AUDIO SOURCE SEPARATION USING BI-DIRECTIONAL GATED RECURRENT UNIT

A Thesis in
Acoustics
by
Sanjay Majumder

© 2022 Sanjay Majumder

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

August 2022
The thesis of Sanjay Majumder was reviewed and approved by the following:

Karl Reichard  
Associate Research Professor  
Graduate Program in Acoustics  
Thesis Advisor

Tyler Dare  
Assistant Research Professor  
Graduate Program in Acoustics

Andrew Barnard  
Professor of Acoustics  
Director, Graduate Program in Acoustics
Abstract

In the world of signal processing, although audio source separation is not a new concept, to date, it has remained a fascinatingly complex task. Because of the vast field of practical application, over the years, researchers from varied backgrounds have deployed advanced and sophisticated algorithms of deep learning, signal processing, data augmentation, and computer listening to isolate individual voices or instruments from the audio mixtures in precision and clarity. Among all these new technologies, neural networks, especially recurrent neural networks (RNN), have promising evidence of optimal results in multimedia problems. However, a series of projects are still going on to give the outcomes more accuracy. This thesis aims to contribute to this field of research by introducing the Bi-directional Gated Recurrent Unit (Bi-GRU) - a newer version of RNN to separate audio stems from the audio mixture in the Time-Frequency domain.

The architecture of the GRU is robust yet simple to use compared to its predecessor Long Short Time Memory (LSTM), and most interestingly, it efficiently solves the problem of gradient exploding or gradient vanishing, which could previously result in data over-fitting and under-fitting, respectively. But as information only passes in the forward direction (left to right), both general RNN and GRU suffer from the lack of information from future cells. To resolve this issue, in this study, the bi-directionality feature of RNN has been exploited, which facilitates the accurate learning of the GRU from the previous as well as the future cells, producing a better result. The audio data are transformed into spectrograms, and the Bi-GRU model fetches the essential temporal and spectral information to train and test the system to separate four well-defined audio stems in a supervised manner. This newly developed source separation model is applied on the MUSDB18 [45] dataset to test, and the performance of the model is assessed by using the museval [61] evaluation toolbox and Mean Opinion Score (MOS). The measured performance is then compared with the other known model’s performance. In addition, this thesis provides a detailed survey of the audio source separation work, and at the end of this paper, some observations and shortcomings of the system are discussed.
3.2.6 Loss Function .................................................. 20
3.2.7 Optimizer ...................................................... 21
3.2.8 Dropout ........................................................ 21
3.2.9 Epochs .......................................................... 22
3.2.10 Batch Size ..................................................... 22
3.3 Recurrent Neural Network ...................................... 23
3.3.1 Simple Recurrent Neural Network ............................. 24
3.3.2 Long Short Time Memory ..................................... 25
3.3.3 Gated Recurrent Unit ......................................... 27
3.3.4 Bi-directionality of Recurrent Neural Network .............. 29
3.4 Sound and Audio .................................................. 30
3.4.1 Audio Formats .................................................. 31
3.4.2 Multi-track Audio and Audio Stems ........................... 31
3.5 Audio Representation ............................................. 32
3.5.1 Time Domain, Frequency Domain and Time-Frequency Domain 33
3.5.2 Short-Time Fourier Transformation ............................. 34
3.6 Dataset .......................................................... 36

Chapter 4
Implementation of Bi-directional GRU for Audio Source Separation 38
4.1 Dataset .......................................................... 38
4.2 Processing the Dataset ........................................... 39
4.2.1 Dataset Processing ............................................. 39
4.2.2 Audio Data Processing ........................................ 40
4.2.2.1 Data Augmentation ........................................ 41
4.2.2.2 Spectrogram and Tensor ................................... 43
4.2.2.3 Flatten the Input Tensor .................................... 45
4.3 Architecture of the Bi-directional Gated Recurrent Unit ....... 46
4.3.1 Implementing Gated Recurrent Unit ............................ 48
4.3.2 Implementing Bidirectionality ................................... 48
4.3.3 Input Layer ..................................................... 49
4.3.4 Hidden Layers .................................................. 50
4.3.5 Dense Layers ................................................... 50
4.3.6 Output ........................................................ 51
4.3.7 Combining the Bi-directional GRU networks .................. 51
4.3.8 Hyper-parameters ............................................. 52
4.4 Training ........................................................ 53
4.5 Testing .......................................................... 53
4.6 Architecture of the proposed system ............................. 54

Chapter 5
Results ................................................................. 56
5.1 Training and Validation Loss ..................................... 56
5.2 Comparing waveform, spectrogram and Energy ................ 57
# List of Figures

3.1 A Simple Neural Network with one hidden layer .................. 17  
3.2 Activities within a Neuron ........................................... 18  
3.3 Sigmoid Activation Function ....................................... 19  
3.4 Forward and Backward Propagation of a Neural Network .......... 20  
3.5 Dropout Mechanism in Neural Network [57] ......................... 21  
3.6 Internal Architecture of a Simple RNN cell ....................... 24  
3.7 Internal Architecture of a LSTM cell ............................. 26  
3.8 Internal Architecture of a GRU cell .............................. 28  
3.9 Uni-directionality of Recurrent Neural Network ................ 30  
3.10 Bi-directionality of Recurrent Neural Network ................. 30  
3.11 Snapshots of file system of audio multi-track .................. 32  
3.12 Snapshots of file system of audio stems ......................... 32  
3.13 Time-domain, Frequency-domain and Time-Frequency domain conversion 33  
3.14 Short-Time Fourier Transformation on Whole Audio .............. 35  
3.15 Short-Time Fourier Transformation on Shorter Segments of Audio . 36  
4.1 MusDB18 Dataset in .stem.mp4 format ............................. 39
4.2 File System in a song folder after uncompressing it from .stem.mp4 format 40
4.3 Two Channel (Stereo) Audio to Tensor ................................. 41
4.4 Waveform and Spectrogram of Whole Audio .......................... 42
4.5 Audio Segmentation in five equal length audio segment .......... 43
4.6 Waveform and Spectrogram of a Single Audio Chunk ............ 44
4.7 Spectrogram to Tensor Conversion .................................... 45
4.8 Tensor to Flatten Tensor ............................................. 46
4.9 Cell Architecture of Bi-directional Gated Recurrent Unit .......... 47
4.10 Proposed Network Architecture ....................................... 50
4.11 Output Tensor .......................................................... 51
4.12 Architecture of the proposed system. ............................... 55
5.1 Training loss and validation loss of the proposed system .......... 57
5.2 Comparing the Spectrogram of Drums Stem (a) original and (b) separated of Singer/Songwriter genre music. ......................... 58
5.3 Comparing the waveform of original (a) and separated (b) Drums Stem of Singer/Songwriter genre music. ............................ 59
5.4 Comparing the Spectrogram of Vocals Stem (a) original and (b) separated of Singer/Songwriter genre music. ......................... 60
5.5 Comparing the waveform of original (a) and separated (b) Vocals Stem of Singer/Songwriter genre music. ............................ 60
5.6 Comparing the Spectrogram of Bass Stem (a) original and (b) separated of Singer/Songwriter genre music. ............................ 61
5.7 Comparing the waveform of original (a) and separated (b) Bass Stem of Singer/Songwriter genre music. ............................ 61
5.8 Comparing the Spectrogram of Rest of the accompaniment Stem (a) original and (b) separated of Singer/Songwriter genre music. ......... 62

5.9 Comparing the waveform of original (a) and separated (b) Rest of the accompaniment Stem of Singer/Songwriter genre music. .......... 62

5.10 Comparing the RMS energy of the original and separated audio mixture of Singer/Songwriter genre music. ............................. 63

5.11 Comparing the Spectrogram of Drums Stem (a) original and (b) separated of Rock genre music. .......................... 64

5.12 Comparing the waveform of original (a) and separated (b) Drums Stem of Rock genre music. .......................... 64

5.13 Comparing the Spectrogram of Vocals Stem (a) original and (b) separated of Rock genre music. .......................... 65

5.14 Comparing the waveform of original (a) and separated (b) Vocals Stem of Rock genre music. .......................... 65

5.15 Comparing the Spectrogram of Bass Stem (a) original and (b) separated of Rock genre music. .......................... 66

5.16 Comparing the waveform of original (a) and separated (b) Bass Stem of Rock genre music. .......................... 66

5.17 Comparing the Spectrogram of Rest of the Accompaniment Stem (a) original and (b) separated of Rock genre music. ................. 67

5.18 Comparing the waveform of original (a) and separated (b) Rest of the Accompaniment Stem of Rock genre music. ..................... 68

5.19 Comparing the RMS energy of the Original and the separated audio mixture of Rock genre music. ............................. 69

A.1 Comparing the spectrograms of Drums stem, (a) original and (b) separated of Jazz genre music. ............................. 80

A.2 Comparing the waveform of (a) original and (b) separated Drums Stem of Jazz genre music. ............................. 81
A.3 Comparing the Spectrogram of Vocals Stem (a) original and (b) separated of Jazz genre music.  

A.4 Comparing the waveform of original (a) and separated (b) Vocals Stem of Jazz genre music.  

A.5 Comparing the Spectrogram of Bass Stem (a) original and (b) Separated of Jazz genre music.  

A.6 Comparing the waveform of original (a) and separated (b) Bass stem of of Jazz genre music.  

A.7 Comparing the Spectrogram of Rest of the accompaniment Stem (a) original and (b) separated of Jazz genre music.  

A.8 Comparing the waveform of original (a) and separated (b) Rest of the accompaniment Stem of Jazz genre music.  

A.9 Comparing the RMS energy of the original and separated audio mixture of Jazz genre music.  

A.10 Comparing the Spectrogram of Drums Stem (a) original and (b) separated of Electronic genre music.  

A.11 Comparing the waveform of original (a) and separated (b) Drums Stem of Electronic genre music.  

A.12 Comparing the Spectrogram of Vocals Stem (a) original and (b) separated of Electronic genre music.  

A.13 Comparing the waveform of original (a) and separated (b) Vocals Stem of Electronic genre music.  

A.14 Comparing the Spectrogram of Bass Stem (a) original and (b) separated of Electronic genre music.  

A.15 Comparing the waveform of original (a) and separated (b) Bass Stem of Electronic genre music.  

A.16 Comparing the Spectrogram of Rest of the accompaniment Stem (a) original and (b) separated of Electronic genre music.  

xii
A.17 Comparing the waveform of original (a) and separated (b) Rest of the accompaniment Stem of Electronic genre music. .............................. 90

A.18 Comparing the RMS energy of the original and separated audio mixture of Electronic genre music. ......................................................... 91

A.19 Comparing the Spectrogram of Drums Stem (a) original and (b) separated of Pop/Rock genre music. ......................................................... 92

A.20 Comparing the waveform of original (a) and separated (b) Drums Stem of Pop/Rock genre music. ......................................................... 92

A.21 Comparing the Spectrogram of Vocals Stem (a) original and (b) separated of Pop/Rock genre music. ......................................................... 92

A.22 Comparing the waveform of original (a) and separated (b) Vocals Stem of Pop/Rock genre music. ......................................................... 93

A.23 Comparing the Spectrogram of Bass Stem (a) original and (b) separated of Pop/Rock genre music. ......................................................... 94

A.24 Comparing the waveform of original (a) and separated (b) Bass Stem of Pop/Rock genre music. ......................................................... 94

A.25 Comparing the Spectrogram of Rest of the accompaniment Stem (a) original and (b) separated of Pop/Rock genre music. ......................... 95

A.26 Comparing the waveform of original (a) and separated (b) Rest of the accompaniment Stem of Pop/Rock genre music. ......................... 95

A.27 Comparing the RMS energy of the original and separated audio mixture of Pop/Rock genre music. ......................................................... 96

A.28 Comparing the Spectrogram of Drums Stem (a) original and (b) separated of Reggae genre music. ......................................................... 97

A.29 Comparing the waveform of original (a) and separated (b) Drums Stem of Reggae genre music. ......................................................... 97

A.30 Comparing the Spectrogram of Vocals Stem (a) original and (b) separated of Reggae genre music. ......................................................... 98
A.31 Comparing the waveform of original (a) and separated (b) Vocals Stem of Reggae genre music. ................................. 98
A.32 Comparing the Spectrogram of Bass Stem (a) original and (b) separated of Reggae genre music. ........................................ 99
A.33 Comparing the waveform of original (a) and separated (b) Bass Stem of Reggae genre music. ........................................ 99
A.34 Comparing the Spectrogram of Rest of the accompaniment Stem (a) original and (b) separated of Reggae genre music. .................................................. 100
A.35 Comparing the waveform of original (a) and separated (b) Rest of the accompaniment Stem of Reggae genre music. .................................................. 100
A.36 Comparing the RMS energy of the Original and separated audio mixture of Reggae genre music. ........................................ 101
A.37 Comparing the Spectrogram of Drums Stem (a) original and (b) Separated 102
A.38 Comparing the waveform of original (a) and Separated (b) Drums Stem . 102
A.39 Comparing the Spectrogram of Vocals Stem (a) original and (b) Separated 103
A.40 Comparing the waveform of original (a) and Separated (b) Vocals Stem . 103
A.41 Comparing the Spectrogram of Bass Stem (a) original and (b) Separated 104
A.42 Comparing the waveform of original (a) and Separated (b) Bass Stem . 105
A.43 Comparing the Spectrogram of Rest of the accompaniment Stem (a) original and (b) Separated 105
A.44 Comparing the waveform of original (a) and Separated (b) Rest of the accompaniment Stem. 106
A.45 Comparing the RMS energy of the Original and Separated Audio Mixture. 107
A.46 Comparing the Spectrogram of Drums Stem (a) original and (b) Separated 108
A.47 Comparing the waveform of original (a) and Separated (b) Drums Stem . 108
A.48 Comparing the Spectrogram of Vocals Stem (a) original and (b) Separated 108
A.65 Comparing the waveform of original (a) and Separated (b) Drums Stem. 118
A.66 Comparing the Spectrogram of Vocals Stem (a) original and (b) Separated. 119
A.67 Comparing the waveform of original (a) and Separated (b) Vocals Stem. 119
A.68 Comparing the Spectrogram of Bass Stem (a) original and (b) Separated. 120
A.69 Comparing the waveform of original (a) and Separated (b) Bass Stem. 120
A.70 Comparing the Spectrogram of Rest of the accompaniment Stem (a) original and (b) Separated. 121
A.71 Comparing the waveform of original (a) and Separated (b) Rest of the accompaniment Stem. 121
A.72 Comparing the RMS energy of the Original and Separated Audio Mixture. 122

B.1 Sampling Frequency Discrepancy in librosa library 123
B.2 Spectrogram with wrong sampling frequency 124
B.3 Spectrogram with actual sampling frequency 125
B.4 Effect of wrong tensor size selection 126

C.1 Human Research Determination 129
C.2 Communication with IRB Analyst 130
D.1 Datasheet used to collect MOS score. 131
# List of Tables

4.1 Mapping of the separate audio stems encoded in a single audio .stem.mp4 format in MUSDB18. ................................................................. 40

5.1 Evaluation of Separated audio quality of the proposed system on the MUSDB18 test dataset by MUSEVAL ........................................... 71

5.2 Source to Distortion ratio of separated audio stems on MUSDB18 by the proposed model and other reference models .......................... 72

5.3 Mean Opinion Score (MOS) of the separated audio stems from MUSDB18 dataset. ................................................................. 74

5.4 Comparison Mean Opinion Score (MOS) of the proposed model and other audio source separation model. ................................. 74

5.5 Mean Opinion Score (MOS) of the separated audio stems from Youtube. 75
List of Symbols

AI   Artificial Intelligence
Bi-GRU Bi-directional Gated Recurrent Unit
BSS  Blind Source Separation
CASA Computational Auditory Scene Analysis
CNN  Convolution Neural Network
CPU  Central Processing Unit
dB   Decibels
DFT  Discrete Fourier Transformation
DNN  Deep Neural Network
DRNN Deep Recurrent Neural Network
EM   Expectation-Maximization
FT   Fourier Transformation
FFT  Fast Fourier Transformation
GMM  Gaussian Mixture Model
GNN  Gated Neural Network
GPU  Graphical Processing Unit
GRU  Gated Recurrent Unit
ICA  Independent Component Analysis
iSTFT Inverse Short-Time Fourier Transformation
LFPC Log-Frequency Power Coefficients
LLE  Locally Linear Embedding
LSTM  Long-Short Time Memory
MFCC  Mel-Frequency Cepstrum Coefficient
MIR   Music Information Retrieval
ML    Machine Learning
MOS   Mean Opinion Score
NMF   Non-negative Matrix Factorization
PCA   Principal Component Analysis
PLP   Perceptual Linear Predictive Classifier
RNN   Recurrent Neural Network
SAR   Source to Artifact Ratio
SD    Sparse Decomposition
SDR   Source to Distortion Ratio
SNR   Source to Noise Ratio
SVM   Support Vector Machines
STFT  Short-Time Fourier Transformation
TF    Time-Frequency
Acknowledgments

Many people have helped me during my time at Penn State to whom I owe my gratitude. Thanks to my advisor Dr. Karl Reichard, who has been an exceptional mentor throughout the thesis. Thanks to members of the committee, Dr. Andrew Barnard, and Dr. Tyler Dare, for providing the necessary suggestion to improve the thesis.

I like to thank my professors at Penn State, particularly Dr. Victor Sparrow and Dr. Tom Gabrielson, for their unwavering passion for teaching and willingness to answer my endless questions. Thanks are also owed to Melissa Wandrisco, Lydia Olsen, and Kris Popovich for keeping the Graduate Program in Acoustics running.

I like to show my gratitude to my previous advisors, Dr. Benjamin D. Smith and Dr. Sanjay Chakraborty for their support to make this journey possible. I like to thank my guru Rony Chowdhury for his guidance and unconditional support throughout my life. I also want to thank my family and friends back in Bangladesh and India for checking in and making sure I had everything I needed to succeed in a new place nearly thousands of miles away from home. Thanks to Dr. Eshita Majumder for proofreading the thesis. I like to thank Morshadul Alam, Sujan N. Majumder, Popy Roy, Rajesh Roy, Dr. Rabi Narayan Behera, Anup Ghosh, Mili Ghosh, Rana Roy, Happy Roy, Sumana Chowdhury, Dr. Douglas Huntley, Sandra Lemons, and her family for supporting me all the time. Thanks to

Last but not least, special thanks to Shurjo, and Luna, for their unconditional support.
Audio Source separation is a one-to-many mapping problem in artificial intelligence where several audio sources need to be separated from the audio mixture. It is a part of Music Information Retrieval (MIR) that requires expertise in digital audio, audio signal processing, audio information retrieval, and artificial intelligence. The separation process is generally achieved by using time series data in the waveform domain directly or extracting features from audio data in the time and frequency domain or by converting the time series data into a time-frequency representation of the mixture signal such as the Short-Time Fourier Transform (STFT), commonly known as a spectrogram. After segregation, these musical tracks can be used individually or processed separately for other practical audio and acoustic applications. Audio source separation can have potential applications such as automatic music generation, beat tracking, chord mapping, music remastering, karaoke music, speaker identification, etc. As musical audio mixture contains the summation of the audio of different musical instruments, separating these sources is a challenging yet exciting task. Some critical factors for audio source separation are time complexity, quality of the separated audio, distortion, noise, and leakage of the separated audio.

1.1 Motivation

The recent research and innovation of deep learning algorithms have improved signal processing and the audio domain. Several pieces of research have been done so far to improve the shortcomings of the deep learning algorithms and have shown promising results. These newly developed deep learning algorithms are deployed in different source separation models, offering impressive separation quality. But the improvement of audio source separation is still an ongoing process. When a new technology is invented or
introduced, such as a new deep learning algorithm, researchers try to incorporate the technology into the audio source separation problem. In that way, new and improved audio source separation models are introduced into the field. Unfortunately, no source separation model has been submitted by using Bi-directional Gated Recurrent Unit (Bi-directional GRU) to date.

Several research works have shown that GRU, specially bi-directional GRU, outperforms Long Short Time Memory (LSTM). Some research works have been done to compare the efficiency of the GRU, bi-directional GRU, and LSTM in the multimedia domain, especially in the audio field. However, none of the research uses Bi-directional GRU solely in the audio source separation. Bi-directional GRU propagates information forward and backward in a single layer which solves the issue of accessing the memory of the future cells. This bidirectional property of a simple designed GRU is helpful to learning more accurately in a shorter time than other introduced Recurrent Neural Network (RNN) algorithms for audio source separation.

1.2 Context

This thesis builds upon the foundation laid by numerous researchers who worked on audio source separation, audio signal processing, and neural network, which provided a basic framework for separating audio sources (discussed elaborately in the Related work section in Chapter 2). Hybrid Demucs [15], KUIELab-MDX-Net [30], D3Net [65], Spleeter [23] models provided systems by which audio source separation can be performed. Although these models have provided a good platform for audio source separation, none still offer an excellent Source to Distortion ratio (SDR) for the separated audio stems, especially the rest of the accompaniment stem part. Moreover, some of these models used old Convolution Neural Network (CNN) and Recurrent Neural Network (RNN) algorithms such as unet, LSTM which are relatively old algorithms and a known cause of higher time complexity while performing the separation.

Technology, especially deep neural networks, is evolving continuously, and more sophisticated algorithms are introduced to overcome the previously referenced algorithm’s lackings. Bi-directional Gated Recurrent Unit is a relatively new RNN algorithm introduced to overcome the deficiencies of Long Short Time Memory (LSTM) algorithm. It is tested on the other kinds of multimedia problems, such as natural language processing, and yielded promising results in regard to accuracy.

Unfortunately, no audio source separation model has yet been proposed using bi-
directional GRU. The bi-directional GRU unit is successfully implemented for audio source separation for the first time in this project. Mean Opinion Score (MOS) is a subjective audio quality evaluation technique that evaluates the separated audio and the respective original audio quality. This thesis aims to discuss the aspect of audio source separation in detail, provide solutions for an improved model, and validate the same through the proposed model. Once implemented, this enhanced model version is used and applied to the dataset for getting separated audio. The quality of the separated audio is then evaluated using the source separation evaluation tool and Mean Opinion Score (MOS).

