniques turn out to be very efficient at succeeding in such tests, due to their capacity to learn musical style from a given corpus and to generate new music that fits into this style. That said, we consider that such a test is more a means than an end.

2.3 Music Generation Using Deep Learning

This paper deals with the generation of music using raw audio files in the frequency domain relying on various LSTM architectures. Fully connected and convolutional layers are used along with LSTM's to capture rich features in the frequency domain and increase the quality of music generated. The work is focused on unconstrained music generation and uses no information about musical structure (notes or chords) to aid learning. The music generated from various architectures are compared using blind fold tests. Using the raw audio to train models is the direction to tapping the enormous amount of mp3 files that exist over the internet without requiring the manual effort to make structured MIDI files. Moreover, not all audio files can be represented with MIDI files making the study of these models an interesting prospect to the future of such models.

2.4 Music Generation Algorithms

There has been a lot of work where musical features such as notes, chords and notations have been used to generate music using LSTMs[3,4,5].

These works show promising results and demonstrate that LSTMs have the ability to capture enough long-range information required for music generation. A typical architecture in these approaches involves structured input of a music notation from MIDI files that are encoded as vectors and fed into an LSTM at each timestep. The LSTM then predicts the encoding of the next timestep and this continues. The error is computed as negative log loss of the predicted value and the true value.

2.5 Deep Learning Techniques for Music Generation

All of the approaches deal with the frequency representation of the audio samples. The audio used is sampled at 16KHz and we only consider a single channel of audio for the sake of simplicity. Consider the raw audio samples to be represented as $\{a_0, a_1, a_2, \dots, a_n\}$, the Fourier transform is obtained by considering n of these samples at a time to obtain a n dimensional vector. Each of the values in the n dimensional frequency vector consist of real and imaginary components, to simplify calculations, the vector is unrolled as $2n$ dimensional vector (D) where the imaginary values are appended after the n real values. The LSTM used in all the approaches are the vanilla LSTMs.

2.6 Generation of music pieces using machine learning: long short-term memory neural networks approach

In this paper, they explored the usage of long short-term memory neural network (NN) in generating music pieces and propose an approach to do so. Bach's musical style has been selected to train the

NN to make it able to generate similar music pieces. The proposed approach takes midi files, converting them to song files and then encoding them to be as inputs for the NN. Before inputting the files into the NNs, an augmentation process which augments the file into different keys is performed then the file is fed into the NN for training. The last step is the music generation. The main objective is to provide the NN with an arbitrary note and then the NN starts amending it gradually until producing a good piece of music. Various experiments have been conducted to explore the best values of parameters that can be selected to obtain good music generations. The obtained generated music pieces are accepted in terms of rhythm and harmony; however, some other problems exist such as in certain cases the tone stops or in some other cases getting short melodies that do not change

Chapter 3

PROJECT DESCRIPTION

3.1 Existing System

The existing system directly models the raw audio samples for training and produce music independently without correspondence noise note, the best results are obtained from the paper[2] which uses a very deep dilated convolutional network to generate samples one at a time sampled at 16KHz. By increasing the amount of dilation at each depth they were able to capture long range dependencies to obtain a good representation of the audio. It is easy to imagine the kind of depth they had to attain in order capture enough time information that could encode the song well. Despite the large depth, training the network was relatively easy because they treated the problem as a classification problem where they classified the generated audio sample into one of 255 values. This allowed them to use the negative log loss instead of the mean squared loss. This reduced the likelihood of overfitting to outliers and thus decreased convergence time. Although this method works, the depth makes it extremely demanding in terms of computation, it takes about a minute to generate one second of audio on Google's GPU clusters forcing us to consider a faster alternative and something which produces based on the input fed.

3.2 Proposed System

Our method to algorithmically generate music is to train a probabilistic model. Model the music as a probability distribution, mapping measures, or sequences of notes based on likelihood of appearance in the corpus of training music via RNN and KNN algorithm. These probabilities are learnt from the input data without prior specification of particular musical rules. The algorithm uncovers patterns from the music alone. After the model is trained, new music is generated in sequences based the input data fed. This generated music comes from a sampling of the learned probability distribution. This approach is complicated by the structure of music. Structurally, most music contains a melody, or a key sequence of notes with a single instrument or vocal theme. This melody can be monodic, meaning at most one note per time step. The melody can also be polyphonic, meaning greater than one note per time step.

Advantages:

- The models proposed showed improved performance in the quality of music over the base model and provides more insights into the architectures that would suit raw audio representations.
- It would help in music production via providing cheaper alternatives and help amateurs get started with music production in a cheaper way.

3.3 Feasibility Study

The goal of a feasibility report is to objectively and logically reveal the strengths and limitations of a current or planned system, as well as the possibilities and risks that exist in the environment, the resources needed to carry it out, and the likelihood of success. In its most basic form, the two factors for determining feasibility are indeed the required cost and the value to be obtained. A well-designed feasibility study should include a historical overview of the project; in most cases, feasibility studies come before technical design and implementation. A feasibility study assesses the project's likelihood of success.

3.3.1 Economic Feasibility

The project doesn't demand much of economic support, it would probably be an open source software and can be easily made utilised by any target group.

3.3.2 Technical Feasibility

The proposed system makes use of SAAS platforms for training and deployment. These platforms are open source and can handle our heavy requirements, and it is to specify that these are openly available and falls under the limits of today's technology.

3.3.3 Social Feasibility

The proposed system would be widely applicable for music industry, it would help any amateur to get started with music production in a cheaper and effective way.

3.4 System Specification

3.4.1 Hardware Specification

- Processor : i5 or above
- Hard disk : minimum 20 GB
- RAM : minimum 4 GB
- GPU : minimum 4 GB

3.4.2 Software Specification

- Operating System : Linux(ubuntu)

we are making use of Linux as the pachyderm dependencies are automatically handled in Linux using a BS, which would in turn reduces our risk factor.

- Tool : Pachyderm

Pachyderm is a tool for version-controlled, automated, end-to-end data pipelines for data science. If you need to chain together data scraping, ingestion, cleaning, munging, wrangling, processing, modeling, and analysis in a sane way, while ensuring the traceability and provenance of your data, Pachyderm is for you. If you have an existing set of scripts which do this in an ad-hoc fashion and you're looking for a way to "productionize" them, Pachyderm can make this easy for you.

- Tool : Docker Desktop

Docker is a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own

software, libraries and configuration files; they can communicate with each other through well-defined channels.

- Library : magenta

Magenta is distributed as an open source Python library, powered by TensorFlow. This library includes utilities for manipulating source data (primarily music and images), using this data to train machine learning models, and finally generating new content from these models.

- Library : PythonInMusic

it provides various methods to convert one format of music into any format.

3.4.3 Standards and Policies

- Pachyderm: an open source SaaS platform for framing machine learning model and training on huge datasets.
- Docker: an opensource SaaS platform for model deployment for faster execution in real-time.
- Docker type used: docker desktop Standard Used: Stable release 2.0.3
- Magenta: Standard Used: beta release 2.0.
- IS 15930-1
- Division Name : Music Generation
- Section Name : Music direction and production

Chapter 4

MODULE DESCRIPTION

4.1 General Architecture

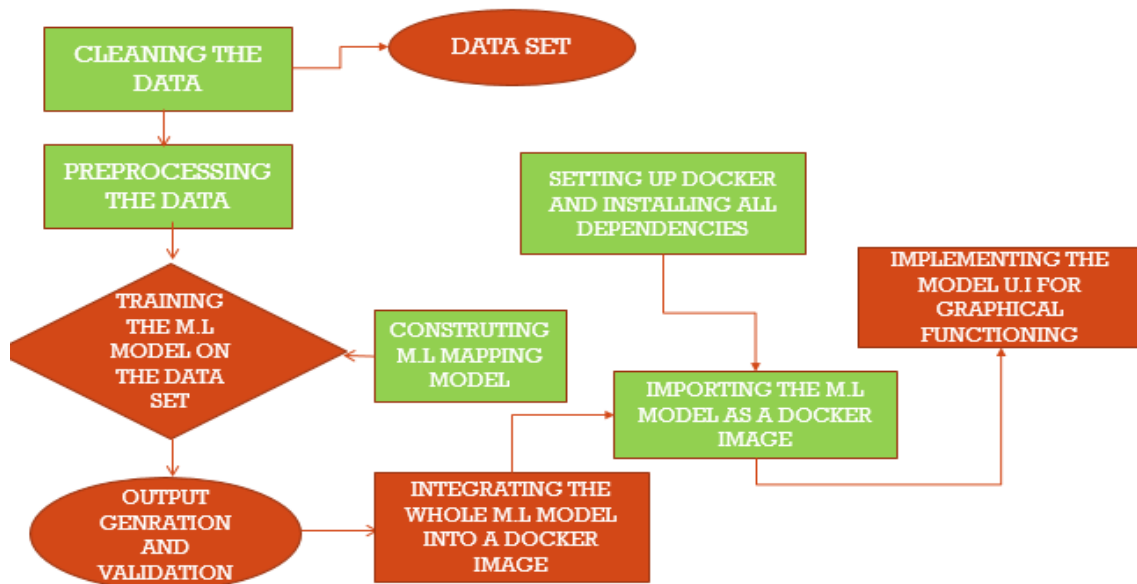


Figure 4.1: **Architecture Diagram**

As its clear in the above architecture, first we will collect a significant amount of data set from sources like spotify and soundcloud, this data is further cleaned and pre-processed. The Framed machine learning model is trained on this dataset and tested. Further it will be deployed on a Saas platform named docker desktop for easy and faster execution.

4.2 Design Phase

4.2.1 Data Flow Diagram

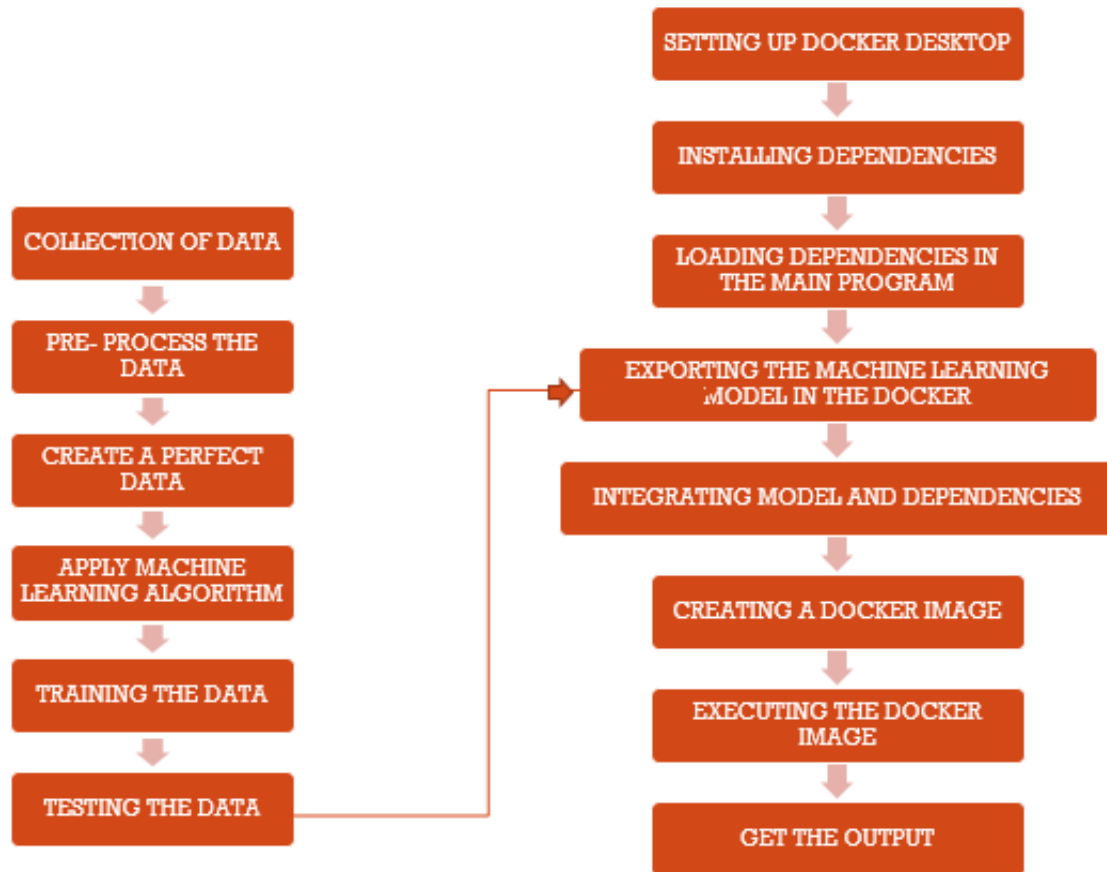


Figure 4.2: Data Flow Diagram

1. First step is to collect the data for the required implementation.
2. Then the data should be checked whether it is suitable data or any unwanted data is present if any unwanted data is present we have to pre-process the data using some pre-processing techniques..
3. We have to train and test the data set for getting equivalent results.
4. This data is then loaded into pachyderm for training.
5. The ML model will then be ready for execution via a GUI.

