MUSIC SEPARATOR FOR WINDOWS APPLICATION

A MINI PROJECT REPORT

Submitted by

AJITHKUMAR P (20EUIT501)

ARAVIND S (20EUIT502)

ARSAAD S (20EUIT503)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

SRI KRISHNA COLLEGE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institution)

(Approved by AICTE and Affiliated to Anna University, Chennai)

ACCREDITED BY NAAC WITH “A” GRADE

MAY 2022

SRI KRISHNA COLLEGE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institution)

(Approved by AICTE and Affiliated to Anna University, Chennai)

ACCREDITED BY NAAC WITH “A” GRADE

BONAFIDE CERTIFICATE

Certified that this project report **“Music Separator for Windows Application”** is the bonafide work of **“Ajithkumar P (20EUIT501), Aravind S (20EUIT502), Arsaad S (20EUIT503)”** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
[Dr. G. Edwin Prem Kumar]	[Dr. N. SUSILA]
PROFESSOR	**PROFESSOR & HEAD**
Information Technology	Information Technology,
Sri Krishna College of Engg & Tech	Sri Krishna College of Engg & Tech
Coimbatore - 641008.	Coimbatore - 641008.

This project report is submitted for the autonomous project viva-voice examination held on ………………………

INTERNAL EXAMINER **EXTERNAL EXAMINER**

TABLE OF CONTENTS

ACKNOWLEDGEMENT

We express our deep sense of gratitude and sincere thanks to our beloved Principal**, Dr. J. Janet** , Sri Krishna College of Engineering and Technology, for permitting us to do project, which was beginning step for our future project.

We also intended our profound thanks to **Dr. N. Susila**, Head Of The Department, Department of Information Technology, Sri Krishna College of Engineering and Technology, for her motivation and encouragement throughout the project

We are sure that the success of this project could have been impossible without the project guide , **Dr. G. Edwin Prem Kumar**, Professor, Department of Information Technology. Sri Krishna College of Engineering and Technology. We thank for your continuous support and guidance which was very much helpful in the completion of the project.

ABSTRACT

Last years, Music Source Separation (MSS) has been one of the most active fields within signal processing. Music Source separation for music is the task of isolating contributions, or stems, from different instruments recorded individually and arranged together to form a song. Such components include voice, bass, drums and any other accompaniments. Music Source Separation is the process of separating a mixture into isolated sounds from individual sources (e.g. just the lead vocals). The design of such algorithms seeks to recreate the human ability to identify individual sound sources. In the music field, efforts are being made to isolate the main instruments from a single audio file with a mixture of stereo audio. Contrarily to many audio synthesis tasks where the best performances are achieved by models that directly generate the waveform, the state-of-the-art in source separation for music is to compute masks on the magnitude spectrum. In the music field the separated audio files are called stems. A stem is a discrete or grouped collection of audio sources mixed together, usually by one person, to be dealt with downstream as one unit. A single stem may be delivered in mono, stereo, or in multiple tracks for surround sound. The goal of these algorithms is to extract multiple audio files with specific instruments, such as bass, voice or drums. These stems allows you, as a creator, to control each of the particular mixes for your production. Stems tend to break down into four tracks, commonly including the melody, instruments, bass, and drums. This project focuses on analyzing the existing systems based on neural networks and their performance. In addition, it goes deeply into the Open-Unmix algorithm structure and tries to improve its results. However, it suffers from some bleeding, especially between the vocals and other source. Our mainly goal, a normal people can separate the music file into stems (Vocal, Bass, Drums and Others) in the easier way.

LIST OF FIGURES

List of Tables

LIST OF ABBREVIATIONS

SSS	Sound Source Separation
MSS	Music Source Separation
NN	Neural Network
ML	Machine Learning
DL	Deep Learning
PCA	Principal Component Analysis
LSTM	Long-Short Time Memory
CNN	Convolutional Neural Networks
RNN	Recurrent Neural Networks
STFT	Short-Time Fourier Transform
SDR	Source to Distortion Ratio
SIR	Source to Interference Ratio
SAR	Source to Artefact Ratio
ReLU	Rectified Linear Unit
GUI	Graphical User Interface

CHAPTER 1
INTRODUCTION

1.1. STATEMENT OF PURPOSE

This project consists on the usage of Spleeter - AI source separation deep neural networks to perform music source separation (decomposing music into its constitutive components) and the usage of C# .Net to develop GUI(Graphical User Interface).