1.3 Objectives of the study

The purpose of audio source separation is to separate the audio signal instrument-wise from a polyphonic or monophonic audio mixture. This thesis attempts to improve separation quality by incorporating the latest technology of deep learning algorithms that have never been explored until now. Some common ideas and techniques are collected from different methods to propose a new framework for source separation. Bi-directional GRU has a simple architecture, and it requires fewer parameters to calculate than other RNN. So, the system requires fewer epochs and less training time than other proposed models. This study aims to find the efficiency of the Bi-directional Gated Recurrent Unit on audio source separation.

1.4 Overview of thesis organization

The first chapter established an introduction and motivation for the research work and laid out the primary objectives to be accomplished. Chapter two provides a survey on the work done by the researchers so far to separate audio from a source. Chapter three introduces the theoretical background required for the audio source separation problem. The fourth chapter talks about the proposed deep neural network architecture and how the improved model is applied to the MUSDB18 [45] dataset for training and validation purposes. Chapter five discusses the results of the enhanced model with the new terrain using new inputs and visitor responses. The final chapter, Chapter six, discusses conclusions and explores opportunities for future work in this project.
Chapter 2  
Related Work

Source separation is an empirical task in the field of signal processing. Music source separation is a trendy topic in the field of music technology. Many works have been done to separate music sources from music. Various researchers adopt different approaches to tackling music source separation. Some researchers divided the vocal and non-vocal parts; some extracted only the vocal part from the rest and separated all the sources. As the objective differs, the music source separation techniques also differ. This related work is written so that anyone who wants to perform audio source separation work can use it as a reference. The techniques used for audio source separation range from Independent Component Analysis (ICA), Principle Component Analysis (PCA), Blind Source Separation (BSS) algorithm, adaptive signal processing, Deep Neural Network (DNN), repeating structure analysis, non-negative matrix factorization (NMF), Sparse Decompositions (SD) and Computational Auditory Scene Analysis (CASA), etc. These works can be broadly categorized into three types. One way is to solve the problem only using signal processing algorithms. The second way is to combine signal processing and traditional machine learning algorithms. Finally, deep learning algorithms are gaining more popularity to overcome the deficiency of the machine learning process among researchers and are the third way to separate audio from the audio mixture. This thesis focuses on the three aspects of the music source separation: Data Dimensionality Reduction for music data, Music Source Separation, and enhancement of the separated sources.

2.1 Data dimensionality reduction

Due to the large size and multidimensionality of the audio data, dimensionality reduction techniques are popular in audio data pre-processing for music source separation.
The objective of the audio data dimensionality reduction is to reduce the file size and dimensionality of the audio data while preserving the critical features of the data. For audio research and development, uncompressed audio data is used. The popular form of uncompressed audio format is wave (.wav). For comparison, .wav format audio is 20 to 25 times larger than mp3 format for the same audio. The main burden of the audio data dimensionality reduction is losing the important features and information of the data, which can jeopardize the overall outcome of the audio source separation system.

Hinton and Salakhutdinov [24] reported a dimensionality reduction technique using an adaptive, multi-layer encoder for nonlinear generalization of PCA. The proposed decoder transforms high dimensional data into low dimensional data using a multi-layer neural network with a small central layer. Paper [58] evaluated the efficiency of Isometric Feature Mapping (ISOMAP), t-SNE (Stochastic Neighborhood Embedding), and PCA for the classification of audio data. The result showed that t-SNE provides better classification results than PCA and ISOMAP in low dimensions. ISOMAP, t-SNE, and PCA were used as the unsupervised learning algorithm. Wack et al. [70] compared the efficiency of unsupervised dimensionality reduction algorithm-PCA, ISOMAP, and LLE (Locally Linear Embedding) for music similarity finding problems. Before applying the data dimensionality reduction algorithm, it also evaluated the effect of three kinds of pre-processing data algorithms, namely normalization, using the box-cox transformation, and gaussianizing on each of the unsupervised dimensionality reduction algorithms. Another paper [10] investigated the efficiency of the supervised algorithm for dimensionality reduction. This paper reviewed the supervised PCA-based, Non-negative Matrix Factorization (NMF) based, and manifold-based dimension reduction methods and describes their usability in speech recognition. Suri and Bailis [62,63] proposed a dimensionality reduction optimizer named DROP, which used Workload-Aware Progressive Sampling. Due to its highly structured nature, time series allow aggressive sampling for dimension reduction, and with the help of progressive sampling, DROP exposed low-dimensional but high-quality data points.

### 2.2 Signal processing algorithms

Barry et al. [3] used a new approach -azimuth discrimination within the stereo field to separate music sources in real-time. The proposed algorithm uses an interaural intensity difference between left and right channels for a single source by taking advantage of the pan pot to achieve image localization within stereophonic recordings. As the
different music sources were recorded separately and then mixed as the stereo channel, the tentative position of a particular instrument in the stereo field was found by deploying a panoramic potentiometer commonly known as Pann Pots. A panoramic Potentiometer is an electronics component that is used to create stereo images by sending signals to the left or right channel or both channels equally in a loudspeaker or headphones.

### 2.3 Signal processing and machine learning algorithm

Signal processing algorithms are used to extract features from the signal and transform these features into numerical form so that the machine learning algorithms can be used on that data. Feature selection methods try to find a suitable subset of the input variables from the data. Data mining finds helpful information from the dataset based on the input variables.

Vembu and Baumann [68] used Independent Component Analysis (ICA) and Non-Negative Matrix Factorization (NMF) techniques to separate vocal from monaural/polyphonic music. Previous research works encountered problems while using ICA or separating non-stationary sources of monaural music. Assuming that the vocal section carries the melody part, the paper tried to separate the vocal part from the rest of the music track, the authors first split the vocal and non-vocal parts for the feature extraction by using the combination of Mel-frequency Cepstral Coefficient (MFCC), Perceptual Linear Predictive Classifier (PLP) and Log-frequency power coefficients (LFPC). A variety of neural networks and support vector machines (SVM) trained the system.

An article [55] written by Smaragdis et al. discussed different types of Non-negative Matrix Factorization (NMF) models, which were more efficient for extracting valuable features from complex mixtures than other matrix decomposition techniques. The authors described Probabilistic, continuous, discrete, and dynamic models of NMF and defined the appropriate use of each of the models in different fields. According to the authors, a few seconds of data is adequate to train the NMF system to separate the source from the complex mixture.

KıRbiZ, and GüNsel [31] used the blind source separation technique to improve the quality of separated music sources. It used subspace learning, based on Non-negative Matrix Factorization (NMF) with a weighted $\beta$-divergence cost function. Woo et al. [74] used Least Square Distance (LSD), an unsupervised learning algorithm for sparse non-negative matrix factor time-frequency deconvolution for music source separation. It used optimized fractional $\beta$-divergence for the cost function.
Chan and Yang [9] suggested using an informed group sparse representation to separate seeds from the music. A model was built using a learned dictionary based on a chord sequence. The dictionary exhibited group sparsity and also included melody annotations. Formulation of the problem was formatted to be similar to Robust Principal Component Analysis (RPCA). That problem was solved by using the alternating direction method of multipliers. In the end, the paper showed the connection between the proposed representation and the low-rank representation.

Due to the repeating nature of accompaniment in music, Rafii et al. [46] posited a framework named REpeating Pattern Extraction Technique (REPET) to separate the complement. A repeating music section was first extracted using a beat spectrum. Then, this extracted information was used to estimate the accompaniment’s spectrogram by averaging the identified repetitions.

Different types of image Pixel-level classification, Image to Image Translation, Automatic colorization, and image segmentation models are also incorporated to solve the music source separation problem. Wolf et al. [72,73] used rigid motion segmentation for singing voice separation from music. First, a velocity vector represented the harmonic template models with amplitude and pitch modulations. Then, a wavelet transformation was performed on the harmonic template models to build an audio image where the amplitude and pitch dynamics can be separated through the velocity vector. A velocity equation similar to the optical flow velocity equation was derived to use in the images for segmenting velocity components. The harmonic templates were identified and modeled into different sources in the mixture and separated by approximating the velocity field over the corresponding harmonic template models.

Yen et al. [75] propounded an approach using spectro-temporal modulation features, which decomposed the audio mixture using a two-stage auditory model which consists of a cochlear module and cortical module. In this model, spectro-temporal modulation features were extracted from the TF units. Then, the TF units were clustered into harmonic, percussive, and vocal components using the EM algorithm and re-synthesized the estimated signals.

Researchers even combined two models to solve the problem of music separation. Wang and Ou [71] used the combination of melody extraction and NMF-based soft masking algorithm to identify accompaniment, unvoiced and voiced segments in the music. First, a melody part was extracted by using a Hidden Markov Model (HMM) with Mel-frequency Cepstrum Coefficient (MFCC) and Gaussian Mixture Model (GMM). Then, they estimated the pitch of the vocals from the voiced segments, and an HMM
was used in the Viterbi algorithm to identify the vocal part. In the end, a soft mask was applied to separate the voice and accompaniment parts from the mixture.

Venkataramani et al. [69] proposed combining sinusoidal modeling and matrix decomposition for music source separation. The combined model incorporated prior knowledge about the singer and phoneme identity. A predominant pitch algorithm was applied to the annotated sung regions, and harmonic sinusoidal modeling was applied. Then, they estimated the spectral envelope of the vocal component from the spectral envelope of the mixture using a phoneme dictionary. After that, an extension of NMF was used to train from the spectral envelope dictionary representing sung vowels from song segments of a given singer. At last, a soft mask using the singer-vowel dictionary was approximated to process and capture the vocal component.

Hu and Liu [26] integrated matrix decomposition and pitch information to separate the vocal part from the music. Non-negative matrix partial cofactorization, which combines prior knowledge about the singing voice and the accompaniment, separated the mixture into singing voice and accompaniment portions. They then identified the singing pitch from the singing voice portions and derived a harmonic mask, and finally, the missing feature method was used to reconstruct the singing voice. This paper also defined temporal and sparsity criteria for the algorithm.

### 2.4 Signal processing and deep learning algorithm

Neural Networks show promising results in the music technology domain ranging from automatic music generation and music source separation. Still, Neural networks require a large amount of training data that need more time and space. Sometimes model and model parameters of any musical application made from the neural network are difficult to interpret.

Neural networks, especially Deep learning models nowadays (gaining more attention), are a powerful tool for separating from audio sources. Nugraha et al. [43] proposed a framework that used a deep neural network-based iterative algorithm combined with spatial covariance matrices to encode the source spatial characteristics to estimate the reference spectra. The proposed framework, built for evaluating large music datasets, used an iterative Expectation-Maximization (EM) algorithm. Jansson et al. [29] adopted a framework named U-Net architecture for music source separation, which was initially developed for medical imaging to enhance the precision and localization of microscopic images of neuronal structures. The U-Net architecture, which comprises an encoder
and decoder, decomposed the audio signal to voice and backing track using low-level audio features. As the neural network model works only on the magnitude of audio spectrograms, the audio signal for each music was rendered by developing a spectrogram. Stoller et al. [59] used end-to-end source separation by using the U-Net framework in the one-dimensional time domain. It also improved the quality of the framework by adding an output layer by enforcing source additivity.

Huang et al. [27] used deep recurrent neural networks with different temporal connections to separate singing voices from monaural recordings in a supervised setting. This paper used a single neural network with multi-layer time scale Deep Recurrent Neural Networks (DRNNs) to improve the previous models of audio source separation, which used a two-stage single-layer linear network, predicting the clean spectra via a linear transform. This model can find hidden features and structures from the data using multi-layer approaches.

Spleeter [23] used the U-net architecture to separate the music sources. It is the only system that trains the system with four audio stems but can separate five audio stems. The fifth audio stem that can be separated from the audio mixture is the piano stem. Though it can separate five audio stems, it works better when separating four audio stems. This model used the U-Net encoder-decoder CNN model with skipped connection proposed by [29] without any major modification. In the training phase, the model approximated a soft mask for each audio stem by using 12 layers of CNN; six of them are used for encoding and the rest six are used for decoding. A soft mask was a computational process to emphasize a particular region in an image while suppressing the other region or information of that image. The model was trained with the MUSDB18 dataset along with the other datasets. The training took a total of one full week using a single core of Graphical Processing Unit (GPU). In the testing phase, the model either used a soft mask or multichannel Wiener filter to separate audio stems from the spectrogram.

Most deep learning networks are designed to either take waveform domain data or temporal data as input. However, most recently, a new system named Hybrid Demucs [15] has been proposed, which can take waveform domain as well as time-frequency domain data as input. Two separate systems were integrated into this model, and the systems can simultaneously train themselves or use only one system to separate audio stems from the audio mixture.

KUIELab-MDX-Net [30] model used a modified version of the TFC (Time-Frequency Convolutions) -TDF (Time-Distributed Fully connected)-U-Net to balance the performance and source required for audio source separation. It proposed a system with
two branches for source separation; one was a time-frequency branch, and the other was a time-domain branch and had six separate networks. The time-frequency branch consisted of five networks. The first four of them were U-net-based separation networks and estimated the four audio sources independently. The fifth network known as the Mixer model enhanced each predicted output. In the time domain, the last network was used to separate the four audio sources separately. The final estimation of the separated source was the weighted average of these two branches.

The Time-Frequency domain conversion requires the long audio divided into shorter segments for proper visualization and extracting features from the audio. This audio division led to the loss of correlation between samples between two segments. Moreover, the spectrogram deals with the magnitude of the audio signal but provides no information about the phase. The Time-Frequency domain conversion requires selecting the proper frame length, window, and overlap.

To solve these issues, Stoller et al. [60] proposed a model Wave-U-Net developed based on the U-Net to separate audio stems in the time domain. This proposed model resample and combine high-level audio features in different time scale from the audio. These features were then used in the Wave-U-Net framework to separate singing voice and multi-instrument from the audio mixture in the time domain. The Wave-U-Net had some modifications over U-Net architecture for better source separation performance. The Wave-U-Net was a CNN network where Convolution, Decimation, Upsampling, and then again Convolution were performed sequentially to achieve audio source separation.

To resolve the issue of discarding phase information by the time-frequency domain, another paper [36] proposed a time-domain music source separation model and compared the separation quality by the time domain and time-frequency domain audio source separation models. This paper modified the Wavenet model for monaural audio source separation by architecting a ten-layer CNN network and deploying residual and skip connections for each layer.

### 2.5 Removing interference from the separated source

Audio signals separated from audio mixtures are often contaminated by noise and contain other audio sources’ audio parts. The noise is introduced while transforming audio to the TF domain by STFT and reversing from the TF domain to the time domain by ISTFT. Audio denoising aims at attenuating the noise and removing unwanted parts of the signal while retaining the underlying or base signal. Grais et al. [21] proposed
two-stage music source separations to avoid interference in the separated music in which sources are separated from the mixed-signal in the separation stage and interference and distortion in the separated music are reduced in the enhancement stage. This paper focused on the second stage or enhancement stage, where the noise and interference of the separated music sources are reduced by proposing two Deep Neural Networks (DNN). The first DNN was used to enhance the quality of each separated music source by using its DNN to fine-tune and remove any unwanted signal from the separated sources. A single DNN trained all sources. The DNN was trained to predict the distortion to maximize the differences between the anticipated sources and minimize the interference and distortion between the separated sources.

### 2.6 Dataset

A well-defined dataset helps the researchers design the source separation system or model properly. On the other hand, ambiguous datasets mislead the researchers and provide false results. The dataset plays a crucial role in deep learning as the training depends on the dataset. There is always a shortage of well-defined audio datasets, especially for audio source separation. Some researchers built their dataset for training and testing to mitigate the issue. The MUSDB18 [45] audio dataset is treated as the complete audio dataset for audio source separation until now. Because it has only 150 full-length audio recordings, this dataset is not enough for training any source separation model. As the dataset is small and contains only English music, the source separation model does not get enough information to separate music other than English music. This problem hence causes dataset bias. To overcome the bias, some researchers train their model with other datasets [23].

The other dataset that is used for audio source separation research and development is MedleyDB [4]. The main issue with this dataset is it does not have any fixed amount of audio stems for every song. Moreover, the dataset represents a total of 82 instrument categories but contains fewer songs (a total of 122 songs) than the MUSDB18 [45] dataset.

To overcome the selection bias of the audio dataset, Manilow et al. [39] proposed a new audio source separation dataset which is derived from the Lakh MIDI dataset [44]. Lakh MIDI dataset contains MIDI tracks of different musical genres and music from different cultures, which solves the issue of the selection bias for the dataset. Each MIDI track of the Lakh MIDI dataset is converted to an audio instrumental mixture, and each MIDI instrument track of that song is converted to the corresponding audio stem.
This dataset has 2100 songs which fill the gap of smaller dataset size. But the major shortcoming of the dataset is that it does not have any vocals stem. It is an instrumental audio source separation dataset.

MIR-1K dataset [25] contains 1000 song clips, and these clips are taken from 110 Chinese karaoke Pop songs. Each song clip has two separate tracks; vocal and non-vocal parts. So, this dataset is suitable for two stem music source separations. Each song clip has a short length, 4 to 13 seconds. The shorter length of the audio clips and selection bias of songs (all audio is from the Pop genre) make the dataset unfavorable to train a neural network.

Though the MUSDB18 [45] dataset has some limitations, it is the most commonly used dataset in the audio source separation domain. It also maintains a webpage named MUSDB18 Benchmark [54] to track the development of a new audio source separation model that uses the MUSDB18 dataset to train and test the model. MUSDB18 Benchmark webpage helps the researchers to compare the performance of their developed model with other audio source separation models.

2.7 Evaluation tools

Although there is a lot of work done to develop the audio source separation model, less attention was given to creating tools for evaluating these models’ separation efficiency and accuracy or comparing the efficiency between two models. There are only a few tools out in the MIR community to compare the separated audio with the original audio. Some evaluation tools are dataset-specific. For example - museval [61] is developed for evaluating separated data on the MUSDB18 [45] dataset. Some are programming language specific such BSS_Eval is for Matlab programming language, mir_eval is for python. Both museval and BSS_Eval have the same metrics and were developed by the same authors. It is used to evaluate the quality of the separated audio in Blind Source Separation(BSS). The Source to Distortion Ratio (SDR), Source to Artifact Ratio (SAR), Source to Interference Ratio (SIR) are some standard matrices used to evaluate the separated audio. These evaluation matrices compare the ground truth (original audio stem) with the separated audio based on Source to Noise Ratio (SNR). Noise is comprised of the original audio, distortion, interference. Each of these matrices calculates the ratio of the original audio and its corresponding noise portion. For example, SIR quantify the ratio of original audio and interference in the separated audio.

Subjective audio testing is a popular method to access audio quality subjectively. By
employing subjective audio testing, a system user can provide their valuable feedback about separated audio, thus evaluating the separation accuracy and efficiency of an audio source separation system. Among all the subjective audio testing methods, the researchers commonly used the Mean Opinion Score (MOS) [15]. In the MOS, participants listen to the original audio and separated audio and then provide a score of the separated audio in the range from 0 to 5 by comparing the original and separated audio quality.
Chapter 3
Theoretical Background

This section describes all the necessary theoretical background for audio source separation that will be helpful if someone wishes to study it.

3.1 Artificial Intelligence, Machine Learning, and Deep Learning

Artificial Intelligence (AI), Machine Learning (ML), and Deep learning are interconnected. Machine Learning is one of the parts of Artificial Intelligence, and deep learning is a subset of traditional Machine Learning. Artificial Intelligence, traditional machine learning, or deep learning, has two learning approaches based on the dataset used: supervised learning and unsupervised learning. In supervised learning labeled datasets are used, which supervises an algorithm to classify or predict the outcome, whereas unsupervised learning uses unlabeled data. So, supervised learning requires human supervision to label the data, but unsupervised learning does not require human supervision. In supervised learning, the type of output of the system is known. The accuracy of supervised learning can be measured by comparing output and given input. Supervised learning is used in classification and regression problems.

For example, if a labeled audio dataset of different musical instruments is provided in the training phase, the supervised learning algorithm will learn the label of the data and hidden architecture from the audio and know the name of the instrument that the audio belongs to. If an audio from the piano is supplied to the algorithm in the testing phase, the model will classify it as piano sound. So, here the type of output is known. The accuracy of the system is also easily measurable.

On the other hand, the unsupervised learning algorithm will cluster the data based on
their hidden architecture. If an unlabeled audio dataset of different musical instruments is given, unsupervised learning will group the audio in a cluster. It can not identify which musical instrument that audio comes from. In traditional supervised machine learning and deep learning, the goal is to design an algorithm to predict or classify an outcome. These algorithms are not straightforwardly programmed to get an outcome. Instead, they are designed using mathematical functions and statistical inference to use data for predicting the outcome. Optimizing the outcome is performed in a phase known as training by utilizing sample data known as training data [17].

The difference between traditional machine learning and deep learning mainly lies in the types of algorithms they use, and how the data is used in the input. Traditional machine learning models generally use a statistical algorithm such as Linear Regression, Logistic Regression, Support Vector Machine (SVM), Naive Bayes Classifier. These algorithms do not have any internal mechanism to change the raw data into the data it can handle. So, in traditional machine learning, data are processed, and essential features from the processed data are selected. Features are the independent variable of the data. These features are fed as the input to train the machine learning model. So, data pre-processing and feature selection are integral for the machine learning model. For example, Mel-Frequency Cepstral Coefficients (MFCCs), Zero Crossing Rate (ZCR), Spectral Roll-off, etc., are used as audio features in audio-related problems. These features are extracted from time-domain or frequency-domain audio data. Another intention of providing selected features in the machine learning model is to maximize its learning from the variables tied to the problem and reduce training time.

Machine learning algorithm learns and provides an output based on the supplied features. The algorithm learns the data label, maps the supplied and outcome variables, and predicts the outcome. If any attributes are not appropriate for the type of task, then that unwanted feature can adversely impact the learning of the model. As a result, the algorithm may provide less accurate output or outcome. There are no specific rules or research that has been made on feature selection that is appropriate for a machine learning algorithm or a task. So, in short, in machine learning algorithms, human supervision is required for feature selection. If the selected features are not appropriate for the model, new features from the data need to be selected. Moreover, human control is necessary to tune the parameter of the machine learning model.

Deep learning is the subclass of machine learning that generally involves supervised learning without human intervention. Deep learning is called deep due to its architecture. Neural networks, the backbone of deep learning, have input, hidden, and output layers.
All the calculations and learning of deep learning algorithms are performed in the hidden layers of neural networks deep inside the network. That is why these algorithms are called deep learning algorithms. The more hidden layers neural networks have, the more deep that network is. In a typical Neural Network, 5–10 or more hidden layers are common. More deep networks are employed with more than 50–100 layers in cutting-edge applications [51].