4.2.2 UML Diagram

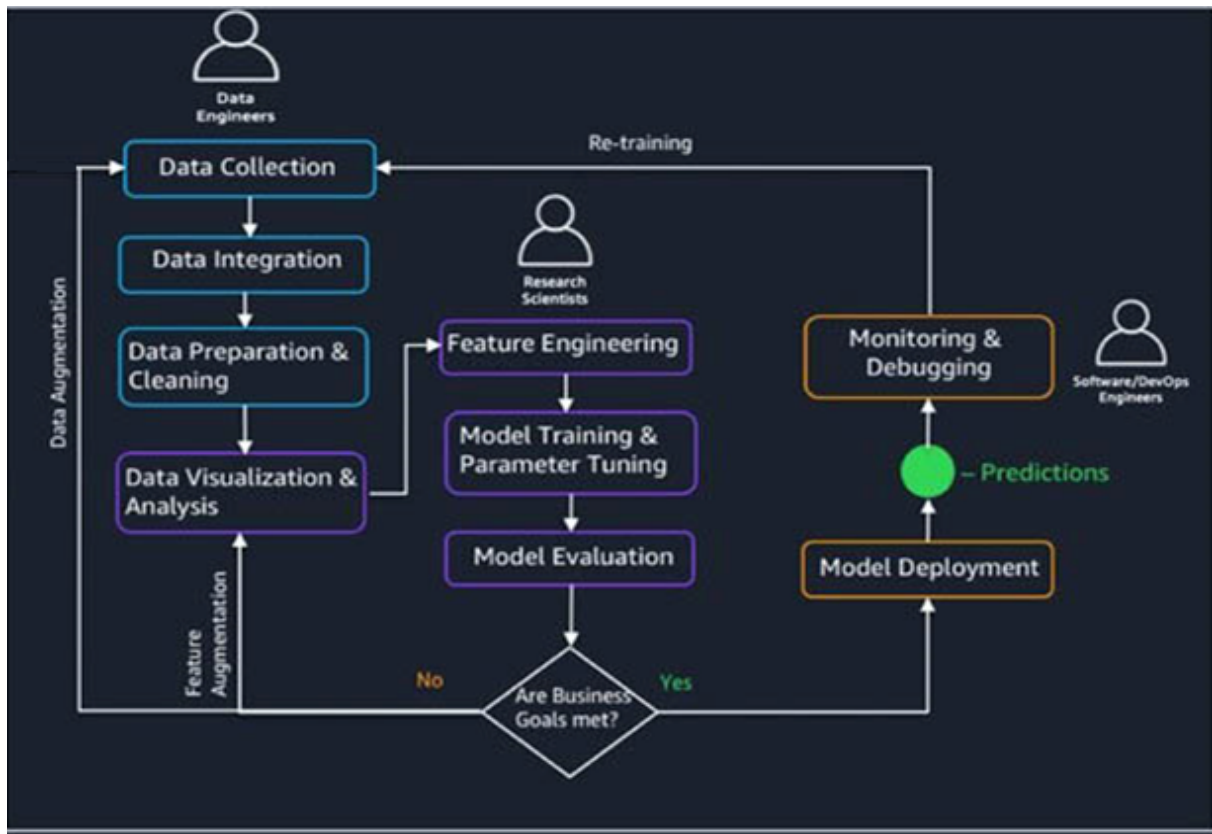


Figure 4.3: UML Diagram

UML is an acronym that stands for Unified Modeling Language. Simply put, UML is a modern approach to modeling and documenting software. In fact, it's one of the most popular business process modeling techniques.

It is based on diagrammatic representations of software components. As the old proverb says: “a picture is worth a thousand words”. By using visual representations, we are able to better understand possible flaws or errors in software or business processes.

4.2.3 Use Case Diagram

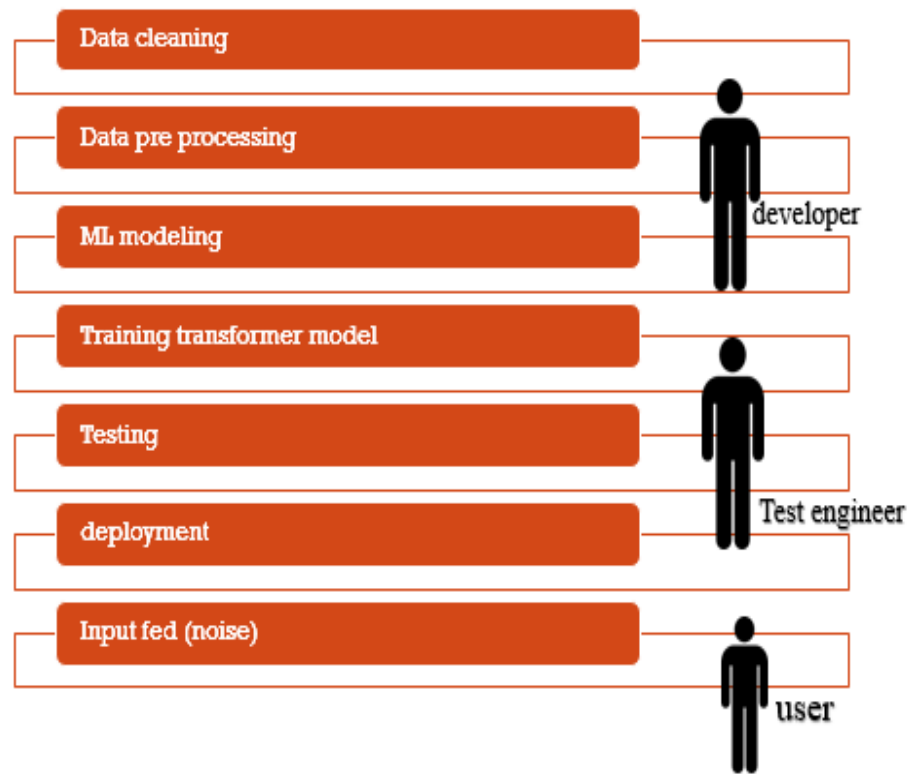


Figure 4.4: Use Case Diagram

A use case diagram is a graphical depiction of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses. Use case diagrams are considered for high level requirement analysis of a system. So when the requirements of a system are analyzed the functionalities are captured in use cases. So we can say that uses cases are nothing but the system functionalities written in an organized manner.

4.2.4 Class Diagram

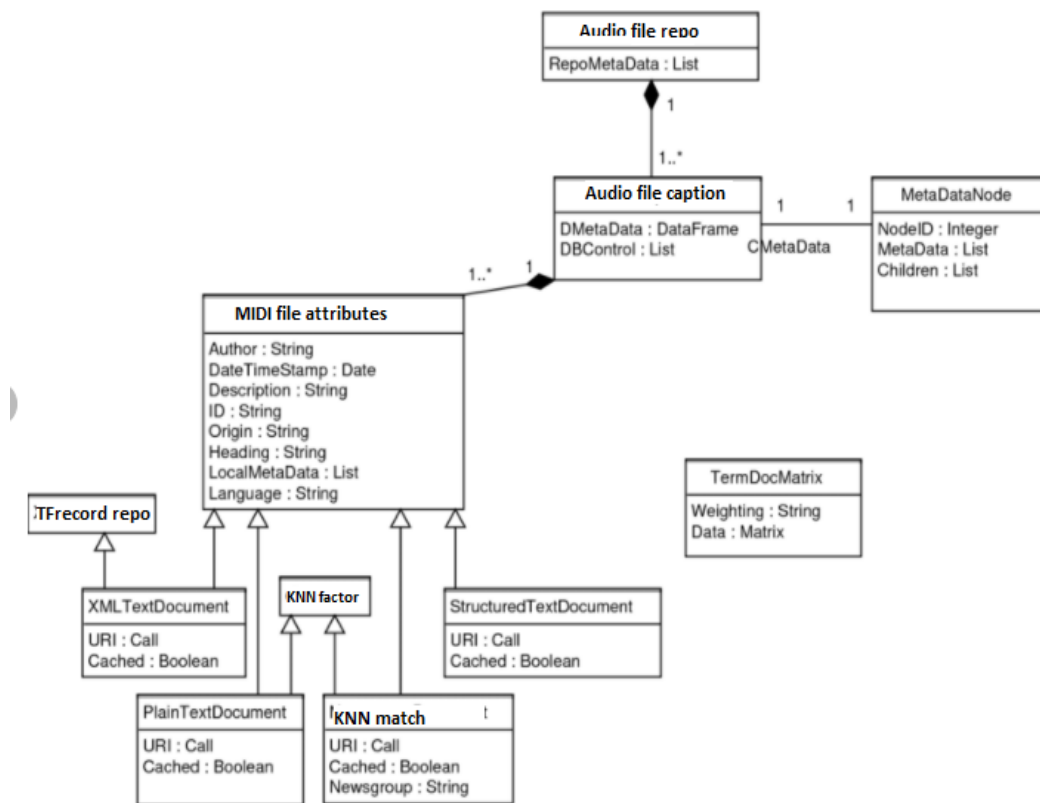


Figure 4.5: Class Diagram

The class diagram is the main building block of object-oriented modeling. It is used for general conceptual modeling of the structure of the application, and for detailed modeling, translating the models into programming code. Class diagrams can also be used for data modeling. Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. So a collection of class diagrams represent the whole system. The name of the class diagram should be meaningful to describe the aspect of the system.