The main goals of the project are:

1. Being able to extract acapella from a mixed audio.
2. Being able to extract the stems from a mixed audio.
3. Make it easy for Windows users to download and run without needing to use the command line tools to do so.
4. Being able to set solid next steps to improve the system.

1.2. REQUIREMENTS AND SPECIFICATIONS

The project should perform well on the mixture separation from an audio source. It has to be able to extract stems from a audio file. The software will be complemented with a detailed description of the network architecture used and the main reasons of the choice.

System performance will be evaluated using concrete metrics and will be compared with state-of-art of Music Source Separation, described in sections below.

1.3. METHODS AND PROCEDURES

The final implementation of the project is based on Open-Unmix Music Separation Software . The training is performed using MUSDB18 Music Source Separation database .

The main study results are centered on the voice track. Focusing the problem to a single track extraction has allowed us to do a more consistent study and comparison of the results, especially when tuning the network hyperparameters. The main reasons of the choice and the overall base structure will be defined in next sections.

1.4. WORK PLAN

The final work plan has not changed much from the initial planning. The work has been organized using different work blocks (packages) and following the structure below. As it can be seen, the work organization follows a natural path with the focus on getting knowledge. Software Development and Result Analysis were conceived in an iterative way, so that new results encourage changes to the software parameters or structure.

BREAKDOWN STRUCTURE

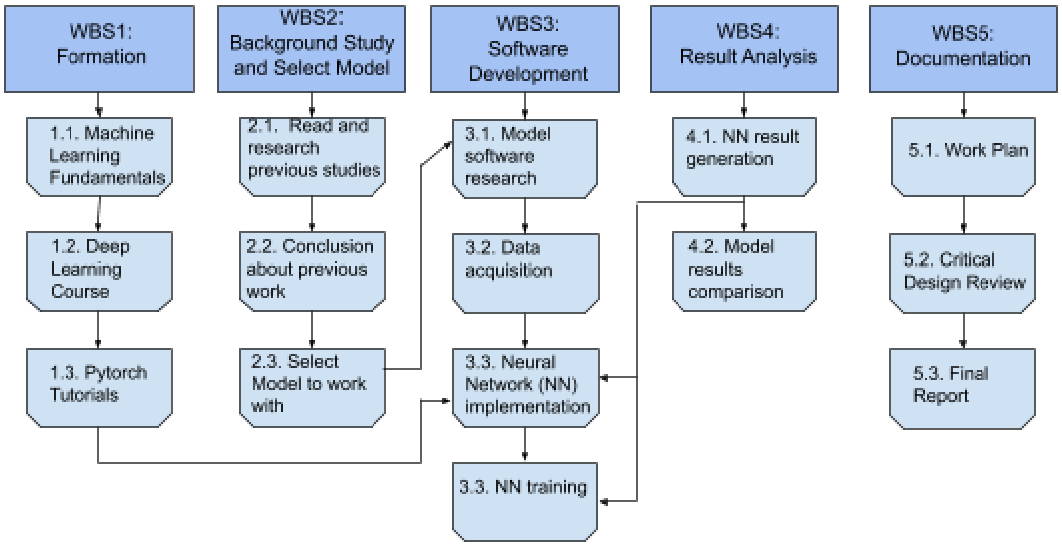

WBS1: Formation
WBS2: Background Study and Select Model
WBS3: Software Development
WBS4: Result Analysis
WBS5: Documentation
1.1. Machine Learning Fundamentals
1.2. Deep Learning Course
1.3. Pytorch Tutorials
2.1. Read and research previous studies
2.2. Conclusion about previous work
2.3. Select Model to work with
3.1. Model software research
3.2. Data acquisition
3.3. Neural Network (NN) implementation
3.3. NN training
4.1. NN result generation
4.2. Model results comparison
5.1. Work Plan
5.2. Critical Design Review
5.3. Final Report

Fig 1.1. Breakdown work structure of the project

The monitoring of this structure has been done by setting milestones. Every milestone had a concrete objective and a fixed date. This is shown in the fig below.