Neural networks can handle and learn from raw data due to their internal mechanism. So, raw or processed data are used as the deep learning model/algorithm input. The deep learning model selects features independently without supervision to maximize its knowledge and produce better output than the machine learning model. It also adjusts the parameters on its own. As deep learning models extract features from the data themselves, the feature selection process is not involved. Also, as the model selects the features, it can maximize the learning. As a result, human supervision is not required. A deep learning network is trained to reproduce the input, and the hidden layers learn the features needed to predict and reproduce the input. As deep learning models take raw data or pre-processed data as input, it requires more training time than traditional machine learning models. However, deep learning models provide more accurate output than machine learning models.

3.2 Neural Network

Neural Network algorithms are the heart of deep learning, whose structure is inspired by the biological mechanism of the human brain [50]. The neural network is vastly deployed in complex and nonlinear problems where the accuracy of the output is crucial. They are used in automatic music generation, audio source separation, automatic music instrument identification. For each of these problems, the system’s output must be precise and correct. For example, for automatic music generation, the output music must be musically correct. Similarly, for audio source separation, each separated audio instrument track must only contain the audio of a single instrument. It should not have any audio of other instruments in the separated track.

Figure 3.1 shows a simple neural network with one input, one hidden layer, and one output. Both the input and hidden layer have four nodes or neurons each, represented by the circle. The connection between neurons represented by lines in Figure 3.1 is known as the axon. The input layer node receives the input and passes the information to the hidden layer by the axon. Each hidden layer neuron receives one or more inputs. All the
calculations and predictions occur in the hidden layer. The final prediction is transmitted from the hidden layer to the output layer via axon, and the output layer presents the prediction.

Neural Networks tend to have higher time complexity, but they provide compelling results. There are several kinds of Neural Networks available. Two commonly and widely used types of neural networks are Convolution Neural Network (CNN) and Recurrent Neural Network (RNN). Both CNN and RNN share some common parameters. These parameters are categorized in two ways: learnable parameters and hyper-parameters. The hyper-parameters are the top-level parameters whose values ensure a model’s efficient learning from the data in the training phase by controlling the model’s parameter. These elements are described below

\[
\text{output} = \text{activation function}(\text{sum}((\text{dot product} (\text{weight, input})) + \text{bias}))
\]  

3.2.1 Weights

A weight is a learnable parameter in the neural network that determines the strength of the connection between two neurons. At the beginning of forwarding propagation, an arbitrary weight is assigned to an input. The input is multiplied by the weight associated with the axon connecting the previous layer neuron to the current neuron. The weight is adjusted based on the loss function value in the backpropagation. In the same layer,
the different neurons have distinct weights. Higher weight signifies more importance of a neuron over a low weight neuron [1].

From Figure 3.2 and Equation 3.1, it is seen that the weight is multiplied (dot product) by the input, and each axon has a different weight.

### 3.2.2 Bias

In a neural network, bias is a learnable parameter that has a constant value. It is added with the dot product result of weight and input (Equation 3.1). Without a bias, the neural network will be a simple dot multiplication, which will create a data overfitting issue [38]. Figure 3.2 shows bias \( b \) is added with the dot product of the weight and input.

At the beginning of the neural network training, a random bias is assigned per neuron. While backpropagation, the bias values are adjusted using the optimization function and based on the loss function value.

### 3.2.3 Activation Function

In a neural network, an activation function is a mathematical function to assist the model in extracting and learning complex patterns from the input data. The activation function decides which information needs to be transmitted and discarded. Standard
activation functions used in the neural networks are sigmoid, tanh, ReLU, softmax, and Leaky ReLU [52].

\[ y = mx + c \]  \hspace{1cm} (3.2)

Equation 3.2 is a general equation of a straight line where \( m \) is the gradient (steepness) of the line, and \( c \) is the y-intercept which is a constant.

After comparing Equation 3.1 and Equation 3.2, it is seen that Equation 3.1 is a form of a straight line equation where weight is the gradient \( m \) and bias is constant \( c \). So, the weight determines the steepness of the activation function, which means how quickly or slowly, the current neuron will send the information to the next neuron. The bias moves the activation function curve left or right. Figure 3.3 shows the Sigmoid activation function curve [33,67].

### 3.2.4 Forward and Back Propagation

Forward propagation starts from the input layer, goes through the hidden layers, and ends at the output layer. A random weight is assigned with each axon, and a random bias is assigned for each neuron in the forward propagation. In forward propagation, each learnable parameter learns and updates its values from data, bias, and weight.

On the other hand, the backward propagation flows in the reverse order; starts from the output layer and ends in the input layer, and changes the values of learnable...
parameters. Backward propagation uses the loss function value and the optimization algorithm to change the weight of the axon of each layer and optimizes the network.

**Figure 3.4 Forward and Backward Propagation of a Neural Network**

In the Figure 3.4, the forward propagation is denoted by black arrow and the back propagation is denoted by red arrows.

### 3.2.5 Tensor

A tensor is a data representation technique in AI, traditional ML, and deep learning. A tensor is a multi-dimensional (n-dimensional) array data structure where variables are arranged in a grid format with a variable number of axis. Each data is converted into a numerical format. A neural network deals with numerical data only. All data must be converted into a numerical format before feeding them to the neural network. Tensor is the data structure where data are represented in the n-dimensional array so that neural networks can use the data [32,53].

### 3.2.6 Loss Function

In the training phase, a neural network is trained to replicate its input from data. The loss function calculates the difference between the model’s predicted and actual values while training. The information about this difference or error helps optimize the deep learning model.
While developing a model, the main aim is to minimize the difference or loss of the model so that the predicted output is the same as the actual output. The loss function helps identify and quantify the loss or difference and using that loss information, the neural network will change its parameters to minimize the loss and optimize the output [8].

### 3.2.7 Optimizer

Optimization is a technique to maximize the learning of a deep learning model by modifying the model parameters such as learning rate, bias, and weight. It helps to minimize the overall loss of the model and maximize the model’s accuracy. Using the value from the loss function, the neural network adjusts the values of the learnable parameters in backpropagation by using an optimizer. The most popular optimizers are Gradient Descent, Stochastic Gradient Descent (SGD), Adam, AdaDelta.

### 3.2.8 Dropout

Data overfitting is a statistical error that occurs when a deep learning model is trained with a smaller size dataset. There can be noise and abnormalities in the data. When the dataset size is small, deep learning considers the noises and anomalies are part of the data. These noises can take a more significant part of the learning if the dataset is small and the deep learning model predicts and classifies incorrectly by learning from these noises [7].

![Figure 3.5 Dropout Mechanism in Neural Network [57].](image-url)
The dropout mechanism is used to avoid data overfitting. Dropout is a method where the result or output of a few neurons is ignored randomly in each epoch during the training process. When several neurons drop randomly in each epoch, then the shape of the neural network changes in each epoch and is trained as a different neural network each time. As different neural networks fit the data differently, so the overall overfitting problem is reduced by a significant amount. A good value for dropout in a hidden layer is between 0.5 and 0.8. The dropout mechanism has a proven record of minimizing overfitting problems in the neural network, especially image classification, image segmentation problems [57].

Figure 3.5 shows the dropout mechanism used in the neural network. Figure 3.5(a) shows the neural network where all the neurons take part in the prediction of an output. Figure 3.5(b) shows the neural network where few neurons drop arbitrarily while predicting an output. When neurons are dropped randomly in each epoch, the neural network acts as a new version of the network and fits the data differently. This mechanism reduces the data overfitting problem.

### 3.2.9 Epochs

Epochs is a hyperparameter that controls how many times the entire training dataset will pass through the learning algorithm. In general, an epoch consists of at least one complete cycle through the training data. In each epoch, the number of data that passes through the model equals batch size. So, if the size of the dataset is $s$ and the batch is $b$, then to cover the entire dataset, the number of epochs required can be calculated by the Equation 3.3

$$
minimum \ epochs \ or \ timesteps = \frac{size \ of \ the \ dataset}{batch \ size} = \frac{s}{b} \quad (3.3)
$$

A full epoch consists of a forward propagation and a backward propagation [5, 6].

### 3.2.10 Batch Size

Batch size is a hyper-parameter that governs the number of training data to pass through the network for each timestep before the model’s internal parameters are updated. There are three types of batch size choice options. They are batch mode, mini-batch mode, and Stochastic mode [14].

For batch mode, the whole dataset is passed through the deep learning algorithm for
training in a single epoch/timesteps. Batch mode (equation 3.4) is suitable if extensive data is available in the dataset. In reality, a larger dataset is not available, especially for audio. As all the data samples of the dataset passed in a single epoch, the learning rate of the algorithm is relatively low [5].

\[ \text{Batch Size} = \text{Size of Training dataset} \] (3.4)

For Mini-batch mode, the whole dataset is divided into small groups, and in each epoch, a group of data samples is passed through the system for training. If a dataset is small, then Mini-batch mode is a suitable option to select. In mini-batch mode, the commonly adopted batch size is set to 8, 16, 32, etc. In reality, the mini-batch mode (equation 3.5) is selected as a preferred batch mode because the algorithm learns faster.

\[ 1 < \text{Batch Size} < \text{Size of Training dataset} \] (3.5)

For Stochastic mode, each data sample of the dataset comprises a batch. So, the minimum number of epochs required to pass the whole dataset equals the number of data samples in the dataset. The training process takes longer in Stochastic mode (Equation 3.6), but the algorithm’s learning efficiency is low compared to the time required to train the system.

\[ \text{Batch Size} = 1 \] (3.6)

### 3.3 Recurrent Neural Network

Recurrent Neural Network (RNN) algorithms are famous for solving tasks that deal with sequential data and multi-class classification problem. Sequential data refers to where the sequence or order of the data points is essential. In sequential data, one data point in the dataset is dependent on the other data points in the dataset. A memory function is necessary to remember the dependency and order of the data samples [48]. Multimedia data are sequential in nature. Examples of sequential data are time-series, audio, text, and video. For audio data, each audio sample is dependent on the other audio samples of the time series, and the order of the samples is fixed. Recurrent Neural Network (RNN) has a memory function to model sequence data, making the RNN popular in multimedia problems [11,34].

Multi-class classification refers to the task where output can only belong to one
out of many possible categories, and the model must accurately decide that category. Separating musical sources from audio mixture or identifying musical instruments played in the audio mixture are examples of multi-class classification.

### 3.3.1 Simple Recurrent Neural Network

Simple Recurrent Neural Network (RNN), also known as Vanilla RNN, has a memory function to model sequence data. However, it has a deficiency of carrying information for a short time which creates problems in the sequential tasks. Figure 3.6 shows the architecture of a simple RNN cell.

![Internal Architecture of a Simple RNN cell](image)

**Figure 3.6 Internal Architecture of a Simple RNN cell**

Simple architectured RNN cell takes the input $x^{<t>}$ and information of the previous cell $h^{<t-1>}$. Then it calculates the weighted sum of the previous memory state $h^{<t-1>}$ and the input $x^{<t>}$ at timestep $t$, and then add a bias $b_h$ with the result. The $tanh$ activation function regulates the output information need to pass to the next cell via $h^{<t>}$. The equation 3.7 is used to calculate the output.

$$\tilde{h}^{<t>} = tanh(W_h[h^{<t-1>}, x^{<t>}] + b_h)$$  \hspace{1cm} (3.7)

The primary deficiency of the simple RNN is that it can only retain the memory of all the information seen in previous timesteps $t - 1$. 
3.3.2 Long Short Time Memory

Traditional RNN has only one memory line, which only retains and carries information from the previous cell. Moreover, it has no mechanism to control the information flow; to decide whether to use or not use the information from the last cell. These shortcomings create a problem with the sequential data where the flow of information about the order and dependencies of each data sample needs to pass from the first RNN cell to the last RNN cell. Long Short Time Memory (LSTM), a modified version of RNN, was introduced to overcome the deficiency of the simple RNN architecture. Two modifications make LSTM better than traditional RNN. Each LSTM cell has two memory lines. One is for a long-time memory line that carries information from the cell beginning to the end. As sequential data points have dependencies on each other, a long-time memory line is helpful to inherit information from all the previous cells so that any current cell can access the information and make a decision based on the information of the previous cells. The other memory line is a short-time memory line with data from the previous cell. In LSTM, gates are introduced, which regulate the flow of information. Gates decides whether an LSTM cell will keep the long time memory information or modify it. Also, gates decide to keep or discard the short-time memory of the previous cell. For example, the long-time memory line carries the information of the audio sample order and sample dependencies from cell to cell. The gates can modify the dependency if required. The short-time memory line carries information from the previous cell about the last audio sample, and gates can keep the information if necessary or discard the information.

Figure 3.7 shows the cell architecture of an LSTM cell. To carry the memory for long-range, it deployed a long time memory line $c^{t-1}$ and to get the information from the previous cell, the uses short time memory line $h^{t-1}$.

In the beginning, forget $\Gamma_f$ decides whether to keep the information of the previous cell or discard it. The forget gate calculates the weighted sum of the last short-time memory $h^{t-1}$ and the input $x^{t}$ at timestep $t$, and then add a bias $b_f$ with the result. The sigmoid activation function has a range of 0 or 1 decides to keep the information (1) or discard the data (0). The equation 3.8 is used to calculate the forget gate.

$$\Gamma_f = \sigma(W_f[h^{t-1}, x^{t}] + b_f) \quad (3.8)$$

The update gate $\Gamma_u$ computes the weighted sum of the last short-time memory $h^{t-1}$ and the input $x^{t}$ at timestep $t$, and then add a bias $b_u$ with the result. The sigmoid activation function has a range of 0 or 1 decides to update the information (1) or keep the
data from the previous short-time memory cell(0). The equation 3.9 is used to calculate the update gate.

\[ \Gamma_u = \sigma(W_u[h^{t-1}, x^t] + b_u) \]  

(3.9)

\( \tilde{c}^t \) is known as the candidate function for longtime memory line \( c^t \). \( \tilde{c}^t \) evaluates the weighted sum of the last short-time memory \( h^{t-1} \) and the input \( x^t \) at timestep \( t \), and then add a bias \( b_c \) with the result. The \textit{tanh} activation function has a range of 1 or -1 decides to update the information (1) or keep the data from the previous short-time memory cell(0). The equation 3.10 is used to calculate the candidate function \( \tilde{c}^t \).

\[ \tilde{c}^t = \text{tanh}(W_c[h^{t-1}, x^t] + b_c) \]  

(3.10)

The output gate computes the weighted sum of the last short-time memory \( h^{t-1} \) and the input \( x^t \) at timestep \( t \), and then add a bias \( b_o \) with the result. The \textit{sigmoid} activation function has a range of 0 or 1 decides to update the information (1) or keep the data from the previous short-time memory cell(0). The equation 3.11 is used to calculate the output gate.
\[ \Gamma_o = \sigma(W_o[h^{t-1}, x^t] + b_o) \] (3.11)

To calculate the long time memory \( c^{<t>} \), update gate \( \Gamma_u \) is multiplied with the candidate function \( \tilde{c}^{<t>} \). Then the forget gate \( \Gamma_f \) is multiplied with the previous cell long time memory \( c^{<t-1>} \). In the end, both of the results are added together. The equation 3.12 is used to calculate the long time memory.

\[ c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>} \] (3.12)

\( a^{<t>} \) is the short-time memory that passes to the next cell. First the long time memory \( c^{<t>} \) is regulated by \( \tanh \) activation function. Then the result is multiplied by the output gate’s output \( \Gamma_o \). The equation 3.13 calculates the short-time memory.

\[ h^{<t>} = \Gamma_o * \tanh c^{<t>} \] (3.13)

### 3.3.3 Gated Recurrent Unit

Long-Short Time Memory (LSTM) overcomes the traditional Recurrent Neural Network (RNN) deficiency of carrying information for a long time. However, LSTM has some shortcomings too. As LSTM has introduced two memory lines and three gates, many parameter calculations are associated with the LSTM, which makes LSTM a calculation-heavy RNN. These calculations require more time which increases the time complexity of the network. Moreover, LSTM networks tend to over-fit the simpler data due to their inherent complexity. Gated Recurrent Unit (GRU), a simplified version of LSTM, is introduced by Chao et al. [13] to overcome these issues. GRU combines both Long-time and short-time memory lines into a single line. It also reduces the number of gates to regulate the flow of information. These two features of the GRU make its cell architecture more straightforward and reduce the calculation without compromising efficiency compared to LSTM.

Figure 3.8 illustrates the architecture of a GRU cell. GRU architecture combines both short time memory line \( h^{<t-1>} \) and long time memory line \( c^{<t-1>} \) of the LSTM (Figure 3.7) in one memory line \( c^{<t-1>} \). It also combines the forget gate and update gate into one reset gate \( \Gamma_r \). As the number of memory lines and number of gates less than the LSTM, GRU has less parameters to calculate. Figure 3.3 visualize the internal structure of the GRU cell.

In the beginning, the reset gate \( \Gamma_r \) decides whether to keep and use the information of
the previous cell or discard it. The reset gate calculates the weighted sum of the last cell memory \(c^{t-1}\) and the input \(x^{t}\) at timestep \(t\), and then add a bias \(b_r\) with the result. The \textit{sigmoid} activation function has a range of 0 or 1 decides to keep the information (1) or discard the data (0). The equation 3.14 is used to calculate the forget gate.

\[
\Gamma_r = \sigma(W_r[c^{t-1}, x^{t}] + b_r) \tag{3.14}
\]

\(\tilde{c}^{t}\) which is the candidate function of output \(c^{t}\) evaluates the weighted sum of the last memory \(c^{t-1}\) and the input \(x^{t}\) at timestep \(t\), and multiplied the result with reset gate output. Then a bias \(b_c\) is added with the result. The \textit{tanh} activation function has a range of 1 or -1 decides to update the information (1) or keep the data from the previous short-time memory cell(0). The equation 3.15 is used to calculate the candidate function.

\[
\tilde{c}^{t} = \tanh(W_c[\Gamma_r * c^{t-1}, x^{t}] + b_c) \tag{3.15}
\]

The update gate computes the weighted sum of the last short-time memory \(c^{t-1}\) and the input \(x^{t}\) at timestep \(t\), and then add a bias \(b_u\) with the result. The \textit{sigmoid} activation function has a range of 0 or 1 decides to update the information (1) or keep
the data from the previous short-time memory cell(0). The equation 3.16 is used to calculate the update gate.

\[ \Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t}>] + b_u) \quad (3.16) \]

To calculate the output \( c^{<t>} \), update gate \( \Gamma_u \) is multiplied with the candidate function \( \tilde{c}^{<t>} \). Then \( (1 - \Gamma_u) \) is multiplied with the previous cell memory. In the end, both of the results are added together. The equation 3.17 is used to calculate the output.

\[ c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>} \quad (3.17) \]

From equation 3.14, it is seen that the value of \( \Gamma_u \) will be 0 or 1. So, if the \( \Gamma_u \) is 0, then \( \Gamma_u * \tilde{c}^{<t>} \) will be 0, and the output will be \( c^{<t-1>} \) which is the memory of the previous cell. If the \( \Gamma_u \) is 1 then \( (1 - \Gamma_u) * c^{<t-1>} \) will be 0 and the output will be \( \tilde{c}^{<t>} \) which is the output of the current cell.

### 3.3.4 Bi-directionality of Recurrent Neural Network

The traditional RNN is unidirectional. That means the information passes only in one direction - left to right (or input to output). In the case of unidirectionality, the RNN cell only gets data from the previous cell; it can not access the data from the future cell. From Figure 3.9 it is seen that the RNN cell \( a^{<3>} \) can only get the information from cell \( a^{<1>} \) and \( a^{<2>} \). It can not get any information from the future cell \( a^{<4>} \) or \( a^{<5>} \). This one-directional flow of information hinders the learning of the traditional RNN algorithms.

Another layer of RNN is added to remove this deficiency. In the added RNN layer, information flows backward from right to left. So, in that case, the RNN cells can access information from the past and future cells. From Figure 3.10, it can be seen that cell \( a^{<3>} \) and cell \( a^{<8>} \) get the same input. Nevertheless, cell \( a^{<3>} \) gets the information from the last cell, which flows from the left to right direction. On the other hand, cell \( a^{<3>} \) gets the information from its previous cell, which is flowing from right to left. The output from both \( a^{<3>} \) and \( a^{<8>} \) are concatenated together, and a new output is generated. So, the output function gets information from both previous and future cells and can decide which information it needs to keep or discard. Then that output will be used as the input of the next RNN layer.
3.4 Sound and Audio

Sound and audio are often used interchangeably. But in reality, these two terms differ. Sound is the physical phenomenon, whereas audio is the recorded version of this physical phenomenon. The critical difference between sound and audio is as sound is a physical phenomenon, it can not be changed but can be controlled. On the other hand, audio is a
3.4.1 Audio Formats

The audio file format is categorized into two categories. One is lossless, and another one is lossy. In the case of lossless audio file format, all information is preserved. Though lossless audio file format has a larger file size, it is the preferred audio analysis and research format. On the other hand, lossy audio file format exploits psycho-acoustics and human hearing capabilities and discards information outside the human hearing range. As the lossy audio format does not contain all the information and frequency content, the neural network does not get sufficient and suitable audio features and accurate representation of frequency, lossy audio format is not used for audio source separation training.

Wave (.wav) file is a lossless audio format that contains all the information of audio in it. The wave file is used for analysis purposes as it includes and retains all the audio data. The neural network fetches the required features from the lossless audio data. The higher sample rate will provide a more accurate result, i.e., will detect the higher frequency. But a high sample rate increases the file size and needs more time to load and process, thus increasing the system’s time complexity. Though lossless has a larger file size, it contains all the frequency content of the audio. That is why the lossless audio data is used in the audio source separation training and other audio research work.

3.4.2 Multi-track Audio and Audio Stems

A multi-track recording consists of multiple, separate audio tracks for individual instruments or voices. In a multi-track format, each audio source is recorded on different channels. For example, if two guitars are playing in the music, each guitar will be recorded into separate audio tracks. Another excellent example of multi-track audio is recording drums. Drums consist of the hi-hat, snare, bass drums, tom. In general, more than one microphone is used for recording drums. But audio from each microphone is recorded into separated tracks. Figure 3.11 shows the file system of multi-track audio.