4.2.5 Sequence Diagram

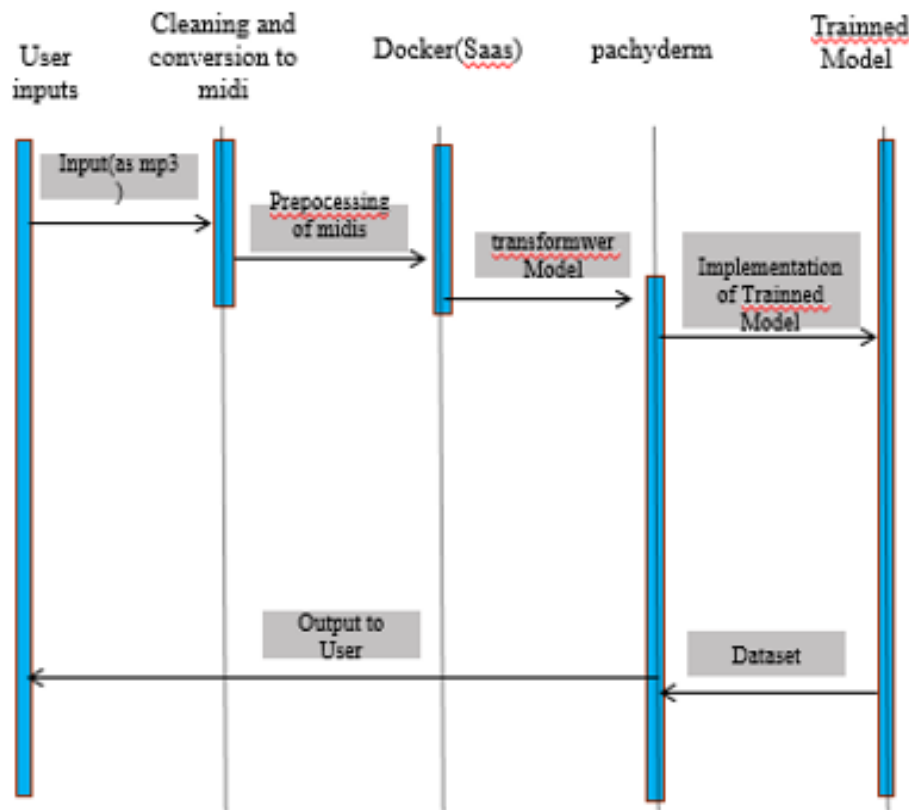


Figure 4.6: Sequence Diagram

A sequence diagram is a type of interaction diagram because it describes how—and in what order—a group of objects works together. These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process. Sequence diagrams are sometimes known as event diagrams or event scenarios.

4.3 Module Description

Following are the modules of the proposed system

4.3.1 Dataset collection

Project Name	Project Manager	Project Status	Project Start Date	Project End Date	Project Duration	Project Budget	Project Actual Cost	Project Variance	Project Risk	Project Quality	Project Customer Satisfaction
Project A	John Doe	Completed	2023-01-01	2023-03-31	90 Days	\$1,000,000	\$950,000	\$50,000	Low	High	95%
Project B	Jane Smith	In Progress	2023-02-01	2023-05-31	120 Days	\$2,500,000	\$2,200,000	\$300,000	Medium	Medium	88%
Project C	Mike Johnson	On Hold	2023-03-01	2023-06-30	120 Days	\$1,500,000	\$1,500,000	\$0	High	Low	72%
Project D	Sarah Lee	Completed	2023-04-01	2023-07-31	120 Days	\$800,000	\$820,000	-\$20,000	Low	High	92%
Project E	David Kim	In Progress	2023-05-01	2023-08-31	120 Days	\$3,000,000	\$3,100,000	-\$100,000	Medium	Medium	85%
Project F	Emily White	On Hold	2023-06-01	2023-09-30	120 Days	\$1,200,000	\$1,200,000	\$0	High	Low	70%
Project G	Chris Brown	Completed	2023-07-01	2023-10-31	120 Days	\$900,000	\$880,000	\$20,000	Low	High	90%
Project H	Alex Green	In Progress	2023-08-01	2023-11-30	120 Days	\$2,000,000	\$1,900,000	\$100,000	Medium	Medium	87%
Project I	Olivia Black	On Hold	2023-09-01	2023-12-31	120 Days	\$1,800,000	\$1,800,000	\$0	High	Low	75%
Project J	Noah Grey	Completed	2023-10-01	2024-01-31	120 Days	\$1,100,000	\$1,050,000	\$50,000	Low	High	93%
Project K	Isabella Blue	In Progress	2023-11-01	2024-02-28	120 Days	\$2,200,000	\$2,100,000	\$100,000	Medium	Medium	86%
Project L	Liam Red	On Hold	2023-12-01	2024-03-31	120 Days	\$1,600,000	\$1,600,000	\$0	High	Low	73%
Project M	Mia Yellow	Completed	2024-01-01	2024-04-30	120 Days	\$1,300,000	\$1,250,000	\$50,000	Low	High	91%
Project N	Ben Green	In Progress	2024-02-01	2024-05-31	120 Days	\$2,800,000	\$2,700,000	\$100,000	Medium	Medium	89%
Project O	Charlotte Blue	On Hold	2024-03-01	2024-06-30	120 Days	\$1,700,000	\$1,700,000	\$0	High	Low	74%
Project P	Lucas Red	Completed	2024-04-01	2024-07-31	120 Days	\$1,000,000	\$980,000	\$20,000	Low	High	94%
Project Q	Hannah Yellow	In Progress	2024-05-01	2024-08-31	120 Days	\$2,100,000	\$2,000,000	\$100,000	Medium	Medium	87%
Project R	Ethan Green	On Hold	2024-06-01	2024-09-30	120 Days	\$1,900,000	\$1,900,000	\$0	High	Low	76%
Project S	Ava Blue	Completed	2024-07-01	2024-10-31	120 Days	\$1,400,000	\$1,350,000	\$50,000	Low	High	92%
Project T	Jack Red	In Progress	2024-08-01	2024-11-30	120 Days	\$2,300,000	\$2,200,000	\$100,000	Medium	Medium	88%
Project U	Chloe Yellow	On Hold	2024-09-01	2025-02-28	120 Days	\$1,800,000	\$1,800,000	\$0	High	Low	77%
Project V	Ryan Green	Completed	2024-10-01	2025-01-31	120 Days	\$1,100,000	\$1,080,000	\$20,000	Low	High	93%
Project W	Natalie Blue	In Progress	2024-11-01	2025-02-28	120 Days	\$2,400,000	\$2,300,000	\$100,000	Medium	Medium	89%
Project X	Kevin Red	On Hold	2024-12-01	2025-03-31	120 Days	\$1,600,000	\$1,600,000	\$0	High	Low	78%
Project Y	Madison Yellow	Completed	2025-01-01	2025-04-30	120 Days	\$1,500,000	\$1,450,000	\$50,000	Low	High	94%
Project Z	Christopher Green	In Progress	2025-02-01	2025-05-31	120 Days	\$2,600,000	\$2,500,000	\$100,000	Medium	Medium	90%

Figure 4.7: **Dataset**

The data has been collected from music platforms like Spotify and SoundCloud, where we have extracted only a specific type of genre, i.e Ambient music genre.

4.3.2 Data Cleaning and Pre Processing

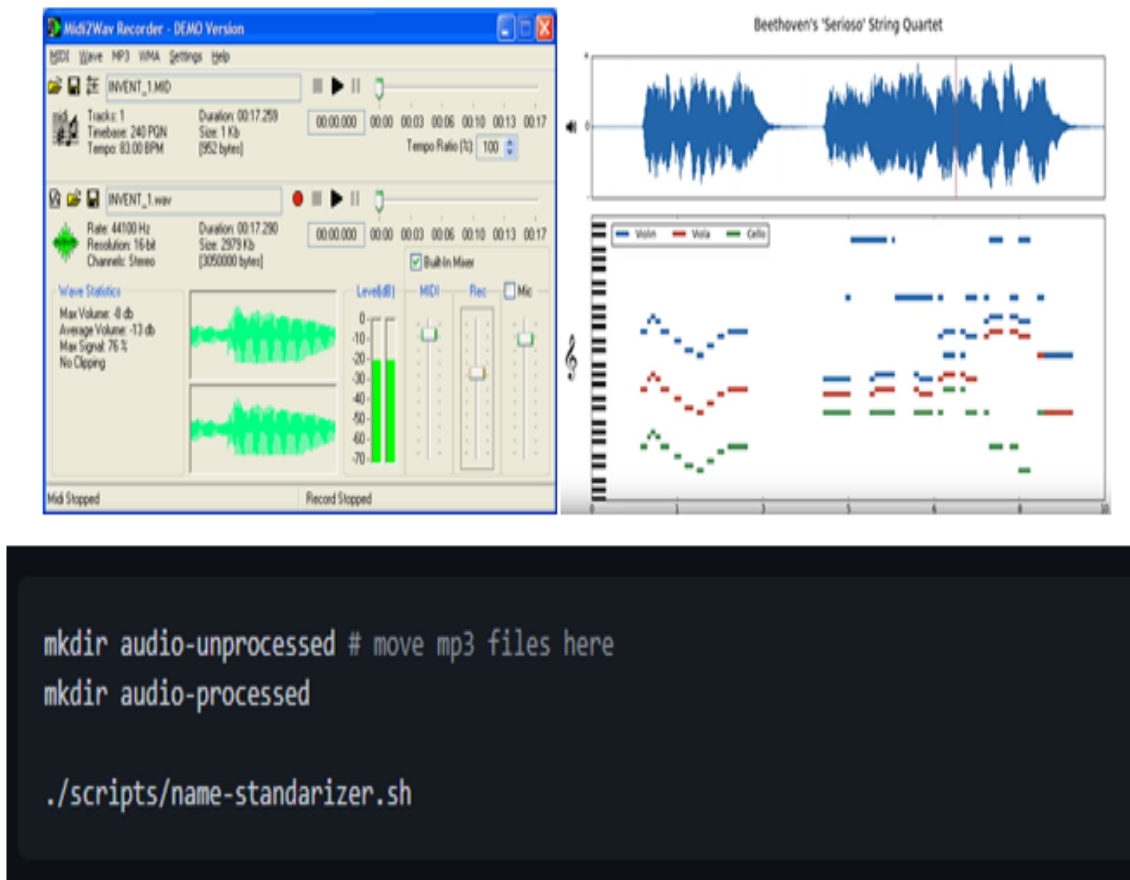


Figure 4.8: Data cleaning and pre-processing

We used a little Bash script to rename all the MP3s, making them lowercase and removing all spaces to give me a standard naming convention. The script then pushes them up to the Pachyderm audio-unprocessed repo that gets it all started. Here we use tools to convert mp3 format files into WAVS and further into MIDI, which is our desired format to load into pachyderm for training. Before training the model, we need to do one more step. We've got to convert those MIDI files into a format TensorFlow can understand, called the TFRecord format.