1.5. DEVIATIONS FROM THE INITIAL PLAN

The initial Work Plan has been affected by some factors related to the chosen source separation model and the installation of a virtual environment. First weeks, all the efforts were put on getting first training results and optimize system training: trying different parameters (like number of parallel workers) and using GPU.

Other problems had appeared in the following weeks. When we needed to adapt the environment to achieve the specifications of the improvement proposal functions, we had several compatibility problems between libraries and Spleeter environment. Furthermore, some functions were not available due to problems with the server specifications.

CHAPTER 2
SYSTEM STUDY AND ANALYSIS

2.1. MUSIC SOURCE SEPARATION (MSS)

Last years, Sound Source Separation (SSS) algorithms have been one of the most active fields in signal processing. These algorithms try to recreate the human capacity to differentiate individual sound sources using spatial and frequency component (timbre) information.

Music Source Separation (MSS) is a concrete field into SSS. It tries to split a music audio clip into its constituent contributions (stems), such as the vocals, bass and drums. This separation can be used to re-mix, suppress or up-mix sources from a final recording mix.

Current algorithms are far from providing high quality individual audio stems. Usually, there may be a lot of musical instruments in a mono or stereo recording (only 1 or 2 channel recording), and the sources have been modified by signal filtering or reverberation additionin the mixing process, among others. All of these modifications make MSS a very complex problem. The nature of the problem can be seen in the time-frequency domain.

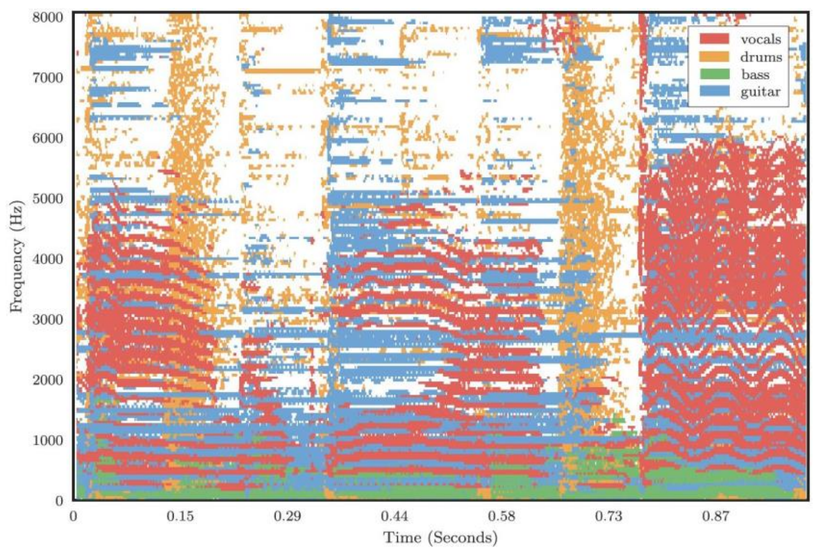


Fig 2.1. Representation of a music mixture in the time-frequency domain.

As it is shown above, the dominant musical source in each time-frequency bin is displayed with a different color. It can be seen that some instruments have a very wide frequency range. Besides, we can see some time-frequency masking problem, which will difficult source separation, especially in tracks like drums or guitar. In the following sections we will define the main strategies for MSS.

2.2. DEEP LEARNING AND MSS

Various algorithms have been implemented to achieve a high quality stem segmentation, such as Principal Component Analysis (PCA), defined in or Non-Negative Matrix Factorization, explained in . These algorithms have been lightly overshadowed by the success of Machine Learning (ML) and deep learning (DL) techniques.

Deep learning algorithms are able to model non-linearities and provide faster implementations than previous algorithms. These are usually formulated as a supervised learning problem using concrete targets (such as vocals or drums) and different cost functions. Most of the algorithms use Recurrent Neural Network structures such as LSTM modules to explore time dependencies. In the next section we will put the focus on those modules and other interesting concepts commonly used in MSS.

SiSEC MUS 18 is a SSS evaluation challenge that introduce state-of-the-art algorithms in MSS and is used as starting point to analyze new algorithms.