In the case of audio stems, audio instruments of the same category are added together and make a new track. For example, all drum instruments are mixed to make a stem. All the voices are combined and create vocals stem. Audio stems are now a popular choice for beat-making, separation of audio sources, and automatic music generation. Figure 3.12 visualize the file system of audio stems.
3.5 Audio Representation

The knowledge of audio representation is required because machine learning and deep learning algorithms use different audio representations for their input. For example, features are extracted from the time and frequency domain data for machine learning models. Feature extraction is the process of reducing the number of input variables of the data. A dataset can contain many variables and noise in the data. All the variables within the data may not be necessary to solve a problem. The feature extraction process only extracts the essential variables of the data. It passes only these variables to the
model so that unrelated variables and noise from the data can be entered into the model.

On the other hand, deep learning algorithm uses raw time domain or time-frequency domain data. The deep learning algorithm automatically selects and extracts features from the data.

### 3.5.1 Time Domain, Frequency Domain and Time-Frequency Domain

Audio is always available in the time domain data. In the time domain data representation, audio samples are graphed against the time. Time-domain or temporal features are easy to extract because they are extracted from audio's primary representation. So, no transformation is required of the audio, and the physical interpretation of these features can easily be understood. Some standard temporal features extracted from time-domain audio are Zero-Crossing rate (ZCR), the total energy of the signal, maximum amplitude, amplitude envelope, etc [18,40].

Frequency-based audio features deal with the frequency content of the audio. A transformation of audio from the time domain to frequency domain is required to extract the frequency-based features from the audio. This transformation is achieved by using Fast Fourier Transformation (FFT) algorithm. Standard frequency-based audio features used in different audio applications are spectral centroid, spectral density, spectral flux, spectral roll-off, etc [18]. The inverse Fast Fourier Transformation (IFFT) algorithm converts the frequency data to time-domain data. Both FFT and IFFT are the algorithms to calculate the Fourier Transformation of the audio signal.

![Time-domain, Frequency-domain and Time-Frequency domain conversion](image)

**Figure 3.13** Time-domain, Frequency-domain and Time-Frequency domain conversion
Time-domain audio representation provides information about amplitude changes over time. Frequency domain representation provides information about frequency in a signal. However, Time-Frequency (TF) domain representation provides information of the frequency changes over time. For TF domain representation, the transformation of time-domain audio is required. Short-Time Fourier Transformation (STFT) algorithm is employed to achieve the transformation. Audio features are extracted from these time-domain and frequency-domain representations. However, from the Time-Frequency domain representation, no feature is extracted. It is a form of representing audio data. Common TF domain representations are Spectrogram, Log power spectrogram, Mel-spectrogram, Log-frequency power spectrogram, and Constant-Q transformation [18]. As deep learning algorithms use raw data or transformed raw data and extract features from the data automatically, TF domain audio representation is vastly used in deep learning. Deep learning models treat the TF domain audio representation (Spectrogram, Mel-Spectrogram) as an image. Inverse Short-Time Fourier Transformation (ISTFT) algorithm converts the TF domain data to time-domain data.

Figure 3.13 represents the conversion between time domain, frequency, and time-frequency domain audio data.

### 3.5.2 Short-Time Fourier Transformation

Short-Time Fourier Transformation (STFT) is a Fourier Transformation algorithm applied to a fixed short-time audio segments using the Fast Fourier Transformation. So, STFT provides the time-localized frequency records where frequency components of a signal change over time. There are few essential parameters of STFT: frame length, frame step or hop size, FFT length and windowing. Each of them has its role in shaping the STFT.

Frame length refers to the number of samples in a small time frame. The hop size determines the number of samples overlapping between two successive frames. Different window functions are used in STFT to remove the spectral leakage in the Fourier transform of the windowed signal. The Hanning window is commonly used in audio applications among these window functions. Hanning window is a cosine window function that taps the boundary sections of a signal to zero. Hanning window provides better frequency resolution and moderate amplitude accuracy of the resulting frequency spectrum. As the audio is converted in the time-frequency domain, a better frequency resolution is desired. So, the Hanning window is the preferred choice for this application. The FFT size signifies the number of bins used for dividing a frame equally. Hence, a single bin
represents a spectrum sample [2].

If the sampling frequency is $F_s$ and the frame length is $L$, the frequency resolution, which means how well we can represent frequency in STFT, is $F_s/L$ Hz. If $T$ is the period that is $1/F_s$, the time resolution is $L*T$ or $L/F_s$. These two equations show that while using the frame length, there is a trade-off between time and frequency resolution in STFT. When a narrow-width frame length is applied, STFT provides a better resolution in the time domain, but it provides a poor resolution in the frequency domain, and vice versa. As the FFT size determines the frequency resolution of the window, it is a common way to increase the FFT size to increase the frequency resolution. However, increased FFT size comes with a cost of higher time complexity of calculating FFT [56]. Most commonly, STFT is visualized in the form of an intensity plot which shows the STFT magnitude over time. This graphical way to represent STFT is known as a spectrogram.

For example, if the audio sampling rate is 16 kHz, the frame length is 512, and the hop size is 256, then the time-analyzed by that frame size is $512 / 16$ kHz $= 32$ms. The overlap is $256/512 = 0.5$, meaning 50% of samples will overlap.

A short time segment is selected to visualize the frequency change over a shorter period. For longer time segments, frequency content and change frequency over time
are hard to visualize. That can be seen from Figure 3.14 where STFT is performed on 3-minute 29-second audio signal.

When the audio is segmented in shorter time segments, i.e., 55-second audio, the time resolution of the spectrogram is enhanced, and the frequency content and how the frequency content changes over time of the audio is easily interpretable. Figure 3.15 shows the STFT performed on shorter time segments of an audio signal.

3.6 Dataset

Dataset is an important and integral part of any data mining, machine learning, and deep learning problem. A deep learning model is heavily dependent on the dataset. Most of the datasets used for audio, especially for audio source separation, are publicly available. Some of the datasets are used for multiple purposes. Some of them are used only for audio source separation purposes. Some commonly used audio dataset are MedleyDB [4], ikala [76], MUSDB18 [45].

There are three phases where data is used in AI, ML, or deep learning. First, data is used to train a model. Then, data is utilized to validate that the model learned from data before testing it, and at the end, data is applied to test the model. So, the dataset
is divided for training, validation, and testing purpose. Generally, a publically available
dataset does not come pre-divided into training, validation, and testing data. Commonly,
a 60%, 20%, 20% rule is applied while dividing the dataset. 60% of the dataset data are
utilized for training the model. 20% data are applied to validate the model, and the rest
of the 20% data are used to test the model.

In most cases, researchers divide datasets for training and testing. They keep a
specific portion of test data for validation. In those cases, they use the 70%-30% rule
where 70% data are applied for training, and the rest 30% data are utilized for testing.
However, they use roughly 15% of the training data to validate the system.

Regardless of the dataset division rule, another import rule is employed while dividing
data for training and testing. The division of data needs to be done in such a way that
the type of training data covers all classes of data in the testing folder. No new type
of test data can be introduced without training the model by the same data type. For
example, the training dataset must contain at least one Jazz music data if the testing
folder contains any music from the Jazz genre. If the dataset’s author does not split the
dataset, researchers should consider the rule before splitting the dataset.
Chapter 4  
Implementation of Bi-directional GRU for Audio Source Separation

The backbone of this source separation system is the Bi-directional Gated Recurrent unit (Bi-directional GRU), which is an updated model of Recurrent Neural Network (RNN). This method section starts with processing the dataset, converts the time-domain audio data to time-frequency domain data, and then the architecture of the Bi-directional GRU is described. Audio Source Separation can be performed in the time domain (raw audio data) or in the time-frequency domain (processed raw audio data). The Time-Frequency domain audio representation provides a far better result than the time-domain audio representation while separating audio stems from the audio mixture. The Neural network treats the time-frequency domain data as an image and analyzes the frequency contents of the spectrogram at the pixel level. Then, based on the analysis, it separates the frequency contents of the audio stems from the audio mixture. Though the separated spectrogram lost the phase information of the audio, Time-Frequency domain audio data representation is the preferable choice for the audio source separation model input due to better separation quality.

4.1 Dataset

The MUSDB18 [45] dataset is the complete audio dataset and is widely used to train, test, and evaluate the efficiency of machine learning and deep learning models of audio source separation. It contains 150 songs of different lengths. The sample rate of each track is 44.1 kHz, and each track is stereophonic. These 150 songs are pre-divided into train and test folders. The training folder has 100 songs and comprises the genre of Rock, Pop, Rock/Pop, Rap, Reggae, Jazz, Heavy Metal, Singer/Songwriter, Country, Electronic.
The remaining 50 songs are used for testing purposes. The test folder comprises songs from Rock/Pop, Rap, Rock, Reggae, Heavy Metal, Singer/Songwriter, Electronic. So, the training dataset has music from ten genres to train a model, but the testing dataset has music from seven genres to test the model.

4.2 Processing the Dataset

The dataset comes in .zip format. First, the dataset is unzipped. The dataset processing has two steps. The first step is to convert the dataset from .stem.mp4 format to .wav format. Then, the audio data is converted to spectrogram using Short-Time Fourier Transformation (STFT).

4.2.1 Dataset Processing

MUSDB dataset is available in the compressed format, and inside, the audio data are in .stem.mp4 format.

![MusDB18 Dataset in .stem.mp4 format](image)

This .stem.mp4 acts like a compressed folder. Each .stem.mp4 file encodes five separate stems of audio. The mapping of the audio is given in Table 4.1. There are two ways to get the audio stems from each audio .stem.mp4 file. One way is to use the python package, and the other way is to use shell script.
Table 4.1 Mapping of the separate audio stems encoded in a single audio .stem.mp4 format in MUSDB18.

<table>
<thead>
<tr>
<th>Mapping</th>
<th>Name of the Audio Stem</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:0</td>
<td>mixture</td>
</tr>
<tr>
<td>0:1</td>
<td>drums</td>
</tr>
<tr>
<td>0:2</td>
<td>bass</td>
</tr>
<tr>
<td>0:3</td>
<td>rest of the accompaniment</td>
</tr>
<tr>
<td>0:4</td>
<td>vocals</td>
</tr>
</tbody>
</table>

After decoding each of the stem.mp4 data, five audio stems are found for each song shown in Figure 4.2.

![Figure 4.2](image)

Figure 4.2 File System in a song folder after uncompressing it from .stem.mp4 format

4.2.2 Audio Data Processing

In deep learning, algorithms either take raw data or processed data. There is no need to select the features from the data as the algorithm will automatically learn the features from the data while training. In the case of audio source separation, either raw audio data (waveform data/time-domain data) or processed audio data (Time-Frequency domain) data can be used. The wave file format is uncompressed raw audio data in the time domain. Raw audio data is more prominent in size and takes considerable time to load and process, which slows down the training process and causes the deep model to be slower. The architecture of the proposed Bi-directional GRU is constructed to take input of the time-frequency domain data. To match with the input format of the Bi-directional GRU, time-domain raw audio data is converted to the time-frequency domain by using the Short Time Fourier Transformation.
4.2.2.1 Data Augmentation

The training data represents ten musical genres, but the test data represent only seven musical genres. The three genres missing in the test folder are Jazz, Pop, and Country. Analyzing the test data folder shows that each of these missing genres has at least three songs in the train folder. One song of each missing genre is transferred from the training folder to the testing folder to analyze the proposed system's separation audio quality genre-wise. As most training data comes from the Rock/Pop genre, three Rock/Pop genre music are transferred from the test folder to the train folder. This rearrangement of data helps to keep the same number of train and test data as before but allows to test each genre present in the training folder.

Figure 4.3 Two Channel (Stereo) Audio to Tensor

In the beginning, each audio stem is treated as a two-dimensional tensor array \((t \times C)\) where the time series value or samples \(t\) and the number of channels \(C\) (Figure 4.3). As each audio track of the dataset is stereophonic, the Channel value \(C\) will be two (2), where channel one is the left channel of the audio, and channel two is the right channel of the audio.
The length of each song and each stem is longer in time. If the whole song is transformed into a single spectrogram, the resolution of the spectrogram is low, which will create difficulties for the Neural Network to extract features and learn from the spectrogram. From Figure 4.4 it is seen that the spectrogram of the audio seems congested, and the frequency change over time is challenging to observe.

To train the neural network efficiently, better resolution of the frequency content in the spectrogram is required. For that purpose, each audio stem is sliced into five shorter time segments of equal length, referred to as audio chunks (Figure 4.5). The Zero-padding technique is used if the audio can not be divided equally into five shorter segments. The Zero-padding technique refers to adding zeros at the end of a sequence.

*librosa* is a well-known python library to load and process audio data. The audio data is loaded, converted to the time-frequency domain using its predefined STFT function,
Figure 4.5 Audio Segmentation in five equal length audio segment

and converted to dB scaled spectrogram.

```python
import librosa
```

## 4.2.2.2 Spectrogram and Tensor

After slicing each audio data into shorter time segment tensors, the audio slices are then converted to time-domain waveform data as spectrograms by using Short Time Fourier Transformation. Segmenting audio in the waveform domain provides better spectrogram resolution that can be observed in Figure 4.6.

The STFT uses a frame length of 2048 samples and a frame step or hop size of 512. Along with that FFT length of 4096 and Hann Window is used. So, in the STFT calculation, the frame length is divided into 4096 bins. The frequency resolution of the STFT is 21.53 Hz and time resolution is 46 millisecond.

After performing the Short-Time Fourier Transformation, the spectrogram is transformed into a three-dimensional tensor (t x f x c) format. Here t is time axis value...
Figure 4.6 Waveform and Spectrogram of a Single Audio Chunk

(X-axis), \( f \) is frequency axis value (Y-axis) and \( C \) is number of channels. For stereo, \( C \) is two (2), and for mono, \( C \) is one (1). To convert the spectrogram into a tensor, the spectrogram’s time axis is divided into 512 sections, and the frequency axis is divided into 2048 sections (Figure 4.7). So, the whole spectrogram of each audio channel is converted to a 512 X 2018 pixel image. So the whole spectrogram is divided into 1048576 bins, and the corresponding tensor has 512 rows, 2018 columns. Each bin represents the frequency content of the spectrogram. As the time axis is divided into 512 equal sections, the time difference of each bin of the spectrogram is equal. Here, dividing the \( f \) axis by the 2048 section is important to represent all the frequencies till 20 kHz. While processing the audio first time, the data is stored as cache memory. So, after the first time, the system used the saved data from memory for training.

In python, several libraries are available to convert time-domain to spectrogram
and convert the data to tensors. The tensorflow library is chosen for this project to accomplish the task. The tensorflow library every kind of data is considered as tensor. So, there will be no explicit steps required to convert data into tensors. It does the task by itself. TensorFlow normalizes dB values between 0 to 1. This normalization of dB values helps the neural network process and learn from the data properly. It takes time domain audio data as two dimensional tensor, convert into spectrogram and return spectrogram data as 3-Dimensional tensor. As tensor is a type of array, the popular NumPy library is required to manipulate the created tensor.

```python
import numpy as np
import tensorflow as tf
```

4.2.2.3 Flatten the Input Tensor

The tensor created after the Short-Time Fourier Transformation has n by n dimensionality. Before feeding the data into the network, the data is flattened. Flattening is the process of converting the multi-dimensional input tensor data into a 1-dimensional array tensor for inputting it to the next layer.

In this problem Flatten function flattens the three-dimensional input tensors into a single dimension, so you can model your input layer and build your neural network model, then pass those data into every single neuron of the model effectively. The Flatten layer is a utility layer that flattens an input of shape N X t X f X C to a simple vector output.
of shape N * (t X f X C), where N is the batch size. This converted tensor is used as the input for the bi-GRU.

Python library keras has the functionality named flatten that takes an n by n array and convert it to a 1-D array

```python
from tensorflow.keras.layers import Flatten
input_array = TimeDistributed(Flatten())(input_tensor)
```

4.3 Architecture of the Bi-directional Gated Recurrent Unit

Bi-directional GRU extends the traditional GRU to improve model performance on sequential data and sequential classification problems. Bi-directional GRU trains two GRU within a single layer instead of one GRU on the input sequence where all the timesteps are available. There are other reasons to select and implement Bi-directional GRU for audio source separation. GRU architecture is more superficial than its predecessor, LSTM. Combining two memory lines into one to get the information of the previous cells and combining two gates into one make GRU faster than LSTM without compromising its performance. Moreover, reducing memory lines and gates also reduces the number of
computational parameters; less computational complexity.

A Bi-directional GRU is an improved version of a GRU neural network with a two-sublayer structure. This two-sublayer architecture provides the output layer with the complete contextual information of the input information at every moment from both directions.

The input of the Bi-directional GRU is the 1-Dimensional flattened tensor in the form of \((t \times f \times C)\) format (Figure 4.8). With the batch size \(N\), the format of the flattened tensor is \(N \times (t \times f \times C)\). Each cell’s information of the flattened tensor is the input of the Bi-directional GRU. So, the first cell of the Bi-directional GRU receives the information on the time axis value, and the second cell gets the information on the frequency content. The third cell gets the channel information which tells the previous frequency content and time axis value come from the right channel or the left channel of the audio. Both forward and backward layer of the Bidirectional GRU receives the input data separately.

The forward layer cell, for example \(\overrightarrow{c}_1^{(t+1)}\) receives the frequency content information as input. It also gets the time axis value information from the previous cell \(\overrightarrow{c}_1^{(t)}\). By using Equation 3.15, the output of the forward layer cell \(\overrightarrow{c}_1^{(t+1)}\) can be written as

\[
\text{Figure 4.9 Cell Architecture of Bi-directional Gated Recurrent Unit}
\]
\[
\overleftarrow{c}_1^{\langle t+1 \rangle} = \Gamma_u \ast \overleftarrow{c}^{\langle t+1 \rangle} + (1 - \Gamma_u) \ast c^{\langle t \rangle}
\]  
(4.1)

The backward layer cell \( \overleftarrow{c}_1^{\langle t+1 \rangle} \) also gets the same frequency content information as input. But it gets the channel information and previous frequency content information from its previous cell \( \overleftarrow{c}_1^{\langle t+2 \rangle} \). By using Equation 3.15, the output of the backward layer cell \( \overleftarrow{c}_1^{\langle t+1 \rangle} \) can be calculated

\[
\overleftarrow{c}_1^{\langle t+1 \rangle} = \Gamma_u \ast \overleftarrow{c}^{\langle t+1 \rangle} + (1 - \Gamma_u) \ast c^{\langle t+2 \rangle}
\]  
(4.2)

From the previous two equations (4.2 and 4.3), it is seen that the forward cell is getting information from previous cells, and the backward cell is getting information from the future cells. By performing the dot product of the outputs of these two sublayers, the final result gets information on all previous timesteps and future timesteps.

\[
c_1^{\langle t+1 \rangle} = \overrightarrow{c}_1^{\langle t+1 \rangle} \ast \overleftarrow{c}_1^{\langle t+1 \rangle}
\]  
(4.3)

### 4.3.1 Implementing Gated Recurrent Unit

*Python* programming language made it easier to implement GRU using the *pytorch* and the *keras* library. Two ways can implement the GRU. One way is to exploit the computer’s Graphical Processing Unit (GPU) by using CuDNNGRU. For using CuDNNGRU, the computer must have a specific Nvidia graphics unit. Another way is to use the *GRU* method from the *keras* library. *GRU* in *keras* library only uses the CPU of the computer. As loading audio data and training neural networks consumes a lot of time, it is most beneficial to use the *CuDNNGRU* if GPU is available.

```python
import tensorflow as tf
from tensorflow.compat.v1.keras.layers import CuDNNGRU

if GPU is not available, GRU can be accessed from *keras*.

from tensorflow.keras.layers import GRU
```

### 4.3.2 Implementing Bidirectionality

Bidirectionality is the process of making the neural network of getting the sequence information in both directions, backward (future to past) or forward(past to future).
The bidirectionality of the GRU or any other RNN can only be implemented when all the timesteps (past and future steps) are available. As the proposed model uses only recorded audio, all the past and future information are available. As the nodes or units of the Bidirectional GRU use past and future information, the nodes or units are non-causal. Thus, the proposed system can not be used for real-time audio source separation. Bidirectionality helps to get improved performance for sequential data modeling. As the information flows from both directions, any cell can access the information from past and future cells. It is significant for the audio data because each audio data sample is dependent on the others. In the case of bidirectionality, neural networks connect two hidden sub-layers of opposite directions to the same layer and create a single output. The output layer can simultaneously access information from both past and future states by applying this method.

A Bi-directional GRU calculates the input sequence from the opposite direction to a hidden forward sequence and a backward hidden sequence. The final output vector is created by concatenating the forward and backward outputs.

In **python**, bidirectionality feature can be implemented by calling the `Bidirectional` function from the `keras` library. To perform bidirectionality correctly, the `return_sequences` and `return_state` must be set as `True` inside the `Bidirectional` function.

```python
from tensorflow.keras.layers import Bidirectional
Bidirectional(CuDNNGRU(units = 480,
    kernel_initializer=kernel_initializer, return_sequences=True,
    return_state=True))
```

### 4.3.3 Input Layer

The input layer works as an entry point in the neural network. In the input layer, no operation is done. It just receives the input and passes it to the hidden layer.

In this model, the input layer takes the flattened tensor and passes it to the hidden layer. The flattened tensor is given one at a time. As the song is divided into five shorter time segments of equal length, there will be five spectrogram tensors for each song. Five spectrogram tensors X batch size are the actual batch size, like this system’s stochastic gradient descent (SGD).
4.3.4 Hidden Layers

In RNN, all the calculations and predictions have occurred in the hidden layer. The bi-directional GRU hidden layer takes the flatten spectrogram tensor as input, which is 1-dimensional array, as timesteps and uses the previous cell’s information to predict the future result. This exact prediction is also carried out to the next cell of GRU as the last input. The backward layer also takes the same tensor input of the forward layer as timesteps. Still, it uses the information of the previous cell opposite the forward layer to predict the future result. This exact prediction is also carried out to the next cell of GRU as the last input.

![Figure 4.10 Proposed Network Architecture](image)

The hidden layers predict the input audio stem from the audio mixture. In this model, there are four hidden layers. Each hidden layer has two sub-layers: the forward and the backward layers. For this model, each sub-layer has 480 units or neurons. The first hidden sub-layer takes the input and passes the information in the forward direction, and the second hidden sub-layer takes the input from the input layer and passes the information to the backward direction. The outputs from both layers are concatenated, and the new output is used as the input for the next hidden layer. This process follows till before the dense layer.