4.3.3 Data Loading & Analysing

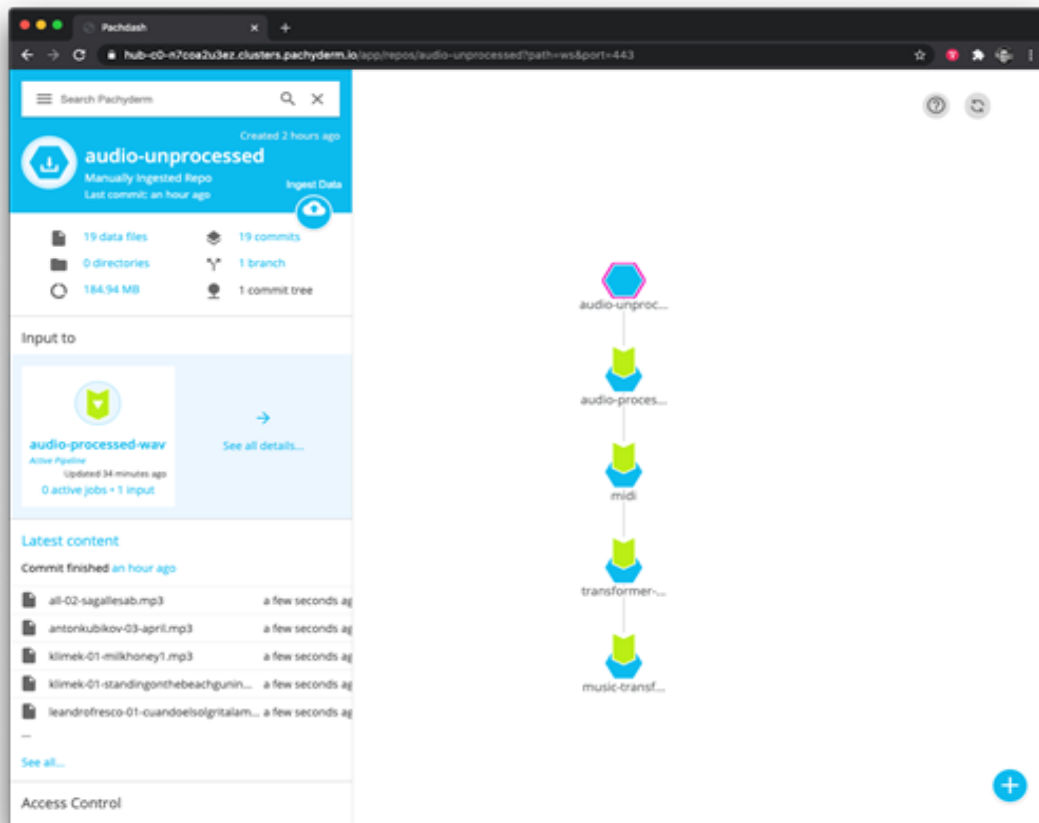


Figure 4.9: Pachyderm work-space

The data is loaded on an open source SaaS platform named Pachyderm. Data analysing is one of the important skill in the applied statistics and the machine learning. This can be helpful when exploring about the data to know the patterns, corrupt data, Outliers and much more.

4.3.4 ML model & Algorithm

In this proposed system we have used three different types of Machine learning Supervised algorithms to compare the closeness of the each audio note.

KNN Algorithm

K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique. K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories. K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm. K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data. KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

LSTM Technique

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network capable of learning order dependence in sequence prediction problems. This is a behavior required in complex problem domains like machine translation, speech recognition, and more.

RNN Algorithm

Recurrent neural networks (RNN) are the state of the art algorithm for sequential data and are used by Apple's Siri and and Google's voice search. It is the first algorithm that remembers its input, due to an internal memory, which makes it perfectly suited for machine learning problems that involve sequential data. It is one of the algorithms behind the scenes of the amazing achievements seen in deep learning over the past few years. In this post, we'll cover the basic concepts of

how recurrent neural networks work, what the biggest issues are and how to solve them.

4.3.5 Deployment of the ML model

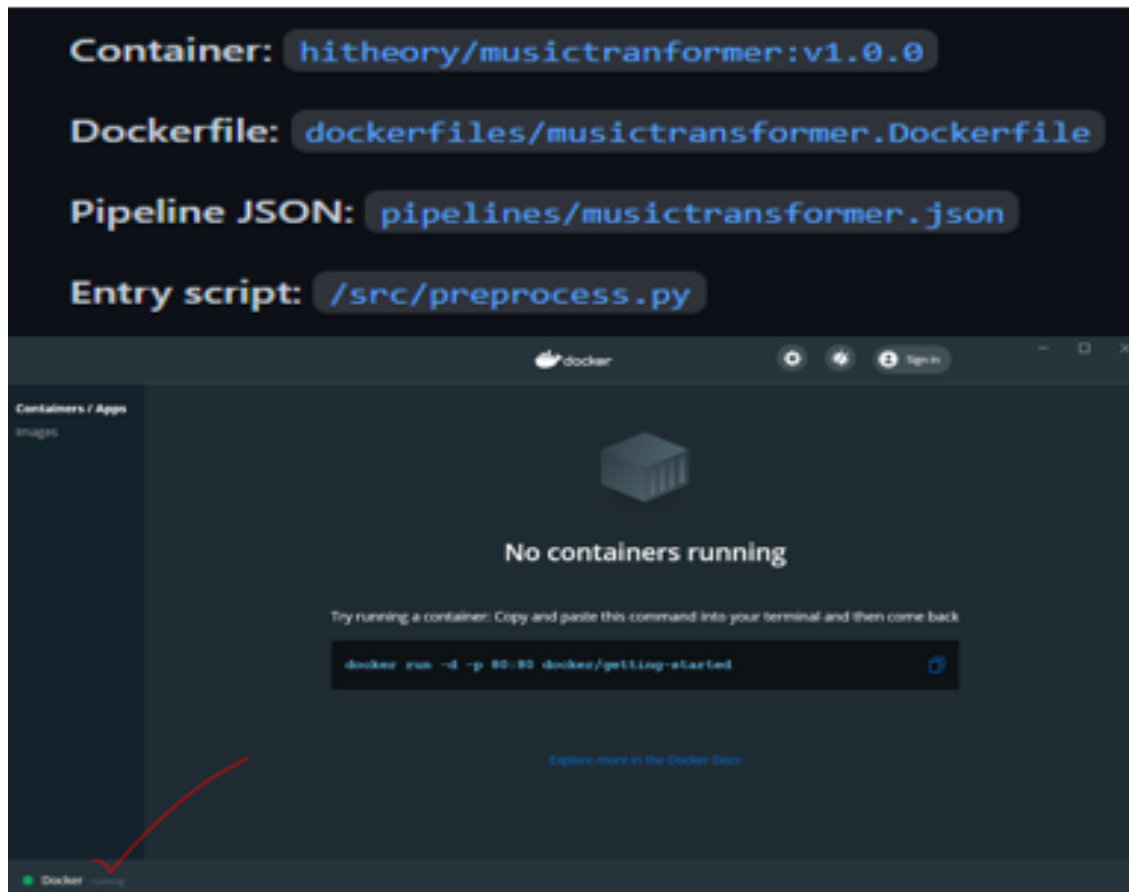


Figure 4.10: **Docker desktop**

In pachyderm the machine learning module, completely act as a one single unit which is then imported and deployed on another SaaS platform named Docker, which makes the real-time execution much efficient.

Chapter 5

IMPLEMENTATION AND TESTING

5.1 Input and Output

The input consists of a noise clip, which is preferably a musical instrument noise or any cinematic noise. And the output would include a musical clip, corresponding to the noise clip fed.

5.1.1 Input Design

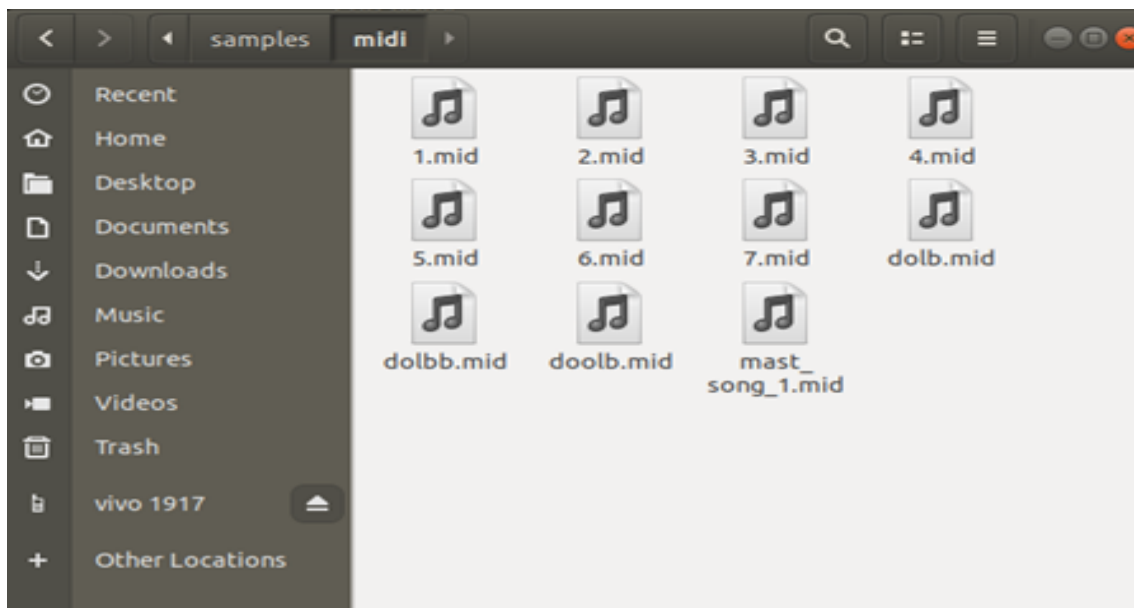


Figure 5.1: MIDI auto-formatted noise clips

5.1.2 Output Design

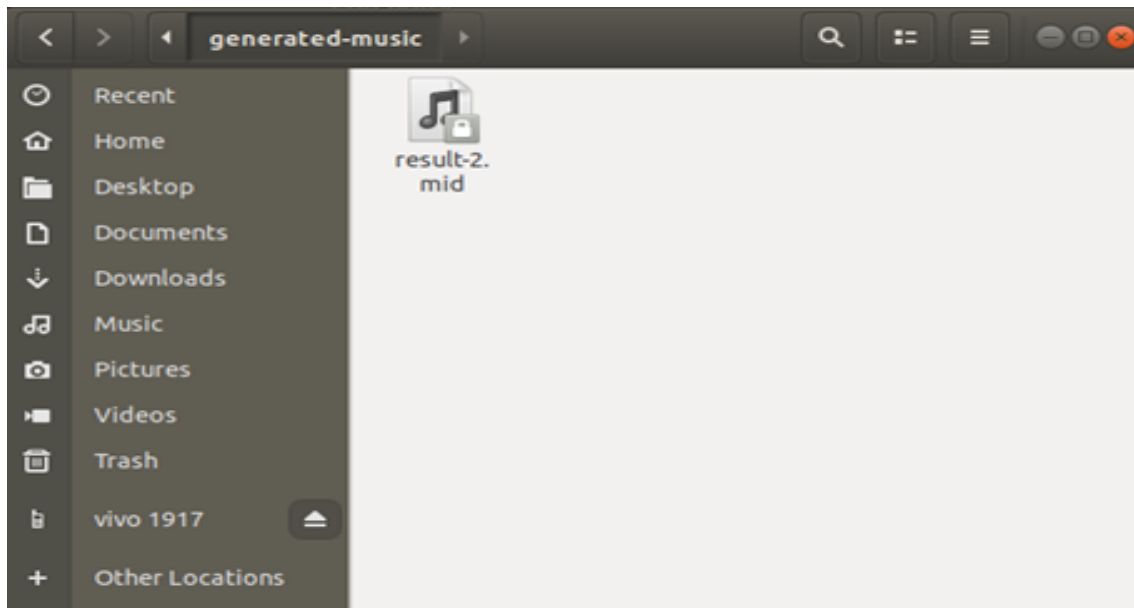


Figure 5.2: MIDI auto-formatted music clip generated

```
notshiv@water-gun: ~/ambient-music-generation
File Edit View Search Terminal Help
ights) but not all checkpointed values were used. See above for spec
ific issues. Use expect_partial() on the load status object, e.g. tf
.train.Checkpoint.restore(...).expect_partial(), to silence these wa
rnings, or use assert_consumed() to make the check explicit. See htt
ps://www.tensorflow.org/guide/checkpoint#loading_mechanics for detai
ls.
notshiv@water-gun:~/ambient-music-generation$ docker run -it -v `pwd
`::/data --entrypoint python jimmywhitaker/music-transformer:0.1 gene
rate.py --load_path=/data/trained-models/ambient-musictransformer-mo
del-2-june-3-2020-750-epochs-more-data --inputs /data/samples/midi/d
oolb.mid --length=348 --save_path=/data/generated-music/result-2.mid
2021-05-11 21:31:49.064162: I tensorflow/stream_executor/platform/de
fault/dso_loader.cc:44] Successfully opened dynamic library libcuda.
so.1
2021-05-11 21:31:49.064207: E tensorflow/stream_executor/cuda/cuda_d
river.cc:313] failed call to cuInit: UNKNOWN ERROR (-1)
2021-05-11 21:31:49.064244: I tensorflow/stream_executor/cuda/cuda_d
iagnostics.cc:156] kernel driver does not appear to be running on th
is host (e219eb56d2d1): /proc/driver/nvidia/version does not exist
2021-05-11 21:31:49.064620: I tensorflow/core/platform/cpu_feature_g
uard.cc:143] Your CPU supports instructions that this TensorFlow bin
ary was not compiled to use: AVX2 FMA
2021-05-11 21:31:49.092264: I tensorflow/core/platform/profile_utils
/cpu_utils.cc:102] CPU Frequency: 1800000000 Hz
2021-05-11 21:31:49.093030: I tensorflow/compiler/xla/service/servic
e.cc:168] XLA service 0x7fb808000b20 initialized for platform Host (
this does not guarantee that XLA will be used). Devices:
2021-05-11 21:31:49.093069: I tensorflow/compiler/xla/service/servic
e.cc:176] StreamExecutor device (0): Host, Default Version
>> generate with decoder wise... beam size is None
generating |#####| 272/348
```