2.2.1. MACHINE LEARNING AND DEEP LEARNING FUNDAMENTALS

In this section we will explain the key concepts to understand how MSS Deep Learning algorithms work. With this in mind, we will start with a short general description of how a neural network works and then we will explain three of the most common used modules in MSS. Finally, we will briefly explain the fundamentals of training diagnosis.

2.2.1.1. GENERAL DESCRIPTION

Deep Learning is a subfield of machine learning concerned with algorithms inspired by the structure and function of the brain called artificial neural networks. Furthermore, a Neural Network is combination of several layers of neurons or perceptron’s (the basic elementary unit). NNs are made of input and output layers/dimensions, and in most cases, they also have a hidden layer consisting of units that transform the input into something that the outputlayer.

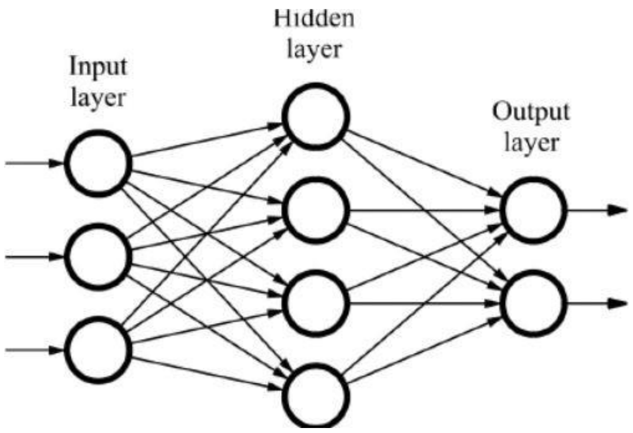


Fig 2.2. Neural Network basic structure.

In a single neuron unit the input data is poundered and transformed in a linear or non-linear way to generate a single output.

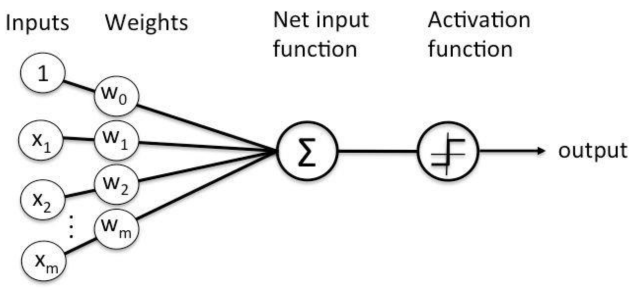


Fig 2.3. Rosenblatt's Perceptron Unit.

The weights are transformed in every iteration until the network outputs are close to the desired outputs. This weight actualization is called back-propagation and it uses the difference between the actual output and the ideal output of every layer (minimization of lossfunction), from the output layer to the input layer. Some parameters are used in order to achieve a good network performance, such as learning rate, that controls how quickly is the weights adaption, or regularization, which control how much we want to penalize the flexibility of our model by shrinking the coefficients towards zero.

2.2.1.2. CONVOLUTIONAL LAYERS

Convolutional Neural Networks (CNN) were originally designed for image processing. A convolutional layer was designed to reduce the images into a form that is easier to process (dimensionality reduction) without losing features which are essential for a good prediction.

CNN are able to successfully capture the spatial and temporal dependencies through the application of relevant filters (Kernel).

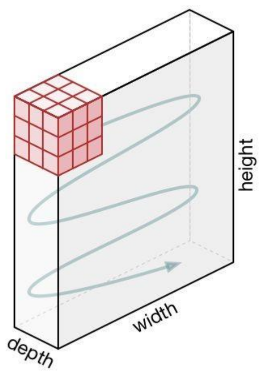


Fig 2.4. Kernel filter performance example.

In the image, we can see that the Kernel (red block) moves through the entire image with a certain Stride Value (positions to be moved after the first position in every iteration). The output in every position is the sum of the element-wise products between the filter and the multidimensional input. During the network training the filter coefficients change to optimize results.

The capacity for dimensionality reduction and capturing spatial and temporal dependence makes CNN a good choice in audio processing. Depending on the input dimensionality, 1D or 2D layers can be used (e.g. raw audio signal or spectrogram).

2.2.1.3. LSTM MODULES

Long-Short Time Memory (LSTM) modules are special kind of Recurrent Neural Networks (RNN), networks with loops in them, allowing information to persist. LSTMs are capable of learning long-term dependencies between input samples. Furthermore, they are great remembering information for long time periods.