4.3.5 Dense Layers

A Dense layer receives all predictions from the preceding layer to all its neurons, every neuron imparting one prediction to the following layer. It’s the maximum simple layer in neural networks. In this GRU implementation, Timedistributed dense layer is used.
Timedistributed dense layer facilitates the preservation of one-to-one relationships in the input and output.

```python
from tensorflow.keras.layers import Dense, TimeDistributed
```

### 4.3.6 Output

After passing through the dense layer, the shaped tensor is the model’s outcome. The final output is a 3-dimensional tensor with \((t \times f \times C)\) shape (Figure 4.11. The final output is a 512 X 2014 X 2 tensor which signifies that the tensor has 512 rows, 2014 columns and has 2 axis for stereo music. Each value of the tensor represent the frequency content of an single music instrument. The output tensor then converted back to the spectrogram.

![Figure 4.11 Output Tensor](image)

#### 4.3.7 Combining the Bi-directional GRU networks

Each instrument is trained in a separate Bi-directional GRU network against the mixture, and a spectrogram for each instrument is estimated. The prediction of each network is used at the final stage to separate the sources from the audio mixture by using a *Norbert* multichannel Wiener filter [35]. A Wiener filter is used to statistically estimate the desired signal from a noisy signal. It uses Linear Time-Invariant (LTI) filtering technique to isolate and estimate the desired signal. Wiener filter calculation assumes that the signal and noise processes are second-order stationary. The second-order stationarity signal is the signal that variance, mean, and autocovariance do not change.
with time. *Norbert* implementation of the Wiener filter designed for time-frequency domain audio filtering. As the output of each bi-directional GRU produces a spectrogram, i.e., time-frequency domain audio, the Wiener filter is the preferred choice for audio source separation for this system. This filtering implementation requires the power or magnitude estimation of the spectrograms for all the audio sources present in the audio mixture. It treats the audio mixture as a noisy signal and each instrument’s signal as a target or desired signal. By deploying *ExpectationMaximization* algorithm and using multichannel information, this filtering technique uses the estimated spectrograms of each instrument generated by the Bi-directional GRU to separate audio sources from the audio mixture.

### 4.3.8 Hyper-parameters

To facilitate the learning of the algorithm, some hyper-parameters are used in this model. In this proposed model, few standard hyper-parameters are used. The dropout rate, which decides the number of dropping neurons from the network in the training phase, is used in this model. The dropout rate for the proposed system is 0.5.

The loss function determines the difference between the predicted output and actual output. Based on the loss function, the model tunes its parameter to converge the difference of predicted output and actual output. Defining the loss function for audio source separation is a complicated task. Several researchers use their custom loss function to optimize the efficiency of the audio source separation model. In his master’s thesis, Enric Gusó Muñoz [41] evaluated some of the commonly used loss functions in the audio source separation. The proposed neural network is architected based on the magnitude spectrogram. As the L1 Magnitude loss function is utilized in many well-known audio source separation systems, it is used as the loss function of the proposed system. In the L1 magnitude loss function, the difference between the absolute magnitude value of the separated audio stem spectrogram is subtracted from the absolute magnitude value of the original audio stem spectrogram.

\[
L_{freq} = \frac{1}{NK} |Y - \tilde{Y}|
\]

where \( Y \) is the absolute magnitude value of the original audio stem spectrogram, \( \tilde{Y} \) is the absolute magnitude value of the separated audio stem spectrogram, \( N \) is the number of samples, \( K \) is the number of audio Stems.

Optimization is a technique to maximize the learning of a deep learning model. Based
on the loss function value, the deep learning model tunes its parameter to minimize the loss while training and thus maximize its learning. In this proposed model, Stochastic Gradient Descent (SGD) optimizer is used to train the model.

The learning rate determines at which rate the learnable parameters of the algorithm update their estimation parameter. The learning rate for this proposed system is $1e^{-4}$.

4.4 Training

The train folder of the MUSDB18 [45] dataset has 100 full-length songs to train the model. From these 100 songs, 84 songs from the training folder are used for training. Sixteen songs are used for validating the learning of the model. The training and validation data size is relatively small, so the mini-batch mode is selected as the preferred batch size. In this model, the batch size is the number of songs passing through the algorithm to train it each time. The batch size for this system is four. So, four songs will comprise a batch and pass in each epoch before the model parameters are updated. As there are 84 songs for training, $(84 / 4) = 21$ batches of data are used to train the model, and sixteen songs comprise $(16 / 4) = $four batches to validate the learning of the model. As the total number of batches is twenty-five, a minimum of twenty-five epochs is required to pass the entire training dataset through the model. However, the system is trained for 7600 epochs, and the training took 52 hours to complete. So, the entire dataset (training and validation) passed 304 times through the system to train and validate the system.

4.5 Testing

This proposed audio source separation system separates four audio stems from the audio mixture. The audio mixture or song is fed into the system in the testing phase, and the whole audio is sliced into five shorter time segments of equal length. Then each time-domain signal segment is converted into a Power Spectrum or spectrogram segment by using Short-Time Fourier transformation. These five segmented spectrograms are fed into the Bi-directional GRU network sequentially. The network separates the four audio stems from each spectrogram and creates four separate spectrograms for four audio stems from each spectrogram. So, five spectrograms total produces a total of 20 spectrograms, five spectrograms for each audio instrument. These spectrograms contain the magnitude information of the audio stems in dB and lost the phase information. The segments of
each instrument then join together and create a whole spectrogram for the respective instrument. The spectrograms are then converted to audio using Inverse Short-Time Fourier Transformation (ISTFT).

4.6 Architecture of the proposed system

The whole architecture and mechanism of the proposed system are illustrated in Figure 4.12.
Figure 4.12 Architecture of the proposed system.
Chapter 5  |  Results

Several methods are available to evaluate the quality of the separated music from the audio mixture. These available resources to assess audio quality can be categorized as subjective audio testing, objective audio testing, and evaluation tools and software for audio source separation. These evaluation processes require the original and separated audio stems to compare either mathematically or subjectively. Objective Audio Testing methods such as Perceptual Evaluation of Speech Quality (PESQ), Perceptual Objective Listening Quality Analysis (POLQA) require software license from a specific vendor, which is out of the scope of this thesis. In the result section, the quality of the separated audio stems is evaluated in three ways. First, the visual analysis and comparison of the waveform and spectrogram of the original and separated audio stems are performed. Then MusEval [20] tool, which is specially designed for MUSDB18 dataset performance evaluation, is used as an evaluation tool to evaluate the separation quality. Finally, the subjective audio testing result is presented.

5.1 Training and Validation Loss

From the training and validation loss graph (Figure 5.1), it is seen that from 6000 to 7600 epochs, the change of loss is stable. From that graph, it is concluded that the neural network reaches some kind of convergence point. As it is a prototype version of the source separation algorithm, different learnable parameters will be tuned for better convergence.
Comparing waveform, spectrogram and Energy

The modified test folder of the MUSDB18 dataset has 50 full-length songs representing ten musical genres. To compare the spectrogram and waveform of the original audio stems and separated audio stems using the Bi-directional GRU model, a song from each genre is arbitrarily selected. The comparison is made instrument-wise.

The energy of the original audio mixture and separated audio mixture are compared to verify if there is any energy loss or energy added to the separated audio stems while separating from the audio mixture by the audio source separation system. The loss of energy will signify the loss of audio data while pre-processing the audio and separating the audio stems from the audio mixture. Similarly, the addition of energy will signify the introduction of noise while pre-processing the audio and separating the audio stems from the audio mixture. All the separated audio stems are added together to get the separated audio mixture. According to the theory of energy, the new separated audio mixture should contain the same energy as the original audio mixture. The Root Mean Square (RMS) method is used to calculate the energy of the original and separated audio mixture. RMS energy is the root-mean-square energy of a signal, calculated by squaring each sample value of the signal, finding the arithmetic mean of those squared sample values, and then taking the square root of the result [40]. The following equation
(Equation 5.1) is used to calculate the RMS energy.

\[
RMS\ Energy = \sqrt{\frac{1}{N} \sum_{i=1}^{n} |x_i|^2}
\]  \hspace{1cm} (5.1)

where \( x_i \) is the sample of the signal ranging from 1 to \( n \) and \( N \) is the total number of samples.

### 5.2.1 Singer/Songwriter Genre

For the Singer/Songwriter genre, *A Classic Education* − *NightOwl* song from the test folder is chosen.

#### 5.2.1.1 Drums Stem

From the spectrograms (Figure 5.2), it is seen that the proposed model separated the drums stem in the frequency range of 4 kHz to 15 kHz from the audio mixture effectively. But it fails to segregate the audio in the low-frequency range from 105 Hz to 4 kHz in the whole audio stem.

![Figure 5.2 Comparing the Spectrogram of Drums Stem (a) original and (b) separated of Singer/Songwriter genre music.](image)

**Figure 5.2** Comparing the Spectrogram of Drums Stem (a) original and (b) separated of Singer/Songwriter genre music.
In the waveform view (Figure 5.3), it is seen that the separated drums stem has the same shape as the original drums stem, but the separated drums stem has a lower amplitude than the original drums stem throughout the audio.

5.2.1.2 Vocals Stem

From the spectrograms (Figure 5.4), it is seen that the proposed model separated the vocals stem from the audio mixture successfully. But the network also extracted audio that is not part of the original audio stem. This problem occurred in the middle of the audio - from 1.37 minutes to 2.09 minutes, but after 2.09 minutes, the network separated the vocals stem near perfectly. The model also fails to separate some high amplitude and low amplitude frequencies from the audio mixture.
Figure 5.4 Comparing the Spectrogram of Vocals Stem (a) original and (b) separated of Singer/Songwriter genre music.

Figure 5.5 Comparing the waveform of original (a) and separated (b) Vocals Stem of Singer/Songwriter genre music.

The waveform view supports the spectrogram data. While comparing the original vocals stem (Figure 5.5(a)) and separated vocals stem (Figure 5.5(b)), it is observed that the original vocals stem does not have any audio from 1:37 minute to 2:09 minute, but the separated vocals stem contains audio on that time. Undoubtedly, the proposed model extracted that part of audio from other stems. It is also noticed that the separated vocals stem has a higher amplitude than the original vocals stem.

5.2.1.3 Bass Stem

By comparing the spectrograms (Figure 5.6), it is seen that the proposed model separated the bass stem from the audio mixture near perfectly. Though it segregates the
bass stem properly, it also adds the low frequency of the other audio stems.

![Original Bass Stem Spectrogram](image1) ![Separated Bass Stem Spectrogram](image2)

**Figure 5.6** Comparing the Spectrogram of Bass Stem (a) original and (b) separated of Singer/Songwriter genre music.

![Original Bass Stem Waveform](image3) ![Separated Bass Stem Waveform](image4)

**Figure 5.7** Comparing the waveform of original (a) and separated (b) Bass Stem of Singer/Songwriter genre music.

The waveform view also aligns with the spectrogram data. By comparing both the waveform, the original bass stem Figure 5.7(a), and the separated bass stem Figure 5.7(b), it is seen that both original and separated bass stem have identical shape. The separated bass stem has a higher amplitude than the original bass stem in fewer places.

### 5.2.1.4 Rest of the Accompaniment Stem

By comparing the rest of the accompaniment stem spectrograms (Figure 5.8), it is seen that the trained network smoothly separates the rest of the accompaniment stem
from the audio mixture. Most of the data is lost. Specifically, the network segregated audio properly in the low-frequency end - from 0 Hz to 5 kHz, but the segregation is irregular after that frequency range.

**Figure 5.8** Comparing the Spectrogram of Rest of the accompaniment Stem (a) original and (b) separated of Singer/Songwriter genre music.

**Figure 5.9** Comparing the waveform of original (a) and separated (b) Rest of the accompaniment Stem of Singer/Songwriter genre music.

The waveform view also aligns with the spectrogram data. By comparing both the waveform, the original rest of the accompaniment stem (Figure 5.9(a)), and the separated rest of the accompaniment stem (Figure 5.9(b)), it is seen that both original and separated bass stem have identical shape. The separated bass stem has lost some audio data from 1:35 minutes to 2:05 minutes.
5.2.1.5 Energy Comparison

From Figure 5.10, it is seen that both the original audio mixture and the separated audio mixture have the same RMS energy. While separating the audio stems from the audio mixture, no energy is added or lost. So, the total energy of the audio mixture is preserved.

![Graph showing RMS energy comparison](image)

**Figure 5.10** Comparing the RMS energy of the original and separated audio mixture of Singer/Songwriter genre music.

5.2.2 Rock Genre

For the Rock genre, Angels In Amplifiers – I'm Alright song from the test folder is chosen.

5.2.2.1 Drums Stem

From the spectrograms (Figure 5.11), it is seen that the proposed model separated the drums stem in the frequency range of 3 kHz to 17.5 kHz from the audio mixture effectively. But it fails to segregate the audio in the low-frequency range from 15 Hz to 3
kHz in the audio. This problem also occurs in the low-frequency range - from 0 Hz to 2.5 kHz. In the whole audio, the proposed model fails to separate audio frequency with high amplitude (in the 7.5 kHz and 12.5 kHz).

Figure 5.11 Comparing the Spectrogram of Drums Stem (a) original and (b) separated of Rock genre music.

Figure 5.12 Comparing the waveform of original (a) and separated (b) Drums Stem of Rock genre music.

The waveform view (Figure 5.12), also support the loss of amplitude seen in the spectrogram. The separated drums stem has a lower amplitude than the original drums stem throughout the audio.
5.2.2.2 Vocals Stem

From the spectrograms (Figure 5.13), it is seen that the proposed model separated the vocals stem from the audio mixture successfully. But the network also extracted audio that is not part of the original audio stem. This problem occurred at the beginning of the audio - from 0.00 minutes to 0.09 minutes and from 1:20 minute to 1:43 minute. Other than these two occurrences, the network separated the vocals stem perfectly.

Figure 5.13 Comparing the Spectrogram of Vocals Stem (a) original and (b) separated of Rock genre music.

Figure 5.14 Comparing the waveform of original (a) and separated (b) Vocals Stem of Rock genre music.

The waveform view supports the spectrogram data. While comparing the original vocals stem (Figure 5.14(a)) and separated vocals stem (Figure 5.14(b)), it is observed
that the original vocals stem does not have any audio from 0:00 minute to 0:09 minute, from 0:53 minute to 1:06 minute and from 1:20 minute to 1:43 minute. But the separated vocals stem contains audio from the 0:00 minute. Undoubtedly, the proposed model extracted parts of audio from another stem.

5.2.2.3 Bass Stem

By comparing the spectrograms (Figure 5.15), it is seen that the proposed model separated the bass stem from the audio mixture poorly. Though it segregates the low-frequency part of the bass stem, it also separated the other frequency contents which are not part of the actual bass stem.

Figure 5.15 Comparing the Spectrogram of Bass Stem (a) original and (b) separated of Rock genre music.

Figure 5.16 Comparing the waveform of original (a) and separated (b) Bass Stem of Rock genre music.
The waveform view also aligns with the spectrogram data. By comparing both the waveform, the original bass stem Figure 5.16(a), and the separated bass stem Figure 5.16(b), it is seen that more audio data are extracted in the separated audio stem than the original audio stem. The separated bass stem has a lower amplitude than the original bass stem.

5.2.2.4 Rest of the Accompaniment Stem

By comparing the rest of the accompaniment stem spectrograms (Figure 5.17), it is seen that the trained network separated data from 0 Hz to 5 kHz. But it failed to separate the stem from the 5kHz and beyond. Most of the data is lost in this frequency range.

![Figure 5.17 Comparing the Spectrogram of Rest of the Accompaniment Stem (a) original and (b) separated of Rock genre music.](image)

By comparing the original waveform (Figure 5.18(a)) and the separated audio waveform (Figure 5.18(b)) of rest of the accompaniment stem, it is seen that the separated audio stem waveform has almost the identical shape of the original audio stem, but the amplitude of the separated waveform is lower than the original audio stem. It is seen throughout the audio waveform.

5.2.2.5 Energy Comparison

From Figure 5.19, it is seen that both the original audio mixture and the separated audio mixture have the same RMS energy. While separating the audio stems from the audio mixture, no energy is added or lost. So, the total energy of the audio mixture is preserved.
Figure 5.18 Comparing the waveform of original (a) and separated (b) Rest of the Accompaniment Stem of Rock genre music.

5.3 Audio Source Separation Toolbox

There are only a few tools available to evaluate the quality of separated audio where both the ground truth (original audio stems) and separated audio stems are available. For this project, the MusEval tool [19, 22] is used in the evaluation. MusEval is a popular package to compare the quality of the original audio with the separated audio. This tool is available in the python library and is used to evaluate the segregated audio quality based on Source to Distortion Ratio (SDR), Source to Artifact Ratio (SAR), Source to Interference Ratio (SIR), and Image to Spatial distortion Ratio (ISR) metrics.

The metrics that are calculated using this evaluation tool are inspired by the classical Signal to Noise Ratio equation

\[
SNR = 10 \log_{10} \left( \frac{||s||^2}{||n||^2} \right) dB
\]

(5.2)

where \( s \) is the power of the wanted part of the signal and \( n \) is the unwanted part of the signal, i.e. noise.

The evaluation algorithm used orthogonal projection instead of decomposing the signal to compute the signal and error terms. Orthogonal projection is a form of parallel projection to illustrate a three-dimensional object in two dimensions. In Orthogonal projection, all the projected lines are orthogonal to the projection plane [].

If \( s_j \) is the source, then the estimated source \( \hat{s}_j \) will be
Figure 5.19 Comparing the RMS energy of the Original and the separated audio mixture of Rock genre music.

\[
\hat{s}_j = (s_{\text{target}} + e_{\text{total}})
\]  

(5.3)

Here \(\hat{s}_j\) is the estimated source, \(s_{\text{target}}\) is a version of \(s_j\) altered by an allowed distortion, and \(e_{\text{total}}\) is the allowed noise and distortion introduced by the source separation system. By decomposing the total error into three terms,

\[
e_{\text{total}} = (e_{\text{interf}} + e_{\text{noise}} + e_{\text{artif}})
\]  

(5.4)

where \(e_{\text{interf}}\) is the interference error, \(e_{\text{noise}}\) is the noise error, and \(e_{\text{artif}}\) is the artifacts. The equation (5.3) then can be modified as

\[
\hat{s}_j = (s_{\text{target}} + e_{\text{interf}} + e_{\text{noise}} + e_{\text{artif}})
\]  

(5.5)

If \(P_s\) is an orthogonal projector to the source signal, \(P_{s,n}\) is the orthogonal projector of the noisy source signal, and \(P_{s,j}\) is the projector of the source signal, then all the terms in the equation (5.5) can be calculated.
$s_{\text{target}}$ which is a version of $s_j$ altered by an allowed distortion, will be

$$s_{\text{target}} = P_{sj} \hat{s}_j$$  \hspace{1cm} (5.6)

$e_{\text{interf}}$ which is the error term due to interference, will be

$$e_{\text{interf}} = P_s \hat{s}_j - P_{sj} \hat{s}_j$$ \hspace{1cm} (5.7)

$e_{\text{noise}}$ which is the error term due to the cause of the additive noise, will be

$$e_{\text{noise}} = P_{s,n} \hat{s}_j - P_s \hat{s}_j$$ \hspace{1cm} (5.8)

$e_{\text{artif}}$ which is the error term due to the cause of the numerical artifacts by the audio source separation system, will be

$$e_{\text{artif}} = \hat{s}_j - P_{s,n} \hat{s}_j$$ \hspace{1cm} (5.9)

Signal to Distortion Ratio (SDR), the primary evaluation method of the separated source in the tool, calculates the log-ratio between signal energy and the overall error energy.

$$SDR = 10 \log_{10}(\frac{||s_{\text{target}}||^2}{||e_{\text{interf}} + e_{\text{noise}} + e_{\text{artif}}||^2}) dB$$ \hspace{1cm} (5.10)

Signal to Interference Ratio (SIR) computes the log-ratio between signal energy and interference energy.

$$SIR = 10 \log_{10}(\frac{||s_{\text{target}}||^2}{||e_{\text{interf}}||^2}) dB$$ \hspace{1cm} (5.11)

Signal to Artifacts Ratio (SAR) quantifies the log-ratio between signal energy and noise energy in the audio.

$$SAR = 10 \log_{10}(\frac{||s_{\text{target}} + e_{\text{interf}} + e_{\text{noise}}||^2}{||e_{\text{artif}}||^2}) dB$$ \hspace{1cm} (5.12)

Image to Spatial distortion Ratio (ISR) evaluates the Spatial distortion in the audio.

$$ISR = 10 \log_{10}(\frac{||s_{\text{target}}||^2}{||e_{\text{noise}}||^2}) dB$$ \hspace{1cm} (5.13)

The classic Signal to Noise Ratio (SNR) will be
\[ SNR = 10 \log_{10} \left( \frac{\|s_{\text{target}} + e_{\text{interf}}\|^2}{\|e_{\text{noise}}\|^2} \right) dB \]  \hspace{1cm} (5.14)

The evaluation tool compares the separated audio stem with the original computationally and calculates the metrics. For example, to calculate the drum stem’s SDR metric, the SDR value of all drum stems is calculated separately. Then all the values are added together and calculate the mean of the drum stem. The mean is the SDR score for drums stem. The SDR values of the four audio stems are averaged to calculate the overall SDR score.

**Table 5.1 Evaluation of Separated audio quality of the proposed system on the MUSDB18 test dataset by MUSEVAL**

<table>
<thead>
<tr>
<th>Name of the Audio Stem</th>
<th>SDR</th>
<th>SAR</th>
<th>SIR</th>
<th>ISR</th>
</tr>
</thead>
<tbody>
<tr>
<td>vocals</td>
<td>7.89</td>
<td>13.08</td>
<td>6.41</td>
<td>10.35</td>
</tr>
<tr>
<td>drums</td>
<td>7.53</td>
<td>12.86</td>
<td>5.32</td>
<td>9.98</td>
</tr>
<tr>
<td>bass</td>
<td>6.08</td>
<td>11.74</td>
<td>6.22</td>
<td>3.96</td>
</tr>
<tr>
<td>rest of the accompaniment</td>
<td>3.81</td>
<td>7.31</td>
<td>4.66</td>
<td>6.19</td>
</tr>
<tr>
<td>overall</td>
<td>6.33</td>
<td>11.25</td>
<td>5.51</td>
<td>7.12</td>
</tr>
</tbody>
</table>

After separating all the fifty songs from the MUSDB18 [45] dataset test folder, the MUSEVAL [20] tool is used to calculate different separated audio metrics to evaluate the proposed system’s source separation quality. Table 5.1 illustrates the audio source separation quality of the proposed system evaluated using MusEval [20]. The higher SDR, SAR, SIR, and ISR values signify better separated audio quality. In contrast, lower values of these metrics indicate poor audio quality. Higher values of the source separation metrics represent that the signal is clean and stronger than the distortion, artifacts, interference, and spatial distortion. Based on the separated audio quality, the performance of an audio source separation model is judged.