Figure 5.3: Terminal execution

5.2 Testing

There are some testings should be done before a software is proposed

5.3 Types of Testing

5.3.1 Unit testing

The different modules are tested isolately and independent of each other modules. And the results are obtained. Unit Testing that test the single components in the proposed system Individual units of code is being tested in the unit testing.

Input

```
1 # Deprecated modules which should not be used , separated by a comma
2 deprecated-modules=regsub ,
3             TERMIOS,
4             Bastion ,
5             rexec ,
6             sets
7
8 # Create a graph of every (i.e. internal and external) dependencies in the
9 # given file (report RP0402 must not be disabled)
10 #this will diplay all the unit modules
```

Test result

```
75  CONSOLE_SCRIPTS = [  
76      'magenta.interfaces.midi.magenta_midi',  
77      'magenta.interfaces.midi.midi_clock',  
78      'magenta.models.arbitrary_image_stylization.arbitrary_image_stylization_evaluate',  
79      'magenta.models.arbitrary_image_stylization.arbitrary_image_stylization_train',  
80      'magenta.models.arbitrary_image_stylization.arbitrary_image_stylization_with_weights',  
81      'magenta.models.arbitrary_image_stylization.arbitrary_image_stylization_distill_mobilenet',  
82      'magenta.models.drums_rnn.drums_rnn_create_dataset',  
83      'magenta.models.drums_rnn.drums_rnn_generate',  
84      'magenta.models.drums_rnn.drums_rnn_train',  
85      'magenta.models.image_stylization.image_stylization_create_dataset',  
86      'magenta.models.image_stylization.image_stylization_evaluate',  
87      'magenta.models.image_stylization.image_stylization_finetune',  
88      'magenta.models.image_stylization.image_stylization_train',  
89      'magenta.models.image_stylization.image_stylization_transform',  
90      'magenta.models.improv_rnn.improv_rnn_create_dataset',  
91      'magenta.models.improv_rnn.improv_rnn_generate',  
92      'magenta.models.improv_rnn.improv_rnn_train',  
93      'magenta.models.gansynth.gansynth_train',  
94      'magenta.models.gansynth.gansynth_generate',  
95      'magenta.models.melody_rnn.melody_rnn_create_dataset',  
96      'magenta.models.melody_rnn.melody_rnn_generate',  
97      'magenta.models.melody_rnn.melody_rnn_train',  
98      'magenta.models.music_vae.music_vae_generate',  
99      'magenta.models.music_vae.music_vae_train',  
100     'magenta.models.nsynth.wavenet.nsynth_generate',  
101     'magenta.models.nsynth.wavenet.nsynth_save_embeddings',  
102     'magenta.models.onsets_frames_transcription.onsets_frames_transcription_create_dataset',  
103     'magenta.models.onsets_frames_transcription.onsets_frames_transcription_create_dataset_maps',  
104     'magenta.models.onsets_frames_transcription.onsets_frames_transcription_create_tfrecords',
```

Figure 5.4: Independent Unit module execution

5.3.2 Integration testing

In the Integration testing we are testing multiple components or units of the code to check the associate difference between the different programming parts

Input

```
1  {  
2      "pipeline": {  
3          "name": "audio-processed-wav"  
4      },  
5      "description": "A pipeline that converts MP3s to WAVs with ffmpeg.",  
6      "transform": {  
7          "cmd": [ "bash", "/ffmpeg-convert-pachy.sh" ],  
8          "image": "rabbit37/audio-preprocessor:v5"  
9      },  
10     "input": {
```

```

11     "pfs": {
12         "repo": "audio-unprocessed",
13         "glob": "/*"
14     }
15 }
16 }

```

Test result

```

notshiv@water-gun: ~/ambient-music-generation
File Edit View Search Terminal Help
ights) but not all checkpointed values were used. See above for specific
ific issues. Use expect_partial() on the load status object, e.g. tf
.train.Checkpoint.restore(...).expect_partial(), to silence these wa
rnings, or use assert_consumed() to make the check explicit. See htt
ps://www.tensorflow.org/guide/checkpoint#loading_mechanics for detai
ls.
notshiv@water-gun:~/ambient-music-generation$ docker run -it -v `pwd
`:/data --entrypoint python jimmywhitaker/music-transformer:0.1 gene
rate.py --load_path=/data/trained-models/ambient-musictransformer-mo
del-2-june-3-2020-750-epochs-more-data --inputs /data/samples/midi/d
oolb.mid --length=348 --save_path=/data/generated-music/result-2.mid
2021-05-11 21:31:49.064162: I tensorflow/stream_executor/platform/de
fault/dso_loader.cc:44] Successfully opened dynamic library libcuda.
so.1
2021-05-11 21:31:49.064207: E tensorflow/stream_executor/cuda/cuda_d
river.cc:313] failed call to cuInit: UNKNOWN ERROR (-1)
2021-05-11 21:31:49.064244: I tensorflow/stream_executor/cuda/cuda_d
iagnostics.cc:156] kernel driver does not appear to be running on th
is host (e219eb56d2d1): /proc/driver/nvidia/version does not exist
2021-05-11 21:31:49.064620: I tensorflow/core/platform/cpu_feature_g
uard.cc:143] Your CPU supports instructions that this TensorFlow bin
ary was not compiled to use: AVX2 FMA
2021-05-11 21:31:49.092264: I tensorflow/core/platform/profile_utils
/cpu_utils.cc:102] CPU Frequency: 1800000000 Hz
2021-05-11 21:31:49.093030: I tensorflow/compiler/xla/service/servic
e.cc:168] XLA service 0x7fb808000b20 initialized for platform Host (
this does not guarantee that XLA will be used). Devices:
2021-05-11 21:31:49.093069: I tensorflow/compiler/xla/service/servic
e.cc:176] StreamExecutor device (0): Host, Default Version
>> generate with decoder wise... beam size is None
generating |#####| 272/348

```

Figure 5.5: Integration in sub-modules

5.3.3 Functional testing

Functional tests are procedures for ensuring that all elements or components of a piece of programming or software work properly. Functional testing examines the application's user interface to guarantee that all user criteria for a properly functioning program are met.

Input

```
python3 /src/train.py --epochs 500 --save_path /pfs/out --input_path /pfs/
transformer-preprocess --batch-size 2 --max-seq 2048
```

Test Result



Figure 5.6: Testing via LLMS(synthesizer)

5.3.4 White Box Testing

It is a testing in which the internal structure of the item is known as tester. Development process and programming are checked to determine input-output flow and also to improve design, usability, and security.

Input

```
cd pipelines
pachctl create pipeline -f ./mp3-to-wav.json
```


Test result

```
# Once Pachyderm is set up, run the following:
pachctl create repo audio-unprocessed

### Clean Your Data
mkdir audio-unprocessed # move downloaded mp3 files here
mkdir audio-processed
./scripts/name-standarizer.sh # standardizes and uploads your data to pachyderm

# Convert MP3s to Waves
cd pipelines/
pachctl create pipeline -f ./mp3-to-wav.json

# Converting WAVs to MIDI
pachctl create pipeline -f ./midi.json

# MIDI to TFRecord
pachctl create pipeline -f ./transformer-preprocess.json

# Train the Transformer
pachctl create pipeline -f ./music-transformer.json
```

Figure 5.7: Execution of unit commands

5.3.5 Black Box Testing

Black box testing will be executed manually or automatically. Because one of the desired outcomes of black box testing to conform the end users will be able to use, the product . It is mainly focused on external or end-user perspective

Test result

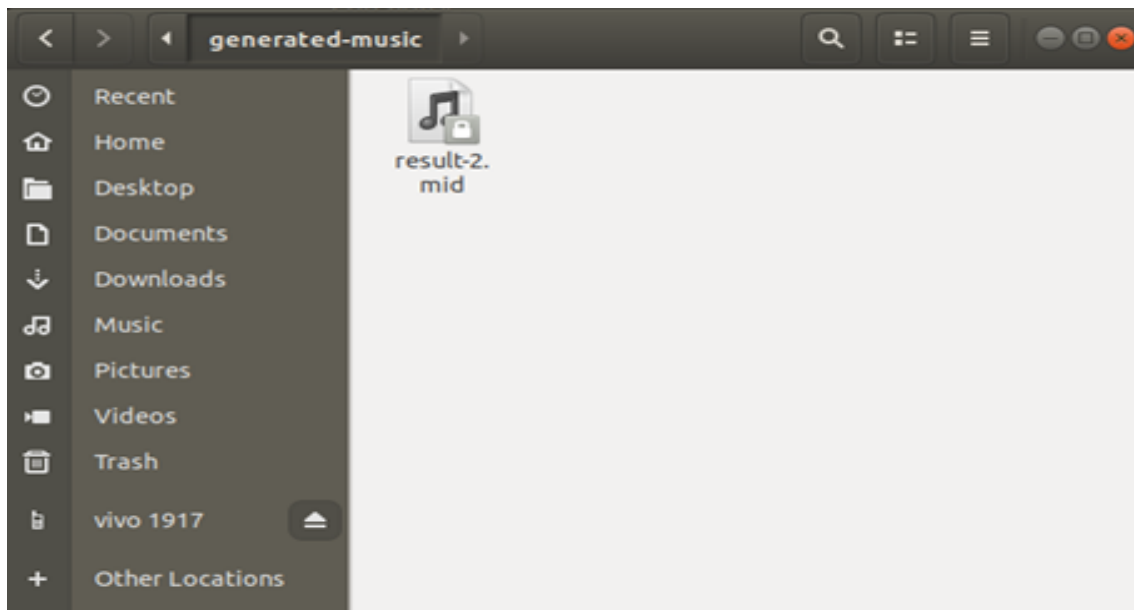


Figure 5.8: Auto-formatted MIDI music file generated

5.4 Testing Strategy

A technique for structure testing fuses system tests and structure frameworks into an inside and out masterminded plan of steps that results in the viable improvement of graphical depiction. The testing system must collaborate test organizing, test arrangement, test execution, and the resultant data combination and evaluation. A procedure for programming testing must suit low-level tests that are essential to watch that a little source code segment has been precisely realized similarly as raised level tests that favour huge system limits against customer requirements

Chapter 6

RESULTS AND DISCUSSIONS

6.1 Efficiency of the Proposed System

The bilinear model allows the LSTM's to learn in parallel from scratch by achieving the performance of the other models such as the RNN with FC layer at a lower number of epochs. Since the outputs of the LSTM's in the bilinear model get summed up each of the LSTM's can converge fast. So compare to other models, this system is much versatile and board spectrum.

6.2 Comparison of Existing and Proposed System

Existing System is only able to generate a particular music with least variation. But in this proposed system, user can produce diverse music based on the input fed. The existing system doesn't produce any correspondence music, while this system does.

6.3 Advantages of the Proposed System

1. Easy to predict the yield before starting of the cultivation, this could give more advantage to the farmer to cultivate different crops.
2. Machine learning algorithms could help to improve agriculture in many aspects.

3. This system has high accuracy which could help to predict accurate results of the crop yield.

6.4 Sample Code

```
1 from fastai.learner import *
2 from subprocess import call
3 import os, argparse
4 import torchtext
5 from torchtext import vocab, data
6 from torchtext.datasets import language_modeling
7 from fastai.rnn_reg import *
8 from fastai.rnn_train import *
9 from fastai.nlp import *
10 from fastai.lm_rnn import *
11 from utils import *
12
13 import dill as pickle
14 import random
15 import numpy as np
16
17
18
19 def load_long_prompts(folder):
20     """ folder is either the path to train or to test
21         load_long_prompts loads all the files in that folder and returns
22         a list holding the text inside each file
23     """
24     prompts=[]
25     all_files=os.listdir(folder)
26     for i in range(len(all_files)):
27         f=open(folder/all_files[i])
28         prompt=f.read()
29         prompts.append(prompt)
30         f.close()
31     return prompts
32
33 def music_tokenizer(x): return x.split(" ")
```

```

34
35
36 def generate_musical_prompts(prompts, bptt, bs):
37     prompt_size=bptt
38     musical_prompts=[]
39
40     # Randomly select bs different prompts and hold them in musical_prompts
41     for i in range(bs):
42         this_prompt=[]
43         timeout=0
44         while timeout<100 and len(this_prompt)-prompt_size <=1:
45             sample=random.randint(0,len(prompts)-1)
46             this_prompt=prompts[sample].split(" ")
47             timeout+=1
48         assert len(this_prompt)-prompt_size >1, f'After 100 tries, unable to find
           prompt file longer than {bptt}. Run with smaller --bptt'
49
50         offset=random.randint(0, len(this_prompt)-prompt_size-1)
51         musical_prompts.append(" ".join(this_prompt[offset:prompt_size+offset]))
52
53     return musical_prompts
54
55
56 def create_generation_batch(model, num_words, random_choice_frequency,
57                             trunc_size, bs, bptt, prompts, params, TEXT):
58     """ Generate a batch of musical samples
59     Input:
60         model – pretrained generator model
61         num_words – number of steps to generate
62         random_choice_frequency – how often to pick a random choice rather than
           the top choice (range 0 to 1)
63         trunc_size – for the random choice, cut off the options to include only
           the best trunc_size guesses (range 1 to vocab_size)
64         bs – batch size – number of samples to generate
65         bptt – back prop through time – size of prompt
66         prompts – a list of training or test folder texts
67         params – parameters of the generator model
68         TEXT – holds vocab word to index dictionary
69

```

Output:

```
musical_prompts – the randomly selected prompts that were used to prime
                  the model (these are human-composed samples)
results – the generated samples
```

This is very loosely based on an example in the FastAI notebooks, but is modified to include randomized prompts, to generate a batch at a time rather than a single example, and to include truncated random sampling.

```
"""
```

```
musical_prompts=generate_musical_prompts(prompts, bptt, bs)
```

```
results=[ ' ']*bs
```

```
model.eval()
```

```
model.reset()
```

```
# Tokenize prompts and translate them to indices for input into model
```

```
s = [music_tokenizer(prompt)[:bptt] for prompt in musical_prompts]
```

```
t=TEXT.numericalize(s)
```

```
print("Prompting network")
```

```
# Feed the prompt one by one into the model (b is a vector of all the
indices for each prompt at a given timestep)
```

```
for b in t:
```

```
    res,*_ = model(b.unsqueeze(0))
```

```
print("Generating new sample")
```

```
for i in range(num_words):
```

```
    # res holds the probabilities the model predicted given the input
    sequence
```

```
    # n_tok is the number of tokens (ie the vocab size)
```

```
    [ps, n] =res.topk(params["n_tok"])
```

```
    # By default, choose the most likely word (choice 0) for the next
    timestep (for all the samples in the batch)
```

```
w=n[:,0]
```

```

102     # Cycle through the batch , randomly assign some of them to choose from
        the top trunc guesses , rather than to
103     # automatically take the top choice
104     for j in range(bs):
105         if random.random()<random_choice_frequency:
106             # Truncate to top trunc_size guesses only
107             ps=ps[:,trunc_size]
108             # Sample based on the probability the model predicted for those
                top choices
109             r=torch.multinomial(ps[j].exp(), 1)
110             # Translate this to an index
111             ind=to_np(r[0])[0]
112             if ind!=0:
113                 w[j].data[0]=n[j,ind].data[0]
114
115             # Translate the index back to a word (itos is index to string)
116             # Append to the ongoing sample
117             results[j]+=TEXT.vocab.itos[w[j].data[0]]+" "
118
119             # Feed all the predicted words from this timestep into the model , in
                order to get predictions for the next step
120             res,*_ = model(w.unsqueeze(0))
121     return musical_prompts , results
122
123 def main(model_to_load , training , test , train , gen_size , sample_freq , chordwise ,
124         note_offset , use_test_prompt , output_folder , generator_bs , trunc ,
                random_freq , prompt_size):
125
126     PATHS=create_paths()
127     print("Loading network")
128     lm,params,TEXT=load_pretrained_model(model_to_load , PATHS, training ,
                generator_bs)
129     bptt=prompt_size if prompt_size else params["bptt"]
130
131     prompts=load_long_prompts(PATHS["data"]/test) if use_test_prompt else
        load_long_prompts(PATHS["data"]/train)
132     print("Preparing to generate a batch of "+str(generator_bs)+" samples.")
133     musical_prompts , results=create_generation_batch(model=lm.model , num_words=
        gen_size ,

```

```

134         bs=generator_bs , bptt=bptt ,
135         random_choice_frequency=
136             random_freq ,
137         trunc_size=trunc , prompts=
138             prompts ,
139         params=params , TEXT=TEXT)
140
141 # Create the output folder if it doesn't already exist
142 out=PATHS["output"]/output_folder
143 out.mkdir(parents=True , exist_ok=True)
144
145 # For each generated sample, write mid, mp3, wav, and txt files to the
146     output folder (as 1.mid, etc)
147 for i in range(len(results)):
148     write_mid_mp3_wav(results[i] , str(i).zfill(2)+".mid" , sample_freq ,
149         note_offset , out , chordwise)
150     fname=str(i)+".txt"
151     f=open(out/fname,"w")
152     f.write(results[i])
153     f.close()
154
155 # For each human-composed sample, write mid, mp3, and wav files to the
156     output folder (as prompt1.mid, etc)
157 for i in range(len(musical_prompts)):
158     write_mid_mp3_wav(musical_prompts[i] , "prompt"+str(i).zfill(2)+".mid" ,
159         sample_freq , note_offset , out , chordwise)
160
161 if __name__ == "__main__":
162     parser = argparse.ArgumentParser()
163     parser.add_argument("-model" , help="Trained model in ./data/models" ,
164         required=True)
165     parser.add_argument("-output" , help="Folder inside ./data/output for holding
166         generations" , required=True)
167
168     parser.add_argument("--training" , dest="training" , help="Trained level (
169         light , med , full , extra). Default: light")
170     parser.set_defaults(training="light")
171     parser.add_argument("--size" , dest="size" , help="Number of steps to generate
172         (default 2000)" , type=int)

```



```

163 parser.set_defaults(size=2000)
164 parser.add_argument("--bs", dest="bs", help="Batch size: # samples to
    generate (default 16)", type=int)
165 parser.set_defaults(bs=16)
166 parser.add_argument("--trunc", dest="trunc", help="Truncate guesses to top n
    (default 5)", type=int)
167 parser.set_defaults(trunc=5)
168 parser.add_argument("--random-freq", dest="random-freq", help="How
    frequently to sample random note (0-1, default .5)", type=float)
169 parser.set_defaults(random-freq=.5)
170 parser.add_argument("--sample-freq", dest="sample-freq", help="Split beat
    into 4 or 12 parts (default 4 for Chordwise, 12 for Notewise)", type=int)
171 parser.add_argument("--chordwise", dest="chordwise", action="store_true",
    help="Use chordwise encoding (defaults to notewise)")
172 parser.set_defaults(chordwise=False)
173 parser.add_argument("--small-note-range", dest="small-note-range", action="
    store_true", help="Set 38 note range (defaults to 62)")
174 parser.set_defaults(small-note-range=False)
175 parser.add_argument("--use-test-prompt", dest="use-test-prompt", action="
    store_true", help="Use prompt from validation set.")
176 parser.set_defaults(use-test-prompt=False)
177 parser.add_argument("--prompt-size", dest="prompt-size", help="Set prompt
    size (default is model bptt)", type=int)
178 parser.add_argument("--test", dest="test", help="Specify folder name in data
    that holds test data (default 'test')")
179 parser.add_argument("--train", dest="train", help="Specify folder name in
    data that holds train data (default 'train')")
180 args = parser.parse_args()
181
182 if args.sample_freq is None:
183     sample_freq=4 if args.chordwise else 12
184 else:
185     sample_freq=args.sample_freq
186
187 note_offset=45 if args.small_note_range else 33
188
189 random.seed(os.urandom(10))
190
191 test = args.test if args.test else "test"

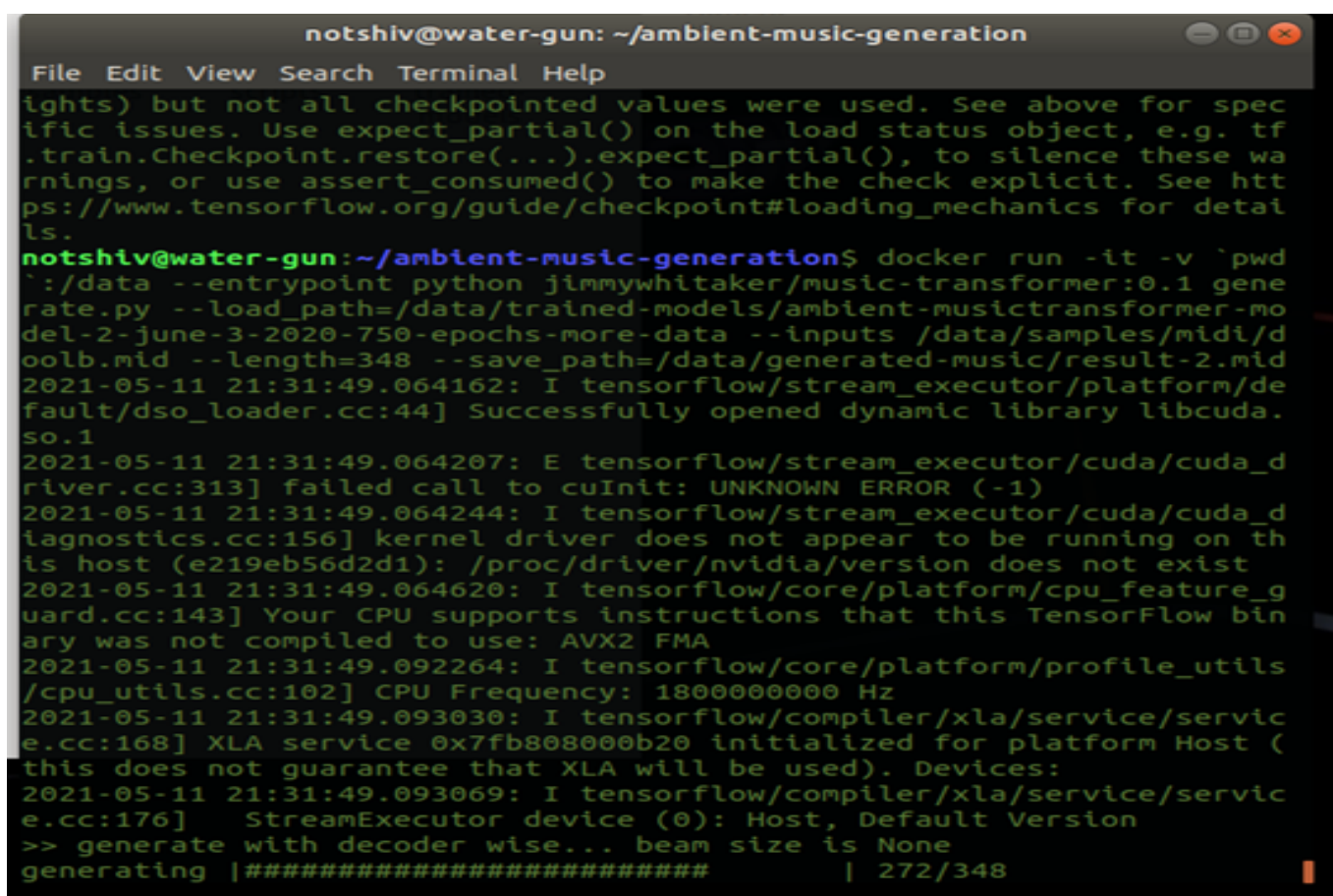
```

```

192     train = args.train if args.train else "train"
193
194     main(args.model, args.training, test, train, args.size, sample_freq, args.
195         chordwise,
196         note_offset, args.use_test_prompt, args.output, args.bs,
197         args.trunc, args.random_freq, args.prompt_size)