The Vocals and Drums stems have relatively higher SDR, SAR, SIR, and ISR values than the Bass stem and the rest of the accompaniment stem. From the waveform and spectrogram analysis, it is seen that for most musical genres, the separation quality of the bass stem and the rest of the accompaniment stem is relatively poorer than the vocals stem and the drums stem. The source separation metrics also signify and support the same.
Table 5.2 *Source to Distortion ratio of separated audio stems on MUSDB18 by the proposed model and other reference models*

<table>
<thead>
<tr>
<th>Audio Stem</th>
<th>Proposed Model SDR</th>
<th>Highest SDR</th>
<th>Lowest SDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>vocals</td>
<td>7.89</td>
<td>9.0 (KUIELab-MDX-Net [30])</td>
<td>3.25 (STL2 [60])</td>
</tr>
<tr>
<td>drums</td>
<td>7.53</td>
<td>8.76 (Hybrid Demucs [15])</td>
<td>2.49 (Wavenet [36])</td>
</tr>
<tr>
<td>bass</td>
<td>6.08</td>
<td>8.24 (Hybrid Demucs [15])</td>
<td>4.22 (STL2 [60])</td>
</tr>
<tr>
<td>rest of the accompaniment</td>
<td>3.81</td>
<td>5.95 (KUIELab-MDX-Net [30])</td>
<td>0.54 (Wavenet [36])</td>
</tr>
<tr>
<td>overall</td>
<td>6.33</td>
<td>7.68 (Hybrid Demucs [15])</td>
<td>3.23 (STL2 [60])</td>
</tr>
</tbody>
</table>

Table 5.2 shows the SDR value comparison of the proposed model and other well-known audio source separation models. While comparing with the highest and lowest SDR values of the MUSDB18 benchmark page [54], it is seen that the system performs well with a low number of training epochs. The proposed audio source separation model trained only 52 hours (7600 epochs) without extra training data. Other source separation model Spleeter [23] trained for seven (7) days. Some source separation models with higher SDR values (Spleeter [23], Demcus V2 [16]) even use more training data to train the model.

### 5.4 Subjective Audio Testing

Subjective audio testing is a method to evaluate the quality of the separated audio subjectively. The expert user performs subjective audio testing. So, the user’s opinion can be helpful to determine the quality of the separated audio. Under subjective audio testing, the Mean opinion score (MOS) is the most popular. The International Telecommunication Union (ITU) and Audio Engineering Society - European Broadcasting Union (AES-EBU) have written recommendations to perform subjective audio testing [66]. Other subjective audio tests are Descriptive analysis (Free Choice Profiling, IVP, Flash Profile) and Discrimination testing (ABX, MUSHRA, Paired Comparison).

In subjective audio testing, a few participants are selected. Each participant first listens to an original audio stem of a song and then listens to the same audio stem separated by the proposed system of the same song. Then the participant subjectively compares the audio quality of the separated stem with the original audio stem and scores the separated audio stem based on its quality on a scale of 0 to 5, where 0 is the
When all participants provide a score of a single audio stem, all the scores are added. Then the mean of the score is calculated. That mean score is the MOS score for that audio stem of that particular song. When all the scores for a particular stem across all songs are summed up, and a mean is derived, that mean score is the system’s MOS score for that stem. This process continues for all four stems for all songs. Each song has four stems. So, each participant listens to eight stems (four original stems and four separated audio stems) in a stem-wise pair for a single song.

For example, a participant listens to the original drums stem and then listens to the same song’s separated drums stem. The participant then subjectively compares the audio quality of the separated drums stem with the original drums stem and provides a score to the separated drums stem on a scale of 0 to 5 based on the quality. When all of the participants have given a score for the drums stem of the song, the scores are summed up, and the mean is calculated. That mean score is the MOS score for the drums stem of that particular song. When all the MOS scores of the drums stem across all songs are summed up, and a mean is derived, that mean score is the system’s MOS score for the drums stem.

For the MOS score collection, all the music pieces and a MOS score sheet with scoring instructions were stored in an online drive. No direct or indirect communication was made with any participants. One online post was published in a Facebook group. So, there was no interaction and intervention with any participants. Moreover, no personal information of the participants was recorded in the MOS score sheet or by any means. The participants provided two ways to provide consent. First, when downloading the audio and MOS score sheet from the online drive. Second, when they uploaded the MOS score sheet to the online drive. In the Facebook post, participants were requested to keep the volume of their device low as possible and adjust the volume of their device as required to listen to the audio properly. After receiving the five responses for each of the tasks, the online drive was removed and the Facebook post was deleted.

The original audio stems of the MUSDB18 [45] dataset and separated audio stems by the proposed system are available for this project, and the standard procedure of finding the MOS score is followed. The five participants (N = 5) listened to eighty (80) audio stems; forty (40) of them are original, and forty (40) of them are separated audio stems using the proposed model. Table 5.3 shows the MOS score of the separated audio by the proposed audio source separation model on the MUSDB18 [45] dataset.
<table>
<thead>
<tr>
<th>Name of the Audio Stem</th>
<th>MOS Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>vocals</td>
<td>3.1</td>
</tr>
<tr>
<td>drums</td>
<td>2.8</td>
</tr>
<tr>
<td>bass</td>
<td>2.4</td>
</tr>
<tr>
<td>rest of the accompaniment</td>
<td>0.9</td>
</tr>
<tr>
<td>overall</td>
<td>2.3</td>
</tr>
</tbody>
</table>

Table 5.4 compares the MOS score of the separated audio by the proposed audio source separation model and other audio source separation models on the MUSDB18 dataset [45,54].

<table>
<thead>
<tr>
<th>Model</th>
<th>MOS Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed Model</td>
<td>2.30</td>
</tr>
<tr>
<td>KUIELAB-MDX-Net [30]</td>
<td>2.86</td>
</tr>
<tr>
<td>Hybrid Demucs (v3) [15]</td>
<td>2.83</td>
</tr>
<tr>
<td>Demucs (v2) [16]</td>
<td>2.37</td>
</tr>
</tbody>
</table>

Six songs are chosen from youtube and converted into .mp3 format to evaluate the system’s efficiency on real-world audio. To prevent bias, these songs are selected from different genres - Indian classical music, Bengali music, and American country music. Then the proposed system is used to separate the four audio stems from the music. Few observations are noted while separating these songs. The percussion instruments are separated as drums stem, especially in Indian music. If any of the three music instruments (vocals, drums, bass) are not played, a respective stem is still created without audio. Though the original music stems are not available, the participants were still asked to evaluate the audio quality of the separated audio stems. Table 5.5 shows the MOS score of the separated audio by the proposed audio source separation model on the Youtube music.
Table 5.5 Mean Opinion Score (MOS) of the separated audio stems from Youtube

<table>
<thead>
<tr>
<th>Name of the Audio Stem</th>
<th>MOS Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>vocals</td>
<td>3.4</td>
</tr>
<tr>
<td>drums</td>
<td>3.4</td>
</tr>
<tr>
<td>bass</td>
<td>1.9</td>
</tr>
<tr>
<td>rest of the accompaniment</td>
<td>0.7</td>
</tr>
<tr>
<td>overall</td>
<td>2.35</td>
</tr>
</tbody>
</table>

The overall MOS score of the separated audio stems of the proposed model on the MUSDB18 dataset (Table 5.3), and Youtube music (Table 5.5) is almost similar. The vocals stem separated from the Youtube music got a slightly higher MOS score than the vocals stem separated from the MUSDB18 dataset, and the separated bass stem of the MUSDB18 dataset got a slightly higher MOS score than the separated bass stem of the Youtube music. The MOS scores of the other two audio stems are almost the same for the MUSDB18 dataset and the Youtube music.
Chapter 6  |  Conclusion and Future Work

The primary outcome of this thesis is implementing Bi-directional Gated Neural Unit (Bi-directional GRU) for audio source separation and assessing the quality of the separated audio. For the first time implemented, this Recurrent Neural Network (RNN) shows better performance with less training than other successful audio source separation models. This thesis also provides a stem-wise comparison of original audio and separate audio in Time-Frequency and Time Domain, which helps the reader understand the system’s performance. The procedure to conduct Mean Opinion Score is described in detail. A broad literature review is provided to understand the technology and algorithms used in the audio source separation.

While working on this project, several technical problems were encountered. These problems, their consequences, and how to avoid these issues are documented in the observation section. These observations are regarding python programming and python libraries that hindered the progress of this work. This observation section will be helpful for someone working on any audio projects in the python programming language. The limitation of this project and future improvement of this model are described in their respective sections.

6.1 Limitations

Due to time and resource limitations, the proposed system was not trained long enough. The system is designed to work on any personal computer regardless of whether the Graphical Processing Unit (GPU) is available or not. As a result, much training could not be done on the system. A GPU-based system can perform much faster in training, requiring a graphical processing unit present in the computer. Source separation systems with higher SDR values were trained for longer time. For example Spleeter [23]
was trained for seven (7) days, where this system was trained only 52 hours (7600 epochs). This low number of training impacted the separation result. But with the more inadequate training, the design performs well comparatively.

A significant limitation of the work thus far is that this model requires the instrument name prefixed before training and testing. One of the prerequisites of this algorithm is to provide the instrument names before processing the audio. A stem is created with no audio when an instrument is not present in the audio mixture. Moreover, only three musical instruments can be separated using this model. All the other instruments are split into a single stem.

For all the musical genres, some specific audio stems are separated better than other audio stems from the audio mixture. One reason behind the issue is due to the genre-specific and overall less training data. For example, after data augmentation, only two Jazz music samples were used to train the model. As a result, the model did not get enough training data to analyze the structure of songs from the Jazz genre. Another reason for this problem can be generalizing a source separation model for every musical genre. Every music genre has different fundamental characteristics, musical structure, and musical techniques [47,49]. On the other hand, the deep learning model acts as a non-linear adaptive filter that aims to create a generalized model or filter to filter a specific audio stem from the audio mixture regardless of the musical genre [37,42]. As a result, the source separation model separates certain audio stems better than the other audio stems.

From SDR, SAR, SIR, and ISR scores, and the spectrogram comparison, it is seen that the vocals stem and the drum stem have higher metrics values and better separation quality than the bass stem and the rest of the accompaniment stem. The same problem is also noticed in the other submitted models on the MUSDB18 benchmark page [54]. Hung and Lerch [12] also reported on the low SDR score of the separated bass stem in their research. In most musical genres, the separated bass stem contains audio from the other three stems, more specifically from the rest of the accompaniment stem. The bass stem has only low-frequency content from 0 Hz to 3 kHz. All other audio stems also have the frequency content in that range. As the bass stem lies in the low-frequency range of the audio mixture and overlaps with the audio of the kick drum, it is difficult for the computer system to extract the underlying structure and representation of the bass stem from the audio mixture [12,28]. Several pieces of research are going on to address the low SDR and SAR scores issue of the separated bass stem.

There could be a few possible reasons behind the poor separation quality and source
separation metrics score of the bass stem and the rest of the accompaniment section. The rest of the accompaniment stem contains all the musical instruments except vocals, drums, and bass. But the numbers or types of musical instruments in the rest of the accompaniment section are not fixed. For example, in one music sample, the rest of the accompaniment stem only contains guitar and piano. But in other music, the rest of the accompaniment stem has guitar, harp, flute, and piano. Less training data and the ambiguity of the instruments playing in the rest of the accompaniment stem can make it difficult for the audio source separation system to create a generalized separation model or filter for the rest of the accompaniment stem.

Overall, the training dataset contains only one hundred full-length songs to train the system. Some models, for example, Spleeter [23], D3Net [65], TAK2 [64], Demicus V2 [16] trained the model with more training data and reported separation quality improvement.

6.2 Future Work

The obstacles described in the limitations section enable the scope of the improvements of the proposed audio source separation model. In addition to this separation model, a new automatic musical instrument identification system can be developed to determine the audio sources, i.e., musical instruments present in the audio, without assigning the stem names manually.

There are several audio source separation systems in the MUSDB18 Benchmark website mentioned that they have used other custom datasets in addition to the MUSDB18 [45] dataset to train their models. As MUSDB18 [45] dataset is smaller in size, more training with different datasets will enrich the learning of the model and reduce bias which enables the opportunity for better audio separation. More training will also help the model learn more accurately and separate instruments from the audio mixture more efficiently. Most of the audio source separation models, including this thesis dealing with studio-recorded music, are split into four instrument categories, vocals, drums, bass, and others. But in the music domain, there are several commonly used musical instruments such as piano, guitar, etc. Future research can be navigated towards extending the number of instrument categories to improve the model to separate these sources. Effects, also known as Fx, such as reverb, delay, decay, sustain are often used in the music tracks to make the music more lively. So far, no work has been done to separate effects from the music tracks. Furthermore, no research has been done to identify the effect of Fx on training and testing an audio source separation system. The MedleyDB dataset contains separate
Although the MUSDB18 [45] dataset is widely used for the music source separation training and evaluation, this dataset limits the training and assessment of audio source separation system only on American music and three widely used instruments. A dataset with multicultural music and more musical instruments is a scope to create a new dataset.

At the beginning of the system, audio is converted into a spectrogram, a form of an image. A Graphical Processing Unit (GPU) can be used to load and process these audio and image data faster. By using GPU, the model can perform more training quickly, and it will enhance the model’s training and validation capabilities and provide better separation. A GPU will also enable the opportunity to train the model with more data in a shorter period.

Several types of research proposed algorithms for denoising audio sources with promising results. A new project can adopt one of these algorithms for smoothing the audio stems after separation.

### 6.3 Conclusion

The thesis adopted and implemented a Bi-directional GRU for audio source separation. This newly proposed system performs better comparatively than other well-known systems with lesser training. Though there are some limitations to the system, this thesis can guide the implementation of a more improved and complete audio source separation system using Bi-directional GRU. The result from this approach seems promising. After experimenting and tuning the model’s parameters and training the system with more data, it will show even more favorable results.

New technology always attracts researchers to introduce and implement it in a new field. The curiosity to deploy a new deep learning algorithm in the audio source separation leads this project, and in the end, this thesis paper is presented.
Appendix A
Result

A.1 Jazz Genre

For the Jazz genre, Matthew Entwistle – Don't You Ever song from the test folder is chosen.

A.1.1 Drums Stem

From the spectrograms (Figure A.1), it is seen that the proposed model separated the drums stem in the frequency range of 2.5 kHz to 15.5 kHz from the audio mixture effectively. But it fails to segregate the audio in the high-frequency range from 0.5 Hz to 3 kHz in most audio. The stem also contains audio data from 0:30 minute to 1:10 minute that is not the part of the drums stem.

Figure A.1 Comparing the spectrograms of Drums stem, (a) original and (b) separated of Jazz genre music.
Figure A.2 Comparing the waveform of (a) original and (b) separated Drums Stem of Jazz genre music.

The waveform view supports the spectrogram data. While comparing the original vocals stem (Figure A.2(a)) and separated vocals stem (Figure A.2(b)), it is observed that the original vocals stem does not have any audio from 0:00 minute to 1:10 minute, but the separated vocals stem contains audio from the 0:00 minute. Undoubtedly, the proposed model extracted that part of audio from another stem. It is also noticed that the high amplitude part of the vocals stem is not adequately extracted.

A.1.2 Vocals Stem

From the spectrograms (Figure A.3), it is seen that the proposed model separated the vocals stem from the audio mixture successfully. But the network also extracted audio at the beginning that is not part of the original audio stem. Besides that, the model irregularly segregates audio in the high-frequency end.
Figure A.3 Comparing the Spectrogram of Vocals Stem (a) original and (b) separated of Jazz genre music.

Figure A.4 Comparing the waveform of original (a) and separated (b) Vocals Stem of Jazz genre music.

The waveform view supports the spectrogram data. While comparing the original vocals stem (Figure A.4(a)) and separated vocals stem (Figure A.4(b)), it is observed that the original vocals stem does not have any audio from 0:00 minute to 0:30 minute, but the separated vocals stem contains audio from the 0:00 minute. Other than that, the separated vocals stem has an almost identical shape to the original vocals stem.

A.1.3 Bass Stem

By comparing the spectrograms (Figure A.5), it is seen that the proposed model separated the bass stem from the audio mixture poorly. Though it segregates the low-
frequency part of the bass stem, it also adds the low frequency and mid-frequency content of the other audio stems.

Figure A.5 Comparing the Spectrogram of Bass Stem (a) original and (b) Separated of Jazz genre music.

Figure A.6 Comparing the waveform of original (a) and separated (b) Bass stem of of Jazz genre music.

The waveform view also aligns with the spectrogram data. By comparing both the waveform, the original bass stem Figure A.6(a), and the separated bass stem Figure A.6(b), it is seen that more audio data are extracted in the separated audio stem than the original audio stem. This anomaly is seen from the 0:00 minute to 1:13 minute in the separated audio stem. The separated bass stem has almost an identical shape to the original bass stem. But the separated bass stem has a much lower amplitude than the original bass stem.
A.1.4 Rest of the Accompaniment Stem

By comparing the rest of the accompaniment stem spectrograms (Figure A.7), it is seen that the trained network failed to separate most of the audio of the rest of the accompaniment stem. Most of the data is lost. Specifically, the network segregated audio properly in the low-frequency end - from 0 Hz to 2.5 kHz, but the segregation is irregular after that frequency range. The network almost failed to isolate audio beyond the 2.5 kHz frequency range.

![Figure A.7 Comparing the Spectrogram of Rest of the accompaniment Stem (a) original and (b) separated of Jazz genre music.](image)

By comparing the original waveform (Figure A.8(a)) and the separated audio waveform (Figure A.8(b)), it is seen that the model lost audio throughout the rest of the
accompaniment audio stem. It extracted the audio correctly in most places, but overall, the model failed to separate audio for the rest of the accompaniment stem properly. The amplitude level of the separated stem is much lower than the original audio stem.

### A.1.5 Energy Comparison

From Figure A.9, it is seen that both the original audio mixture and the separated audio mixture have the same RMS energy. While separating the audio stems from the audio mixture, no energy is added or lost. So, the total energy of the audio mixture is preserved.

![RMS Energy of the Original and Separated Audio Mixture](image)

**Figure A.9** Comparing the RMS energy of the original and separated audio mixture of Jazz genre music.

### A.2 Electronic Genre

For the Electronic genre, *PR – Oh No* song from the test folder is chosen.
A.2.1 Drums Stem

From the spectrograms (Figure A.10), it is seen that the proposed model separated the drums stem in the frequency range of 13 kHz to 17 kHz from the audio mixture effectively. But it fails to segregate the rest of the drums from the audio mixture. The separation is irregular in the rest of the drums stem.

![Figure A.10](image)

**Figure A.10** Comparing the Spectrogram of Drums Stem (a) original and (b) separated of Electronic genre music.

The waveform view (Figure A.11) supports the spectrogram data. While comparing the original drums stem (Figure A.11(a)) and separated drums stem (Figure A.11(b)), it is observed that the original drums stem does not have any audio from 0:00 minute to 0:05
minute, but the separated drums stem contains audio from the 0:00 minute. Undoubtedly, the proposed model extracted that part of audio from another stem. The separated drums stem lost data throughout the audio that can be seen from the amplitude level of the separated drums stem.

A.2.2 Vocals Stem

From the spectrograms (Figure A.12), it is seen that the proposed model separated the vocals stem from the audio mixture poorly. The network also extracted audio that is not part of the original audio stem. This problem is seen throughout the audio.

![Figure A.12 Comparing the Spectrogram of Vocals Stem (a) original and (b) separated of Electronic genre music.](image)

![Figure A.13 Comparing the waveform of original (a) and separated (b) Vocals Stem of Electronic genre music.](image)

Figure A.12: Comparing the Spectrogram of Vocals Stem (a) original and (b) separated of Electronic genre music.

Figure A.13: Comparing the waveform of original (a) and separated (b) Vocals Stem of Electronic genre music.
The waveform view supports the spectrogram data. While comparing the original vocals stem (Figure A.13(a)) and separated vocals stem (Figure A.13(b)), it is observed that the original vocals stem does not have any audio from 0:00 minute to 0:05 minute, but the separated vocals stem contains audio from the 0:00 minute. Undoubtedly, the proposed model extracted that part of audio from another stem. This problem is noticed throughout the audio.

A.2.3 Bass Stem

By comparing the spectrograms (Figure A.14), it is seen that the proposed model separated the low-frequency part of the bass stem from the audio mixture properly. Though it segregates the low-frequency part of the bass stem, it also adds the frequency contents from 2 kHz to 13 kHz of the other audio stems.

![Figure A.14 Comparing the Spectrogram of Bass Stem (a) original and (b) separated of Electronic genre music.](image)

The waveform view also aligns with the spectrogram data. By comparing both the waveform, the original bass stem Figure A.15(a), and the separated bass stem Figure A.15(b), it is seen that more audio data are extracted in the separated audio stem than the original audio stem. This anomaly is seen throughout the audio stem.

A.2.4 Rest of the Accompaniment Stem

By comparing the rest of the accompaniment stem spectrogram (Figure A.16), it is seen that the trained network separated the low-frequency part of the rest of the
The waveform view also aligns with the spectrogram data. By comparing both the waveform, the original rest of the accompaniment stem (Figure A.17(a)), and the separated rest of the accompaniment stem (Figure A.17(b)), it is seen that more audio data are extracted in the separated audio stem than the original audio stem. This anomaly is seen throughout the audio stem.
A.2.5 Energy Comparison

From Figure A.18, it is seen that both the original audio mixture and the separated audio mixture have the same RMS energy. While separating the audio stems from the audio mixture, no energy is added or lost. So, the total energy of the audio mixture is preserved.

A.3 Pop/Rock Genre

For the Pop/Rock genre, AMContra – HeartPeripheral song from the test folder is chosen.