```

Output



```

notshiv@water-gun: ~/ambient-music-generation
File Edit View Search Terminal Help
ights) but not all checkpointed values were used. See above for specific issues. Use expect_partial() on the load status object, e.g. tf.train.Checkpoint.restore(...).expect_partial(), to silence these warnings, or use assert_consumed() to make the check explicit. See https://www.tensorflow.org/guide/checkpoint#loading_mechanics for details.
notshiv@water-gun:~/ambient-music-generation$ docker run -it -v `pwd`:/data --entrypoint python jimmywhitaker/music-transformer:0.1 generate.py --load_path=/data/trained-models/ambient-musictransformer-model-2-june-3-2020-750-epochs-more-data --inputs /data/samples/midi/doolb.mid --length=348 --save_path=/data/generated-music/result-2.mid
2021-05-11 21:31:49.064162: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library libcudaso.1
2021-05-11 21:31:49.064207: E tensorflow/stream_executor/cuda/cuda_driver.cc:313] failed call to cuInit: UNKNOWN ERROR (-1)
2021-05-11 21:31:49.064244: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:156] kernel driver does not appear to be running on this host (e219eb56d2d1): /proc/driver/nvidia/version does not exist
2021-05-11 21:31:49.064620: I tensorflow/core/platform/cpu_feature_guard.cc:143] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
2021-05-11 21:31:49.092264: I tensorflow/core/platform/profile_utils/cpu_utils.cc:102] CPU Frequency: 1800000000 Hz
2021-05-11 21:31:49.093030: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x7fb808000b20 initialized for platform Host (this does not guarantee that XLA will be used). Devices:
2021-05-11 21:31:49.093069: I tensorflow/compiler/xla/service/service.cc:176] StreamExecutor device (0): Host, Default Version
>> generate with decoder wise... beam size is None
generating |#####| 272/348

```

Figure 6.1: Results of sample code execution

Chapter 7

CONCLUSION AND FUTURE ENHANCEMENTS

7.1 Conclusion

The models studied showed improved performance in the quality of music over the base model and provides more insights into the architectures that would suit raw audio representations. The bilinear architecture and the LSTM with 2D convolutional layers produced the best audio of the tested models. This proposed project would help non-professional to easily get started with cinematic sound over and music direction. This would make the sound direction much more convenient and would save a significant amount of cost in sound direction. This technology would allow people to experiment and discover a new genre of music and sound over.

7.2 Future Enhancements

1. The proposed project is still not the fastest it could have been, so we aim to work on it speed of production in future.
2. In future, we would like to investigate the option of changing the loss function from the MSE to using an adversarial network and allow the network to model its own loss function based on the available data, one that can better represent how humans perceive music.

Chapter 8

PLAGIARISM REPORT



PLAGIARISM SCAN REPORT

Words 7753 Date June 03,2021

18%
Plagiarism

82%
Unique

Content Checked For Plagiarism

In this project we are going to implement a machine learning model which maps random sounds to musical notes which is further integrated in an harmonious ways according to the trained data set in the pipelining model. This ML model make uses of Pachyderm to handle the full pipeline for scaling and management. pachyderm makes it supremely simple to string together a bunch of loosely coupled frameworks into a smoothly scaling AI training platform. If you can package up your program in a Docker container you can easily run it in Pachyderm. This model takes a random noise note as input in mp3 format, which is further transformed to different formats for the machine learning model to recognize. This pipeline model is scaled according to the size of input data fed. This project will be consisting of a tool-box formatted container with the transformer model as the backend, where the user can feed the data and covert the noise notes into musical notes and gets an ambient music clip as output. Coming to the data set, we will be training the above-mentioned pipelining model with various noise notes which have been extracted from the data set. In here, each module will be commuting with other using JSON files. We will be integrating this whole file in a Docker container for an efficient implementation and use for the user.

Sources

Similarity

[The Musician in the Machine](https://magenta.tensorflow.org/musician-in-the-machine)

- If you can package up your program in a Docker container you can easily run it in Pachyderm. I created a complete walkthrough that you can check out right here on ...

<https://magenta.tensorflow.org/musician-in-the-machine>

25%

Chapter 9

SOURCE CODE & POSTER PRESENTATION

9.1 Source Code

```
1 In [1]:
2 import os
3 import json
4 import numpy as np
5 import pandas as pd
6 from keras.models import Sequential
7 from keras.layers import LSTM, Dropout, TimeDistributed, Dense, Activation,
   Embedding
8 C:\Users\GauravP\Anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarning:
   Conversion of the second argument of issubdtype from 'float' to 'np.floating
   ' is deprecated. In future, it will be treated as 'np.float64 == np.dtype(
   float).type'.
9 from ..conv import register_converters as _register_converters
10 Using TensorFlow backend.
11 In [2]:
12 data_directory = "../Data2/"
13 data_file = "Data_Tunes.txt"
14 charIndex_json = "char_to_index.json"
15 model_weights_directory = '../Data2/Model_Weights/'
16 BATCH_SIZE = 16
17 SEQ_LENGTH = 64
18 In [3]:
```

```

19 def read_batches(all_chars , unique_chars):
20     length = all_chars.shape[0]
21     batch_chars = int(length / BATCH_SIZE) #155222/16 = 9701
22
23     for start in range(0, batch_chars - SEQ_LENGTH, 64): # (0, 9637, 64) # it
24         # denotes number of batches. It runs everytime when
25         # new batch is created. We have a total of 151 batches.
26         X = np.zeros((BATCH_SIZE, SEQ_LENGTH)) # (16, 64)
27         Y = np.zeros((BATCH_SIZE, SEQ_LENGTH, unique_chars)) # (16, 64, 87)
28         for batch_index in range(0, 16): # it denotes each row in a batch.
29             for i in range(0, 64): # it denotes each column in a batch. Each
30                 column represents each character means
31                 # each time-step character in a sequence.
32                 X[batch_index , i] = all_chars[batch_index * batch_chars + start
33                     + i]
34                 Y[batch_index , i, all_chars[batch_index * batch_chars + start +
35                     i + 1]] = 1 # here we have added '1' because the
36                 # correct label will be the next character in the sequence. So,
37                 the next character will be denoted by
38                 # all_chars[batch_index * batch_chars + start + i] + 1.
39             yield X, Y
40
41 In [7]:
42 def built_model(batch_size , seq_length , unique_chars):
43     model = Sequential()
44
45     model.add(Embedding(input_dim = unique_chars , output_dim = 512,
46         batch_input_shape = (batch_size , seq_length), name = "embd_1"))
47
48     model.add(LSTM(256, return_sequences = True, stateful = True, name = "
49         lstm_first"))
50     model.add(Dropout(0.2 , name = "drp_1"))
51
52     model.add(LSTM(256, return_sequences = True, stateful = True))
53     model.add(Dropout(0.2))
54
55     model.add(LSTM(256, return_sequences = True, stateful = True))
56     model.add(Dropout(0.2))
57
58     model.add(TimeDistributed(Dense(unique_chars)))

```

```

51     model.add(Activation("softmax"))
52
53     model.load_weights("../Data/Model_Weights/Weights_80.h5", by_name = True)
54
55     return model
56 In [8]:
57 def training_model(data, epochs = 90):
58     #mapping character to index
59     char_to_index = {ch: i for (i, ch) in enumerate(sorted(list(set(data))))}
60     print("Number of unique characters in our whole tunes database = {}".format(
        len(char_to_index))) #87
61
62     with open(os.path.join(data_directory, charIndex_json), mode = "w") as f:
63         json.dump(char_to_index, f)
64
65     index_to_char = {i: ch for (ch, i) in char_to_index.items()}
66     unique_chars = len(char_to_index)
67
68     model = built_model(BATCH_SIZE, SEQ_LENGTH, unique_chars)
69     model.summary()
70     model.compile(loss = "categorical_crossentropy", optimizer = "adam", metrics
        = ["accuracy"])
71
72     all_characters = np.asarray([char_to_index[c] for c in data], dtype = np.
        int32)
73     print("Total number of characters = "+str(all_characters.shape[0])) #155222
74
75     epoch_number, loss, accuracy = [], [], []
76
77     for epoch in range(epochs):
78         print("Epoch {}/{}".format(epoch+1, epochs))
79         final_epoch_loss, final_epoch_accuracy = 0, 0
80         epoch_number.append(epoch+1)
81
82         for i, (x, y) in enumerate(read_batches(all_characters, unique_chars)):
83             final_epoch_loss, final_epoch_accuracy = model.train_on_batch(x, y)
            #check documentation of train_on_batch here: https://keras.io/
            models/sequential/