A.3.1 Drums Stem

From the spectrograms (Figure A.19), it is seen that the proposed model separated the drums stem in the frequency range of 2.5 kHz to 15.5 kHz from the audio mixture effectively. But it fails to segregate the audio in the high-frequency range from 15 kHz to 17.5 kHz in most audio. This problem also occurs in the low-frequency range - from 0 Hz to 2.5 kHz.

In the waveform view, the periodicity in both waveforms can be seen. At the beginning of each period, the proposed model failed to extract high amplitude audio of the drum stem.
Figure A.18 Comparing the RMS energy of the original and separated audio mixture of Electronic genre music.

A.3.2 Vocals Stem

From the spectrograms (Figure A.21), it is seen that the proposed model separated the vocals stem from the audio mixture successfully. But the network also extracted audio that is not part of the original audio stem. This problem occurred at the beginning of the audio - from 0.00 minutes to 1.40 minutes, but after 1.40 minutes, the network separated the vocals stem near perfectly.
Figure A.19 Comparing the Spectrogram of Drums Stem (a) original and (b) separated of Pop/Rock genre music.

Figure A.20 Comparing the waveform of original (a) and separated (b) Drums Stem of Pop/Rock genre music.

Figure A.21 Comparing the Spectrogram of Vocals Stem (a) original and (b) separated of Pop/Rock genre music.
Figure A.22 Comparing the waveform of original (a) and separated (b) Vocals Stem of Pop/Rock genre music.

The waveform view supports the spectrogram data. While comparing the original vocals stem (Figure A.22(a)) and separated vocals stem (Figure A.22(b)), it is observed that the original vocals stem do not have any audio from 0:00 minute to 0:20 minute but the separated vocals stem contains audio from the 0:00 minute. Undoubtedly, the proposed model extracted that part of audio from another stem. It is also noticed that the high amplitude part of the vocals stem is adequately extracted. Still, the segregation is performed in the wrong fashion for the low amplitude part.

A.3.3 Bass Stem

By comparing the spectrograms (Figure A.23), it is seen that the proposed model separated the bass stem from the audio mixture poorly. Though it segregates the low-frequency part of the bass stem, it also adds the low frequency of the other audio stems. Moreover, the model separates audio, which is not part of the actual bass stem. Listening to the bass stem shows that the low-frequency audio of the other three stems is separated as a bass stem. There is also faint audio of the guitar and vocals.

The waveform view also aligns with the spectrogram data. By comparing both the waveform, the original bass stem Figure A.24(a), and the separated bass stem Figure A.24(b), it is seen that more audio data are extracted in the separated audio stem than the original audio stem. This anomaly is seen throughout the audio stem. If compared with the separated drums stem waveform, the reason behind this discrepancy can be described. The kick drum has a low frequency, and the bass also has a low frequency. So, the proposed model extracted the kick drum audio with the bass stem.
A.3.4 Rest of the Accompaniment Stem

By comparing the Rest of the Accompaniment stem spectrograms (Figure A.25), it is seen that the trained network failed to separate the stem. Most of the data is lost. Specifically, the network segregated audio properly in the low-frequency end - from 0 Hz to 2.5 kHz, but the segregation is irregular after that frequency range. Though the network separated a few audios in the mid-frequency range, it almost failed to isolate any audio in the high-frequency range.

By comparing the original waveform (Figure A.26(a)) and the separated audio waveform (Figure A.26(b)), it is seen that the model lost audio at the beginning of the
"other" audio stem. It extracted the audio correctly in a few places, but overall, the model failed to separate audio for the "other" stem properly. This problem affects the other three stems’ audio quality, especially the bass stem.

A.3.5 Energy Comparison

From Figure A.27, it is seen that both the original audio mixture and the separated audio mixture have the same RMS energy. While separating the audio stems from the audio mixture, no energy is added or lost. So, the total energy of the audio mixture is preserved.
A.4 Reggae Genre

For the Reggae genre, *Arise – Run Run* song from the test folder is chosen.

A.4.1 Drums Stem

From the spectrograms (Figure A.28), it is seen that the proposed model separated the drums stem almost perfectly. There are minimal discrepancies in the frequency range of 20.5 Hz to 6 kHz.

The waveform view (Figure A.29) also supports the information of the spectrogram. The amplitude level of the original and separated drums audio stems are almost the same, with few exceptions.

A.4.2 Vocals Stem

From the spectrograms (Figure A.30), it is seen that the proposed model separated the vocals stem from the audio mixture successfully. But the network also extracted
Figure A.28 Comparing the Spectrogram of Drums Stem (a) original and (b) separated of Reggae genre music.

Figure A.29 Comparing the waveform of original (a) and separated (b) Drums Stem of Reggae genre music.

audio that is not part of the original audio stem. This problem occurred at the end of the audio - from 2.56 minutes to 3.09 minutes.
Figure A.30 Comparing the Spectrogram of Vocals Stem (a) original and (b) separated of Reggae genre music.

Figure A.31 Comparing the waveform of original (a) and separated (b) Vocals Stem of Reggae genre music.

The waveform view supports the spectrogram data. While comparing the original vocals stem (Figure A.31(a)) and separated vocals stem (Figure A.31(b)), it is observed that the original vocals stem does not have any audio from 0:00 minute to 0:13 minute and from 2.56 minutes to 3.09 minutes. But the separated vocals stem contains audio from the 0:00 minute. Undoubtedly, the proposed model extracted part of audio from another stem.
A.4.3 Bass Stem

By comparing the spectrograms (Figure A.32), it is seen that the proposed model separated the bass stem from the audio mixture poorly. Though it segregates the low-frequency part of the bass stem, it also adds the low frequency of the other audio stems. Moreover, the model separates audio, which is not part of the actual bass stem. Listening to the bass stem shows that the low-frequency audio of the other three stems is separated as a bass stem. There is also faint audio of the other instruments.

Figure A.32 Comparing the Spectrogram of Bass Stem (a) original and (b) separated of Reggae genre music.

Figure A.33 Comparing the waveform of original (a) and separated (b) Bass Stem of Reggae genre music.

The waveform view also aligns with the spectrogram data. By comparing both the waveform, the original bass stem Figure A.33(a), and the separated bass stem Figure
A.33(b), it is seen that more audio data are extracted in the separated audio stem than the original audio stem. This anomaly is seen throughout the audio stem.

### A.4.4 Rest of the Accompaniment Stem

By comparing the rest of the accompaniment stem spectrograms (Figure A.34), it is seen that the trained network is failed to separate the stem. Most of the data is lost. Specifically, the network segregated audio properly in the low-frequency end - till 2.5 kHz, but the segregation is irregular after that frequency range. The network is failed to isolate any audio in the mid-frequency and high-frequency range.

![Comparing the Spectrogram of Rest of the accompaniment Stem](image)

**Figure A.34** Comparing the Spectrogram of Rest of the accompaniment Stem (a) original and (b) separated of Reggae genre music.

![Comparing the waveform of original and separated](image)

**Figure A.35** Comparing the waveform of original (a) and separated (b) Rest of the accompaniment Stem of Reggae genre music.
By comparing the original waveform (Figure A.35(a)) and the separated audio waveform (Figure A.35(b)), it is seen that the model lost audio at the beginning of the rest of the accompaniment audio stem. It extracted the audio correctly in a few places, but overall, the model failed to separate audio for the rest of the accompaniment stem properly. This problem affects the other three stems’ audio quality.

A.4.5 Energy Comparison

From Figure A.36, it is seen that both the original audio mixture and the separated audio mixture have the same RMS energy. While separating the audio stems from the audio mixture, no energy is added or lost. So, the total energy of the audio mixture is preserved.

![RMS Energy Comparison](image.png)

Figure A.36 Comparing the RMS energy of the Original and separated audio mixture of Reggae genre music.
A.5 Heavy Metal Genre

For the Heavy Metal genre, *We Fell From The Sky – Not You* song from the test folder is chosen.

A.5.1 Drums Stem

From the spectrograms (Figure A.37), it is seen that the proposed model separated the drums stem in the frequency range of 4.7 kHz to 17 kHz from the audio mixture effectively. But it fails to segregate the audio in the low-frequency range from 110 Hz to 4.7 kHz in most part of the audio.

The waveform view (Figure A.38) also supports the information of the spectrogram.
The separated drums audio stem has lower amplitude in all parts of the audio.

### A.5.2 Vocals Stem

From the spectrograms (Figure A.39), it is seen that the proposed model separated the vocals stem from the audio mixture successfully. But the network also extracted audio that is not part of the original audio stem. This problem occurred in the whole audio. It also irregularly extracts the audio in the low-frequency range from 0 Hz to 5 kHz.

**Figure A.39** Comparing the Spectrogram of Vocals Stem (a) original and (b) Separated

**Figure A.40** Comparing the waveform of original (a) and Separated (b) Vocals Stem

The waveform view supports the spectrogram data. While comparing the original vocals stem (Figure A.40(a)) and separated vocals stem (Figure A.40(b)), it is observed
that the original vocals stem does not have any audio from 0:00 minute to 0:20 minute but the separated vocals stem contains audio from the 0:00 minute. This problem also occurs in the other parts of the audio. Undoubtedly, the proposed model extracted that part of audio from another stem. It is also noticed that the high amplitude part of the vocals stem is adequately extracted. Still, the segregation is performed in the wrong fashion for the low amplitude part.

A.5.3 Bass Stem

By comparing the spectrograms (Figure A.41), it is seen that the proposed model separated the bass stem from the audio mixture along with the other frequency contents that are not the part of the bass stem. Though it segregates the low-frequency part of the bass stem, there is some irregularity in the low-frequency part.

![Figure A.41 Comparing the Spectrogram of Bass Stem (a) original and (b) Separated](image)

The waveform view also aligns with the spectrogram data. By comparing both the waveform, the original bass stem Figure A.42(a), and the separated bass stem Figure A.42(b), it is seen that the separated bass stem has higher amplitude than the original bass stem in some parts. Throughout the audio, data loss is observed in the separated bass stem.

A.5.4 Rest of the Accompaniment Stem

By comparing the rest of the accompaniment stem spectrograms (Figure A.43), it is seen that the trained network successfully separates the audio of the rest of the accompaniment stem from 0 Hz to 6.5 kHz. Beyond that frequency range, the proposed
Figure A.42 Comparing the waveform of original (a) and Separated (b) Bass Stem model failed to segregate any rest of the accompaniment stem audio from the audio mixture.

Figure A.43 Comparing the Spectrogram of Rest of the accompaniment Stem (a) original and (b) Separated

By comparing the original waveform (Figure A.44(a)) and the separated audio waveform (Figure A.44(b)), it is seen that the amplitude of the rest of the accompaniment stem has higher amplitude than the original rest of the accompaniment stem. The model extracted audio that is not part of the actual rest of the accompaniment stem that can be observed from the added samples in the waveform. This anomaly is seen throughout the audio.
A.5.5 Energy Comparison

From Figure A.45, it is seen that both the original audio mixture and the separated audio mixture have the same RMS energy. While separating the audio stems from the audio mixture, no energy is added or lost. So, the total energy of the audio mixture is preserved.

A.6 Country Genre

For the Country genre, Music Delta – Country – 1 song from the test folder is chosen.

A.6.1 Drums Stem

From the spectrograms (Figure A.46), it is seen that the proposed model failed to separate the drums stem from the audio mixture. It has extracted audio that is not the part of actual drums stem. It fails to segregate the audio in the low-frequency range from 85 Hz to 2.5 kHz in most audio. This problem also occurs in the other frequency ranges.

The waveform view supports the spectrogram data. In the waveform view (Figure A.47), the periodicity in both waveformrs can be seen. But the periodicity of the separated drums stem waveform is different from the original drums stem.
Figure A.45 Comparing the RMS energy of the Original and Separated Audio Mixture.

A.6.2 Vocals Stem

From the spectrograms (Figure A.48), it is seen that the proposed model separated the vocals stem from the audio mixture successfully. But the network also extracted audio that is not part of the original vocals stem. This problem is observed in the middle and at the end of the vocals stem. The proposed model also failed to extract high amplitude frequency content.
Figure A.46 Comparing the Spectrogram of Drums Stem (a) original and (b) Separated

Figure A.47 Comparing the waveform of original (a) and Separated (b) Drums Stem

Figure A.48 Comparing the Spectrogram of Vocals Stem (a) original and (b) Separated
Figure A.49 Comparing the waveform of original (a) and Separated (b) Vocals Stem

The waveform view supports the spectrogram data. While comparing the original vocals stem (Figure A.49(a)) and separated vocals stem (Figure A.49(b)), it is observed that the original vocals stem does not have any audio from 0:13 minute to 0:17 minute and 0.30 minute to 0.35 minute. But the separated vocals stem contains audio in both sections. Undoubtedly, the proposed model extracted that part of the audio from another stem.

A.6.3 Bass Stem

By comparing the spectrograms (Figure A.50), it is seen that the proposed model separated the bass stem from the audio mixture and also extracted audio that is not the part of the bass stem. Though it segregates the actual low-frequency contents of the bass stem, it also adds the low frequency and high-frequency contents of the other audio stems.

The waveform view also aligns with the spectrogram data. By comparing both the waveform, the original bass stem Figure A.51(a), and the separated bass stem Figure A.51(b), it is seen that there is periodicity in the both separated and original bass stem. But the shape of the separated bass stem has dissimilarity in comparison with the original bass stem.

A.6.4 Rest of the Accompaniment Stem

By comparing the rest of the accompaniment stem spectrograms (Figure A.52), it is seen that the trained network failed to separate most of the audio. Most of the data is
lost. Specifically, the network segregated audio properly in the low-frequency end - from 0 Hz to 5 kHz, but it failed to isolate any audio beyond the 5 kHz frequency range.

By comparing the original waveform (Figure A.53(a)) and the separated audio waveform (Figure A.53(b)), it is seen that audio samples are lost throughout the rest of the accompaniment audio stem. The amplitude level of the separated stem is lower than the original stem.

### A.6.5 Energy Comparison

From Figure A.54, it is seen that both the original audio mixture and the separated audio mixture have the same RMS energy. While separating the audio stems from the audio mixture, no energy is added or lost. So, the total energy of the audio mixture is
A.7 Rap Genre

For the Rap genre, \textit{ANiMAL} – \textit{Rockshow} song from the test folder is chosen.

A.7.1 Drums Stem

From the spectrograms (Figure A.55), it is seen that the proposed model separated the drums stem in the frequency range of 5 kHz to 17 kHz from the audio mixture.
effectively. But it irregularly segregates the audio in the low-frequency range from 105 kHz to 5 kHz in most audio. It also separated audio that is not the part of the drums stem at the end, from 2:30 minutes to 3:00 minutes.

![RMS Energy of the Original and Separated Audio Mixture](image1)

Figure A.54 Comparing the RMS energy of the Original and Separated Audio Mixture.

![Comparing the Spectrogram of Drums Stem](image2)

Figure A.55 Comparing the Spectrogram of Drums Stem (a) original and (b) Separated of Rap genre.
In the waveform view (Figure A.56), it is seen that the separated audio waveform has a lower amplitude than the original drums stem throughout the audio. The separated drums stem has added audio to the end which was also noticed in the spectrogram view.

### A.7.2 Vocals Stem

From the spectrograms (Figure A.57), it is seen that the proposed model separated the vocals stem from the audio mixture successfully. But the network also extracted audio that is not part of the original audio stem. This problem occurred at the beginning of the audio - from 0.00 minutes to 0.15 minutes and in the middle from 1:21 minutes to 1:43 minutes. But after 1.43 minutes, the network separated the vocals stem near-perfectly.
Figure A.57 Comparing the Spectrogram of Vocals Stem (a) original and (b) Separated of Rap genre.

Figure A.58 Comparing the waveform of original (a) and Separated (b) Vocals Stem of Rap genre.

The waveform view supports the spectrogram data. While comparing the original vocals stem (Figure A.58(a)) and separated vocals stem (Figure A.58(b)), it is observed that the original vocals stem does not have any audio from 0:00 minute to 0:15 minute, but the separated vocals stem contains audio from the 0:00 minute. This problem also occurred in the rest of the audio. Undoubtedly, the proposed model extracted that part of audio from another stem. It is also noticed that the high amplitude part of the vocals stem is adequately extracted. Still, the segregation is performed in the wrong fashion for the low amplitude part.
A.7.3 Bass Stem

By comparing the spectrograms (Figure A.59), it is seen that the proposed model separated the bass stem from the audio mixture poorly. Throughout the audio, the frequency contents are lost. The proposed model failed to segregate high amplitude frequency content throughout the audio.

![Figure A.59 Comparing the Spectrogram of Bass Stem (a) original and (b) Separated of Rap genre.](image)

The waveform view also aligns with the spectrogram data. By comparing both the waveform, the original bass stem Figure A.60(a), and the separated bass stem Figure A.60(b), it is seen that the amplitude of the separated bass stem is lower than the original bass stem. This anomaly is seen throughout the audio stem.

![Figure A.60 Comparing the waveform of original (a) and Separated (b) Bass Stem of Rap genre.](image)
A.7.4 Rest of the Accompaniment Stem

By comparing the rest of the accompaniment stem spectrograms (Figure A.61), it is seen that the trained network failed to separate the rest of the accompaniment stem from the audio mixture. Most of the data is lost. Specifically, the network segregated audio properly in the low-frequency end - from 0 Hz to 3 kHz, but the segregation is irregular after that frequency range. Though the network separated a few audios in the mid-frequency range, it almost failed to isolate any audio in the high-frequency range.

![Figure A.61 Comparing the Spectrogram of Rest of the accompaniment Stem (a) original and (b) Separated of Rap genre.](image1)

![Figure A.62 Comparing the waveform of original (a) and Separated (b) Rest of the accompaniment Stem of Rap genre.](image2)

By comparing the original waveform (Figure A.62(a)) and the separated audio waveform (Figure A.62(b)), it is seen that the data is lost throughout the audio. The
amplitude of the separated rest of the accompaniment is lower than the original rest of the accompaniment audio stem.

### A.7.5 Energy Comparison

From Figure A.63, it is seen that both the original audio mixture and the separated audio mixture have the same RMS energy. While separating the audio stems from the audio mixture, no energy is added or lost. So, the total energy of the audio mixture is preserved.

![RMS Energy of the Original and Separated Audio Mixture](image)

**Figure A.63** Comparing the RMS energy of the Original and Separated Audio Mixture of Rap genre.

### A.8 Pop Genre

For the Pop genre, *Music Delta – Disco* song from the test folder is chosen.
A.8.1 Drums Stem

From the spectrograms (Figure A.64), it is seen that the proposed model separated the drums stem in the frequency range of 5.5 kHz to 17.3 kHz from the audio mixture effectively. But it fails to properly segregate the audio in the low-frequency range from 105 Hz to 5.5 kHz in the whole audio stem.

![Figure A.64 Comparing the Spectrogram of Drums Stem (a) original and (b) Separated.](image)

In the waveform view (Figure A.65), it is seen that the separated drums stem has the same shape as the original drums stem, but the separated drums stem has a lower amplitude than the original drums stem throughout the audio.

![Figure A.65 Comparing the waveform of original (a) and Separated (b) Drums Stem.](image)
A.8.2 Vocals Stem

From the spectrograms (Figure A.66), it is seen that the proposed model separated the vocals stem from the audio mixture successfully. But the network also extracted frequency contents that are not part of the original audio stem. This problem occurred from the frequency range of 5.3 kHz to 17.3 kHz. But the network separated the vocals stem near perfectly from the frequency range of 0 Hz to 5.3 kHz.

Figure A.66 Comparing the Spectrogram of Vocals Stem (a) original and (b) Separated.

The waveform view supports the spectrogram data. While comparing the original vocals stem (Figure A.67(a)) and separated vocals stem (Figure A.67(b)), it is observed that the original vocals stem does not have any audio at the beginning, but the separated vocals stem contains audio from the 0:00 minute. This anomaly is also noticed in other
parts of the audio. Undoubtedly, the proposed model extracted that part of audio from another stem. It is also noticed that the amplitude of the separated vocals stem is lower than the amplitude of the original vocals stem.

### A.8.3 Bass Stem

By comparing the spectrograms (Figure A.68), it is seen that the proposed model separated the bass stem from the audio mixture poorly. Throughout the audio, the frequency contents are lost. The proposed model failed to segregate high amplitude frequency content throughout the audio.

![Original Bass Stem Spectrogram](image)

![Separated Bass Stem Spectrogram](image)

**Figure A.68** *Comparing the Spectrogram of Bass Stem (a) original and (b) Separated.*

![Original Bass Stem Waveform](image)

![Separated Bass Stem Waveform](image)

**Figure A.69** *Comparing the waveform of original (a) and Separated (b) Bass Stem.*

The waveform view also aligns with the spectrogram data. By comparing both the waveform, the original bass stem (Figure A.69(a)), and the separated bass stem Figure
A.69(b), it is seen that the amplitude of the separated bass stem is lower than the original bass stem. This anomaly is seen throughout the audio stem.

**A.8.4 Rest of the Accompaniment Stem**

By comparing the rest of the accompaniment stem spectrograms (Figure A.70), it is seen that the trained network separates the rest of the accompaniment stem properly. Most of the data of the rest of the accompaniment stem are segregated. Specifically, the network segregated audio properly in the low-frequency end - from 0 Hz to 5 kHz, but the segregation is irregular after that frequency range.

![Original Rest of the Accompaniment Stem Spectrogram](image1)

![Separated Rest of the Accompaniment Stem Spectrogram](image2)

**Figure A.70** Comparing the Spectrogram of Rest of the accompaniment Stem (a) original and (b) Separated.

![Original Rest of the Accompaniment Stem Waveform](image3)

![Separated Rest of the Accompaniment Stem Waveform](image4)

**Figure A.71** Comparing the waveform of original (a) and Separated (b) Rest of the accompaniment Stem.
By comparing the original waveform (Figure A.71(a)) and the separated audio waveform (Figure A.71(b)), it is seen that the amplitude of the separated rest of the accompaniment stem is lower than the original rest of the accompaniment stem throughout the audio.

### A.8.5 Energy Comparison

From Figure A.72, it is seen that both the original audio mixture and the separated audio mixture have the same RMS energy. While separating the audio stems from the audio mixture, no energy is added or lost. So, the total energy of the audio mixture is preserved.

![RMS Energy of the Original and Separated Audio Mixture](image)

**Figure A.72** Comparing the RMS energy of the Original and Separated Audio Mixture.
Appendix B
Observations

During this thesis work, a few unique problems have been encountered, and some of them are conceptual and occurred while implementing the system in the Python environment. Another problem is related to the python library. One of the problems encountered while loading the audio data using the librosa library of the python programming language. librosa library is well known for its features, and it has an immense impact on the audio community. This library is used to load and process audio, audio data visualization, beat tracking, creating spectrogram, etc. But unlike other python audio libraries - scipy or Wave, while loading the data, librosa does not compute the sampling frequency of the original audio data by itself. Instead, either it relies on the predefined value of the sampling frequency, which is 22050 Hz, or the programmer needs to set the sampling frequency value. So, the programmer needs to know the audio sampling frequency before loading the audio using librosa.load. Figure 6.1 visualizes the issue. According to the document of librosa, librosa.load function will either use the sampling frequency defined by the programmer or re-sampled the audio with the predefined sampling frequency of 22050 Hz. It is a huge pitfall for the programmer if they do not know this shortcoming of librosa.load function, leading to the more significant

![Figure B.1 Sampling Frequency Discrepancy in librosa library](image)

Figure B.1 Sampling Frequency Discrepancy in librosa library

123
problems in the rest of the work. Figure 6.1 shows the sampling frequency in the *librosa* and *scipy* library of the same audio file. *scipy* library load and calculate the sampling frequency where *librosa* re-sampled the audio at 22050 Hz.

This sampling frequency problem has considerable implications while calculating and using other signal processing algorithms that use sample frequency. In this project, a spectrogram is used for data processing. This sampling frequency problem causes discrepancy while creating the spectrogram of the audio data.

Figure B.2 and Figure B.3 visualized the problem more clearly. By comparing both the figure, it is observed that the highest frequency (Y-axis) of Figure B.2 is around 10500 Hz, and it discarded the frequency above that range where the audio has an actual frequency over 16 kHz. It holds the Nyquist sampling frequency theorem. This problem act like a low pass filter that cuts the frequency above 10500 Hz. Figure B.3 shows the correct spectrogram of the audio where the actual sampling frequency is defined while loading the audio in the *librosa.load* function. The spectrogram shows the right value in the Y-axis (Figure B.3).
Another problem faced is while converting the spectrogram into a tensor. Spectrograms are images, and image processing books and online lessons, advised and recommended to take width and height of the tensor is 256 (0 to 255) while converting image to tensor. In the case of the audio spectrogram, this assumption does not hold. By assigning a value with 256 in the height of the spectrogram, frequencies above 3000 Hz will be abundant and will act as a low pass filter.

In this thesis, while creating tensor in \((t \times f \times C)\) format, the tensor frequency axis value \(f\) was taken 1024 in the beginning. The resulting spectrogram keeps the value in the frequency axis up to 11 kHz. Figure B.4 shows the problem more clearly. Here, the system rejected the frequency above 11 kHz. This problem also acts as a low pass filter.

Another way to tackle the issue is to reduce the FFT size while converting audio to the spectrogram. For example, the FFT size 4096 is used in this thesis while converting
time-domain data to Time-Frequency domain data (i.e., spectrogram). The selected FFT size maps the full audio bandwidth to 2048 positive bins or frequency samples. To represent 2048 frequency samples, the height of the tensor must be 2048. On the other hand, if the FFT size is 512, the FFT size maps the full audio bandwidth to 256 positive bins or frequency samples. In that case, the height of the tensor can be 256. Though the higher FFT size is computationally expensive, reducing the FFT size will decrease the spectral resolution [2, 56]. The RNN cells treat each frequency sample as a pixel of an image and learn the underlying pattern from the pixel. Higher spectral resolution will help the RNN cells to learn about the frequency content more accurately and this will impact the separation quality of the audio source separation system. For audio applications, it is fairly common to take FFT size 4096.

As an experiment, in this thesis work, the audio dimensionality reduction technique using wavenet and PCA was used on the dataset to reduce the size of the dataset. Some researchers used audio dimensionality reduction techniques before training the source separation system for machine learning algorithms. The audio data dimensionality
Reduction technique was used at the very beginning of the thesis. However, the reduced dataset speeds up the training process but negatively impacts while separating audio in the testing phase. Results obtained from the study, although preliminary, show that the audio dimensionality reduction technique for reducing the size of the audio data makes it faster to train a deep learning model but does not provide exemplary performance in the testing when separating audio. This is one of the outcomes of this thesis.
Appendix C
IRB

C.1 HRP -310 WORKSHEET Human Research Determination
WORKSHEET: Human Research Determination

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>DATE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>HRP-310</td>
<td>02/01/2022</td>
<td>1 of 2</td>
</tr>
</tbody>
</table>

The purpose of this worksheet is to provide support for individuals in determining whether an activity is Human Research or how it is regulated. This worksheet is to be used. It does not need to be completed or retained.¹

1. **Research as Defined by DHHS Regulations**: (Check if "Yes").
   - Is the activity an investigation? (Investigation: A searching inquiry for facts; detailed or careful examination.)
   - Is the investigation systematic? (Systematic: Having or involving a system, method, or plan.)
   - Is the systematic investigation designed to develop or contribute to knowledge? (Designed: observable behaviors used to develop or contribute to knowledge. Develop: to form the basis for a future contribution. Contribute: to result in, Knowledge: truths, facts, information.)
   - Is the knowledge the systematic investigation is designed to develop or contribute generalizable? (Generalizable: Universally or widely applicable.)

2. **Human Subject Under DHHS Regulations**: (Check if "Yes").
   - Is the investigator conducting the Research gathering information or biospecimens about living individuals?

3. **Human Subject Under DHHS Regulations**: (Check if "Yes").
   - Will the investigator use, study, or analyze information or biospecimens obtained through either of the following mechanisms (Specify which mechanism(s) apply, if yes):
     - Physical procedures or manipulations of those individuals or their environment for Research purposes ("Intervention").
     - Communication or interpersonal contact with the individuals. ("Interaction").

4. **Human Subject Under DHHS Regulations**: (Check if "Yes").
   - Will the investigator gather data that is either? Specify which category(s) apply if yes:
     - The data are about behavior that occurs in a context in which an individual can reasonably expect that no observation or recording is taking place (i.e. "Private information").
     - Individuals have provided the data for specific purposes in which the individuals can reasonably expect that it will NOT be made public, such as a medical record (i.e. "Private information").
     - Can the individuals' identities be readily ascertained or associated with the information by the investigator (i.e. "Identifiable Private Information")?
     - Can the individuals' identities be readily ascertained or associated with the biospecimens (i.e. "Identifiable Biospecimen")?

If all items are checked under 1, 2, and 3 or 1, 2, and 4, the activity is Human Research under DHHS regulations.

5. **Human Research Under DHHS Regulations**: (Check if "Yes").
   - Has a department or agency head, covered by the Common Rule, retained final judgment (consistent with the ethical principles of the Belmont Report) that the activity is Human Research under DHHS regulations?

If checked, the activity is Human Research under DHHS regulations.

6. **Human Research Under FDA Regulations**: (Check if "Yes").
   - Does the activity involve any of the following? (Check all that apply)
     - In the United States: The use of a drug; in one or more persons other than use of an approved drug in the course of medical practice.
     - In the United States: The use of a device in one or more persons that evaluates the safety or effectiveness of that device.
     - Data regarding subjects or control subjects submitted to or held for inspection by FDA.
     - Data regarding the use of a device on human specimens (identified or unidentified) submitted to or held for inspection by FDA.

If "Yes", the activity is Human Research under FDA regulations.

If the activity is Human Research under DHHS regulations or under FDA regulations, it is Human Research under organizational policy.

7. **Engagement**: (Complete if the activity is Human Research. Check if "Yes")

---

¹ This document satisfies AAHRPP elements I1.A, III.1.A

---

Figure C.1 HumanResearchDetermination
RE: STUDY00020602 Update

Leitzell, Brigitt <bjh156@psu.edu>
Fri 6/24/2022 4:53 PM
To: Majumder, Sanjay <szm904@psu.edu>
Hi Sanjay,

If you and your advisor make changes to what is currently submitted and, in reviewing the information provided in section 1 of the HRP-594, self-determine that the study does not meet the definition of human subjects research then submission to the IRB is not required (please see section 1 of the HRP-594).

However, if a formal written determination is needed from the IRB, then the HRP-594 should be revised and resubmitted for review.

Best,

Brigitt

Brigitt Leitzell, M5, CIP
IRB Analyst
Human Research Protection Program
The Pennsylvania State University
Direct Telephone Line: 814-865-8397
Preferred pronouns: she/her

For general questions:
Phone: 814-865-1775; Email: irb-orp@psu.edu

The Office for Research Protections at University Park is operating remotely. Please email or call to arrange a Zoom meeting or conference call as needed.

**Figure C.2 Communication with IRB Analyst**
Appendix D
Mean Opinion Score

D.1 Datasheet

Name of the song:

<table>
<thead>
<tr>
<th>Stem Name</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>vocals</td>
<td></td>
</tr>
<tr>
<td>drums</td>
<td></td>
</tr>
<tr>
<td>bass</td>
<td></td>
</tr>
<tr>
<td>rest of the accompaniment</td>
<td></td>
</tr>
</tbody>
</table>

Figure D.1 Datasheet used to collect MOS score.
Appendix E
Python Code

E.1 Sequence of steps for running the audio source separation model

1. Install a working Python environment for the code to run in. This code has been developed using the VS code environment installed using Anaconda.

2. Install all the required libraries before running the script. List of necessary libraries are given below:
   - ffmpeg
   - Numpy
   - Librosa
   - Scipy
   - Tensorflow
   - Matplotlib
   - MUSDB18 Audio Dataset

3. Make sure all the functions, sub-functions and other miscellaneous files are in the same directory and this directory is the working directory while running Python. (List of functions and necessary files given below)
   - bigruarchitecture.py
   - training.py
   - testing.py
   - evaluation.py
E.2 Bi-GRU Architecture

The codes below in model function og Bi-Directional Gated Recurrent Unit

```python
# Bidirection GRU is implemented

import tensorflow as tf

def bi_gru():
    if tf.test.is_gpu_available() and not flags.no_gpu:
        return tf.keras.layers.Bidirectional(tf.keras.layers.CuDNNGRU(
            units = 480,
            activation = 'tanh', recurrent_activation = 'sigmoid',
            kernel_initializer = 'glorot_uniform', dropout = 0.05,
            return_sequences = True))
    else:
        return tf.keras.layers.Bidirectional(tf.keras.layers.GRU(
            units = 480,
            activation = 'tanh', recurrent_activation = 'sigmoid',
            kernel_initializer = 'glorot_uniform', dropout = 0.05,
            return_sequences = True))

def rnn_arch(input_data, stem_name):
    flatten_tensor = tf.keras.layers.TimeDistributed(tf.keras.layers.Flatten())((input_data))
    first_layer = create_bidirectional()((input_data))
    second_layer = create_bidirectional()((first_layer))
    third_layer = create_bidirectional()((second_layer))
    forth_layer = create_bidirectional()((third_layer))
    dense = tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(int(
        flatten_tensor.shape[2]),
        activation = "tanh",
        kernel_initializer = 'glorot_uniform',))((forth_layer))
```

output_layer = tf.keras.layers.TimeDistributed(Reshape(input_data.shape[2:])),

keras.layers.dense(units=480))
return output_layer

def bi_gru_model(input_data, stem_name):
    output = {}
    stems = ['bass', 'drums', 'vocals', 'others']
    for audio_stem in stems:
        stem_name = f'{audio_stem}_spectrogram'
        output[stem_name] = rnn_arch(input_data)
    return output[stem_name]

E.3 Loss Function

The codes below calculates the L1 Magnitude loss

```python
# This loss function calculates the L1 Magnitude loss function
import numpy as np
import tensorflow as tf

def loss_function(original_stems, predicted_stems):
    loss = 0
    for index, original_stem in enumerate(original_stems):
        original_audio_spectrogram = tf.signal.stft(original_stem, frame_length=2048, frame_step=512, fft_length=4096, window_fn=tf.signal.hann_window)
        original_audio_spect_abs = tf.abs(original_audio_spectrogram)
        predicted_stem = predicted_stems[index]
        predicted_audio_spectrogram = tf.signal.stft(predicted_stem, frame_length=2048, frame_step=512, fft_length=4096, window_fn=tf.signal.hann_window)
        predicted_audio_spect_abs = tf.abs(predicted_audio_spectrogram)
        loss += tf.reduce_mean(tf.abs(original_audio_spect_abs - predicted_audio_spect_abs))
    weighted_loss = loss / 4
```

134
E.4 Training

The codes below train the network

```python
# This loss function calculates the L1 Magnitude loss function
import numpy as np
import tensorflow as tf
def loss_function(original_stems, predicted_stems):
    loss = 0
    for index, original_stem in enumerate(original_stems):
        original_audio_spectrogram = tf.signal.stft(original_stem,
                                                   frame_length=2048, frame_step=512,
                                                   fft_length=4096, window_fn=tf.signal.hann_window)
        original_audio_spect_abs = tf.abs(actual_audio_spectrogram)
        predicted_stem = predicted_stems[index]
        predicted_audio_spectrogram = tf.signal.stft(predicted_stem,
                                                   frame_length=2048, frame_step=512,
                                                   fft_length=4096, window_fn=tf.signal.hann_window)
        predicted_audio_spect_abs = tf.abs(predicted_audio_spectrogram)
        loss += tf.reduce_mean(tf.abs(original_audio_spect_abs -
predicted_audio_spect_abs))
    weighted_loss = loss / 4
    return weighted_loss
```

E.5 Testing

The codes below used for testing find the sampling frequency and it’s effect on spectrogram

```bash
#!/usr/bin/env python
# coding: utf8

""
This script is used to show the sampling frequency problem of "librosa" library and its effect on spectrogram in the Conclusion section
""
```
import numpy as np
import librosa
import librosa.display
from scipy.io import wavfile
import matplotlib.pyplot as plt

audio_data_librosa, sampling_frequency_librosa = librosa.load("G:/thesiswork_v2/musdb_dataset/train/A Classic Education - NightOwl/vocals.wav")
sampling_frequency_wav, _ = wavfile.read("G:/thesiswork_v2/musdb_dataset/train/A Classic Education - NightOwl/bass.wav",'rb')

print(f"Sampling Frequency using 'librosa': {sampling_frequency_librosa} Hz")
print(f"Sampling Frequency using 'scipy.io.wavfile': {sampling_frequency_wav} Hz")

time_frequency_domain_data = librosa.stft(audio_data_librosa)
audio_data_to_db = librosa.amplitude_to_db(np.abs(time_frequency_domain_data), ref=np.max)

fig, ax = plt.subplots()
img = librosa.display.specshow(audio_data_to_db, x_axis='time', y_axis='hz', sr=sampling_frequency_librosa, ax=ax)
ax.set(title='Spectrogram with Wrong Sampling Frequency')
ax.set_xlabel('time (s)', fontweight='bold')
ax.set_ylabel('Frequency (Hz)', fontweight='bold')
fig.colorbar(img, ax=ax, format="%+2.1f dB")
plt.show()

E.6 Evaluation

The codes below show the problem of librosa to find the sampling frequency and it’s effect on spectrogram

#!/usr/bin/env python
# coding: utf8

""
This script is used to show the sampling frequency problem of "librosa" library and its effect on spectrogram in the Conclusion section
"""
import numpy as np
import librosa
import librosa.display
from scipy.io import wavfile
import matplotlib.pyplot as plt

audio_data_librosa, sampling_frequency_librosa = librosa.load("G:/thesiswork_v2/musdb_dataset/train/A Classic Education - NightOwl/vocals.wav")
sampling_frequency_wav, _ = wavfile.read("G:/thesiswork_v2/musdb_dataset/train/A Classic Education - NightOwl/bass.wav", 'rb')

print(f"Sampling Frequency using 'librosa': {sampling_frequency_librosa} Hz")
print(f"Sampling Frequency using 'scipy.io.wavfile': {sampling_frequency_wav} Hz")

time_frequency_domain_data = librosa.stft(audio_data_librosa)
audio_data_to_db = librosa.amplitude_to_db(np.abs(time_frequency_domain_data), ref=np.max)

fig, ax = plt.subplots()
img = librosa.display.specshow(audio_data_to_db, x_axis='time', y_axis='hz', sr=sampling_frequency_librosa, ax=ax)
ax.set(title='Spectrogram with Wrong Sampling Frequency',
       xlabel='time (s)', fontweight='bold',
       ylabel='Frequency (Hz)', fontweight='bold')
fig.colorbar(img, ax=ax, format="%.2f dB")
plt.show()

E.7 Plotting waveform and Spectrogram

The script below has the code to calculate and plot waveform and spectrogram

#!/usr/bin/env python
# coding: utf8
""
This script is used for plotting the waveform and spectrogram in the result section
"""
import librosa
import librosa.display
import numpy as np
import matplotlib.pyplot as plt

audio_data, sampling_rate = librosa.load("G:/thesiswork_v2/musdb_dataset_syn/test/AM Contra - Heart Peripheral/drums.wav", sr=44100)
tf_audio_data = librosa.stft(audio_data)
tf_db = librosa.amplitude_to_db(np.abs(tf_audio_data), ref=np.max)

fig, ax = plt.subplots()
librosa.display.waveplot(audio_data, sr=44100, color='g', alpha=0.5, ax=ax, label='Original')
ax.set(title='Waveform')
ax.set_xlabel('time (s)', fontweight='bold')
ax.set_ylabel('Amplitude (dB)', fontweight='bold')
plt.show()

fig, ax = plt.subplots()
img = librosa.display.specshow(tf_db, x_axis='time', y_axis='hz', sr=44100, ax=ax)
ax.set(title='Spectrogram')
ax.set_xlabel('time (s)', fontweight='bold')
ax.set_ylabel('Frequency (Hz)', fontweight='bold')
fig.colorbar(img, ax=ax, format='%.2f dB')
plt.show()

**E.8 RMS Energy Calculation**

The script below has the code to calculate and plot the Root Mean Square (RMS) Energy of an audio.

```python
import librosa
import matplotlib.pyplot as plt

audio_data_original, sr = librosa.load("G:/thesiswork_v2/musdb_dataset_syn/test/AM Contra - Heart Peripheral/mixture.wav", sr=44100)
```
rms_original = librosa.feature.rms(y=audio_data_original)
times_original = librosa.times_like(rms_original)

## RMS Energy of the separated audio mixture
audio_data_vocals, _ = librosa.load("G:/thesiswork_v2/desourcing/metrices_done/separated audio/AM Contra - Heart Peripheral/vocals.wav", sr=44100)
audio_data_drums, _ = librosa.load("G:/thesiswork_v2/desourcing/metrices_done/separated audio/AM Contra - Heart Peripheral/drum.wav", sr=44100)
audio_data_bass, _ = librosa.load("G:/thesiswork_v2/desourcing/metrices_done/separated audio/AM Contra - Heart Peripheral/bass.wav", sr=44100)
audio_data_rest, _ = librosa.load("G:/thesiswork_v2/desourcing/metrices_done/separated audio/AM Contra - Heart Peripheral/other.wav", sr=44100)

audio_data_separated = audio_data_vocals + audio_data_drums + audio_data_bass + audio_data_rest
rms_separated = librosa.feature.rms(y=audio_data_separated)
times_separated = librosa.times_like(rms_separated)

# plotting the RMS Energy of the original audio mixture
# y-axis is in log form
fig, ax = plt.subplots(nrows=2, sharex=True)
ax[0].semilogy(times_original, rms_original[0], label='RMS Energy of the Original Audio Mixture')
ax[0].set(title='RMS Energy of the Original and Separated Audio Mixture')
ax[0].set_xlabel('time (s)', fontweight='bold')
ax[0].set_ylabel('RMS Energy (dB)', fontweight='bold')
# ax[0].set(xticks=[])  # ax[0].set(yticks=[])  # ax[0].legend()  # ax[0].label_outer()

# plotting the RMS Energy of the separated audio mixture
# y-axis is in log form
E.9 Problem with librosa library and its effect

The codes below show the problem of librosa to find the sampling frequency and it’s effect on spectrogram

```python
#!/usr/bin/env python
# coding: utf8

""
This script is used to show the sampling frequency problem of "librosa" library and its effect on spectrogram in the Conclusion section
""
import numpy as np
import librosa
import librosa.display
from scipy.io import wavfile
import matplotlib.pyplot as plt

audio_data_librosa, sampling_frequency_librosa = librosa.load("G:/thesiswork_v2/musdb_dataset/train/A Classic Education - NightOwl/vocals.wav")
sampling_frequency_wav, _ = wavfile.read("G:/thesiswork_v2/musdb_dataset/train/A Classic Education - NightOwl/bass.wav", 'rb')

print(f"Sampling Frequency using 'librosa': {sampling_frequency_librosa} Hz")
print(f"Sampling Frequency using 'scipy.io.wavfile': {sampling_frequency_wav} Hz")
time_frequency_domain_data = librosa.stft(audio_data_librosa)
```

```python
ax[1].semilogy(times_separated, rms_separated[0], "r", label='RMS Energy of the Separated Audio Mixture')
ax[1].set_xlabel('time (s)', fontweight='bold')
ax[1].set_ylabel('RMS Energy (dB)', fontweight='bold')
#ax[1].set(xticks=[]) #ax[1].set(yticks=[])
ax[1].legend()
ax[1].label_outer()
plt.show()
```
audio_data_to_db = librosa.amplitude_to_db(np.abs(time_frequency_domain_data), ref=np.max)

fig, ax = plt.subplots()
img = librosa.display.specshow(audio_data_to_db, x_axis='time', y_axis='hz', sr=sampling_frequency_librosa, ax=ax)
ax.set(title='Spectrogram with Wrong Sampling Frequency')
ax.set_xlabel('time (s)', fontweight='bold')
ax.set_ylabel('Frequency (Hz)', fontweight='bold')
fig.colorbar(img, ax=ax, format='%+2.2f dB')
plt.yscale('log')
plt.show()
Bibliography


URL https://www.acousterr.com/blog/how-to-identify-different-genres-of-music


URL https://www.malicksarr.com/why-is-deep-learning-called-deep-complete-deep-learning-definition/


URL https://medium.com/@minat_silvester/what-are-tensors-how-are-they-used-in-machine-learning-7550198c44ed

URL https://paperswithcode.com/sota/music-source-separation-on-musdb18


URL https://doi.org/10.5281/zenodo.3532214