```

```

84         print("Batch: {}, Loss: {}, Accuracy: {}".format(i+1,
            final_epoch_loss , final_epoch_accuracy))
85         #here , above we are reading the batches one-by-one and train our
            model on each batch one-by-one.
86     loss.append(final_epoch_loss)
87     accuracy.append(final_epoch_accuracy)
88
89     #saving weights after every 10 epochs
90     if (epoch + 1) % 10 == 0:
91         if not os.path.exists(model_weights_directory):
92             os.makedirs(model_weights_directory)
93         model.save_weights(os.path.join(model_weights_directory , "Weights_
            {}.h5".format(epoch+1)))
94         print('Saved Weights at epoch {} to file Weights-{}.h5'.format(epoch
            +1, epoch+1))
95
96     #creating dataframe and record all the losses and accuracies at each epoch
97     log_frame = pd.DataFrame(columns = ["Epoch", "Loss", "Accuracy"])
98     log_frame["Epoch"] = epoch_number
99     log_frame["Loss"] = loss
100    log_frame["Accuracy"] = accuracy
101    log_frame.to_csv("../Data2/log.csv", index = False)
102 In [14]:
103 file = open(os.path.join(data_directory , data_file), mode = 'r')
104 data = file.read()
105 file.close()
106 if __name__ == "__main__":
107     training_model(data)
108 In [10]:
109 log = pd.read_csv(os.path.join(data_directory , "log.csv"))
110 log
111 In [1]:
112 import os
113 import json
114 import numpy as np
115 import pandas as pd
116 from keras.models import Sequential
117 from keras.layers import LSTM, Dropout , TimeDistributed , Dense , Activation ,
    Embedding

```



```

118 C:\Users\GauravP\Anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarning:
    Conversion of the second argument of issubdtype from 'float' to 'np.floating'
    is deprecated. In future, it will be treated as 'np.float64 == np.dtype(
    float).type'.
119 from ..conv import register_converters as _register_converters
120 Using TensorFlow backend.
121 In [3]:
122 data_directory = "../Data/"
123 data_file = "Data_Tunes.txt"
124 charIndex_json = "char_to_index.json"
125 model_weights_directory = '../Data/Model_Weights/'
126 BATCH_SIZE = 16
127 SEQ_LENGTH = 64
128 In [4]:
129 def read_batches(all_chars, unique_chars):
130     length = all_chars.shape[0]
131     batch_chars = int(length / BATCH_SIZE) #155222/16 = 9701
132
133     for start in range(0, batch_chars - SEQ_LENGTH, 64): # (0, 9637, 64) # it
        denotes number of batches. It runs everytime when
134         #new batch is created. We have a total of 151 batches.
135         X = np.zeros((BATCH_SIZE, SEQ_LENGTH)) # (16, 64)
136         Y = np.zeros((BATCH_SIZE, SEQ_LENGTH, unique_chars)) # (16, 64, 87)
137         for batch_index in range(0, 16): # it denotes each row in a batch.
138             for i in range(0, 64): # it denotes each column in a batch. Each
                column represents each character means
139                 #each time-step character in a sequence.
140                 X[batch_index, i] = all_chars[batch_index * batch_chars + start
                    + i]
141                 Y[batch_index, i, all_chars[batch_index * batch_chars + start +
                    i + 1]] = 1 # here we have added '1' because the
142                 #correct label will be the next character in the sequence. So,
                    the next character will be denoted by
143                 #all_chars[batch_index * batch_chars + start + i + 1]
144             yield X, Y
145 In [8]:
146 def built_model(batch_size, seq_length, unique_chars):
147     model = Sequential()
148

```

```

149     model.add(Embedding(input_dim = unique_chars , output_dim = 512,
150                          batch_input_shape = (batch_size , seq_length)))
151
152     model.add(LSTM(256, return_sequences = True , stateful = True))
153     model.add(Dropout(0.2))
154
155     model.add(LSTM(128, return_sequences = True , stateful = True))
156     model.add(Dropout(0.2))
157
158     model.add(TimeDistributed(Dense(unique_chars)))
159
160     model.add(Activation("softmax"))
161
162     return model
163
164 In [13]:
165 def training_model(data , epochs = 80):
166     #mapping character to index
167     char_to_index = {ch: i for (i, ch) in enumerate(sorted(list(set(data))))}
168     print("Number of unique characters in our whole tunes database = {}".format(
169         len(char_to_index))) #87
170
171     with open(os.path.join(data_directory , charIndex_json), mode = "w") as f:
172         json.dump(char_to_index , f)
173
174     index_to_char = {i: ch for (ch, i) in char_to_index.items()}
175     unique_chars = len(char_to_index)
176
177     model = built_model(BATCH_SIZE, SEQ_LENGTH, unique_chars)
178     model.summary()
179     model.compile(loss = "categorical_crossentropy", optimizer = "adam", metrics
180                  = ["accuracy"])
181
182     all_characters = np.asarray([char_to_index[c] for c in data], dtype = np.
183                                int32)
184     print("Total number of characters = "+str(all_characters.shape[0])) #155222
185
186     epoch_number , loss , accuracy = [], [], []
187
188     for epoch in range(epochs):

```

```

184     print("Epoch {}/{}".format(epoch+1, epochs))
185     final_epoch_loss , final_epoch_accuracy = 0, 0
186     epoch_number.append(epoch+1)
187
188     for i, (x, y) in enumerate(read_batches(all_characters , unique_chars)):
189         final_epoch_loss , final_epoch_accuracy = model.train_on_batch(x, y)
190         #check documentation of train_on_batch here: https://keras.io/
191         #here, above we are reading the batches one-by-one and train our
192         #model on each batch one-by-one.
193         print("Batch: {}, Loss: {}, Accuracy: {}".format(i+1,
194             final_epoch_loss , final_epoch_accuracy))
195         loss.append(final_epoch_loss)
196         accuracy.append(final_epoch_accuracy)
197
198     #saving weights after every 10 epochs
199     if (epoch + 1) % 10 == 0:
200         if not os.path.exists(model_weights_directory):
201             os.makedirs(model_weights_directory)
202         model.save_weights(os.path.join(model_weights_directory , "Weights_
203             {}.h5".format(epoch+1)))
204         print('Saved Weights at epoch {} to file Weights_{}.h5'.format(epoch
205             +1, epoch+1))
206
207     #creating dataframe and record all the losses and accuracies at each epoch
208     log_frame = pd.DataFrame(columns = ["Epoch", "Loss", "Accuracy"])
209     log_frame["Epoch"] = epoch_number
210     log_frame["Loss"] = loss
211     log_frame["Accuracy"] = accuracy
212     log_frame.to_csv("../Data/log.csv", index = False)
213
214 In [5]:
215 file = open(os.path.join(data_directory , data_file), mode = 'r')
216 data = file.read()
217 file.close()
218 if __name__ == "__main__":
219     training_model(data)
220
221 In [4]:
222 log = pd.read_csv(os.path.join(data_directory , "log.csv"))
223 log

```

9.2 Poster Presentation

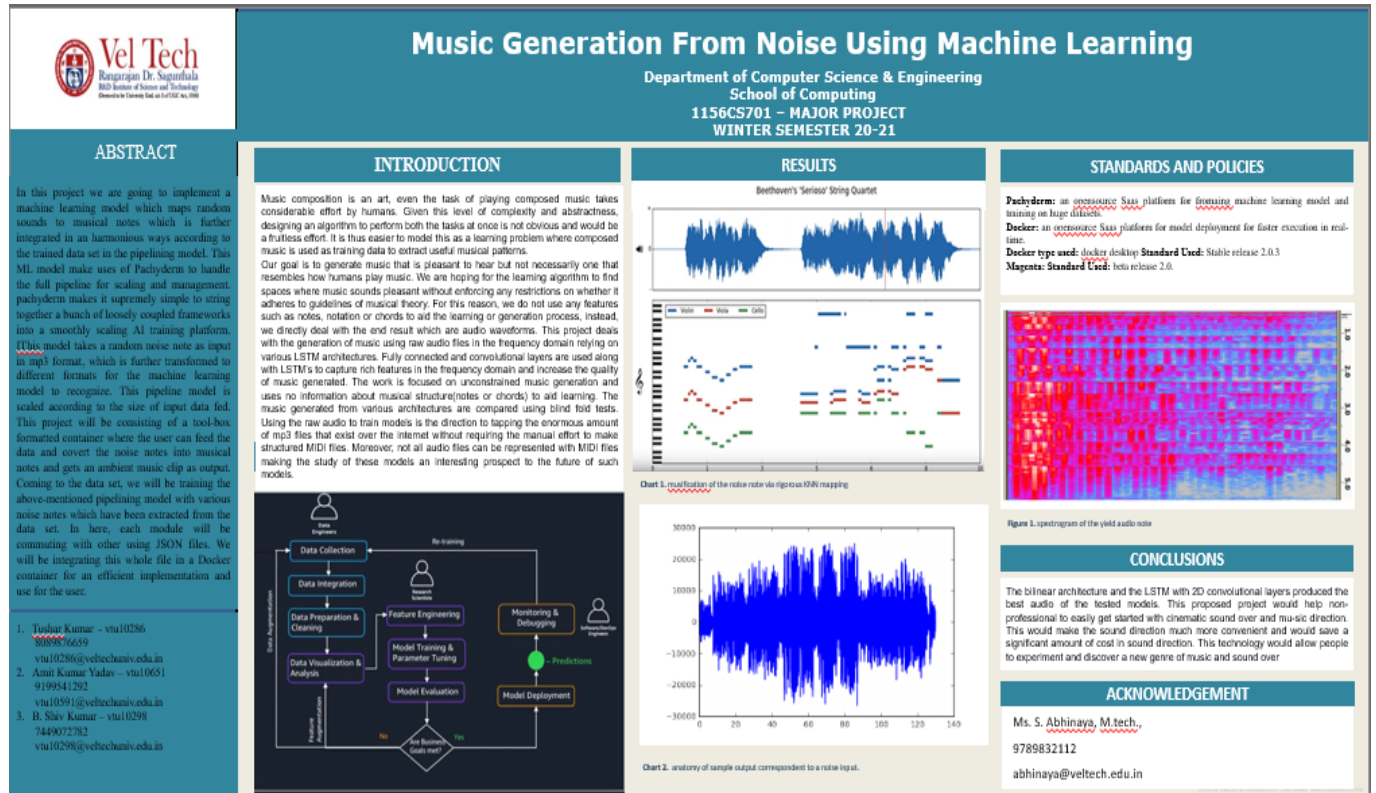


Figure 9.1: Poster Presentation

References

- [1] Nikhil Kotecha, "Bach2Bach: Generating Music Using A Deep Reinforcement Learning Approach", 2017.
- [2] Jean-Pierre Briot, Gaetan Hadjeres and Francois-David Pachet, "Deep Learning Techniques for Music Generation – A Survey", 2019.
- [3] Srikanth Grandhe , "Music Generation Using Deep Learning", University of Massachusetts Amherst Amherst, Massachusetts , 2016.
- [4] Allen Huang, Raymond Wu, "Deep Learning for Music", Department of Management Science and Engineering Stanford University, 2019.
- [5] Davin Hüssel, "Reinforcement Approach to Music Generation" 2018.
- [6] Vasanth Kalingeri , "Deep Learning Techniques for Music generation", 2016.
- [7] Nabil Hewahi, Salman AlSaigal Sulaiman AlJanahi , "Generation of Music pieces using Machine Learning: long short-term memory neural networks approach ", 2019.
- [8] Jean-Pierre Briot, "Deep Learning Techniques for Music Generation (8)", Programa de Pós-Graduação em Informática, 2018.
- [9] Vitor Alves Arrais de Souza, Sandra Eliza Fontes de Avila, "Deep Neural Networks for Generating Music", 2018.
- [10] P.Shobha Rani, S.V.Praneeth, Vattigunta Revanth Kumar Reddy, M.J.Sathish, "Music Generation Using Deep Learning", 2020.