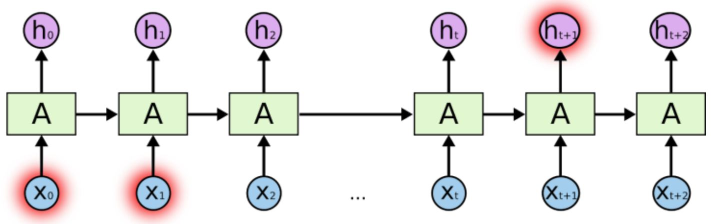


Fig 2.5. Unrolled RNN with time dependency data.

LSTMs have a chain like structure with a repeating module. In one of the most common implementations, each module is composed by a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

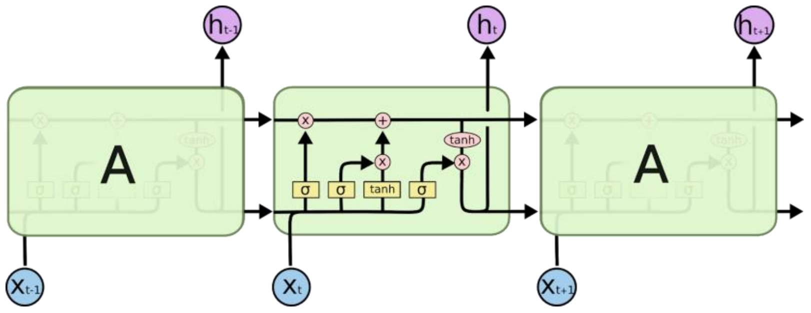


Fig 2.6. LSTM implementation.

2.2.1.4 BATCH NORMALIZATION

Batch normalization is used to increase the stability of a neural network. The process consists on normalizing the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation. It makes sure that there’s no activation that’s gone really high or really low and reduces overfitting by adjusting the parameterization of a model in order to make the loss surface smoother.

2.2.1.5 ITERATION/LOSS TRAINING ANALYSIS

To analyze how good a model performance is, we use the relation between loss function and number of iterations (epochs). A lower loss implies a better data modeling. To understand how good the model generalizing is for new data, the loss is calculated on training and validation (test) and its interpretation is how well the model is doing for these two sets.

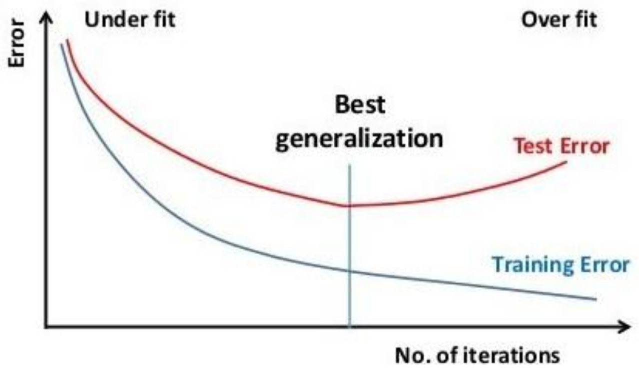


Fig 2.7. Error/Iteration network training graphic example.

For network diagnosis we need to define two terms:

BIAS

Bias measures whether the average predicted values are far from the actual values. It measures whether the model does or does not capture the complexity of data. It is referred to the training sample.

VARIANCE

Variance measures the difference between the model performance in train and test datasets. High-variance problem tells us that the model is not generalizating well for new data.

When training a model we can get stuck as the model "memorizes" the training examples and becomes kind of ineffective for the test set (over-fitting). Over-fitting also occurs in caseswhere you have a very complex model or the dataset is not large enough.

On the other hand, under-fitting happens when a model is not able to accurately capture relationships between dataset features and a target variable.

Some measures can be applied to reduce both problems. To reduce over-fitting we can try to get more training examples, reduce the number of features or increase regularization. To solve under-fitting we can try to get additional features or increase regularization.

2.2.2. MSS ALGORITHMS

MSS algorithms are divided in two families. Depending on the input, we can distinguish two types of algorithms: waveform and pre-processed waveform (e.g.: spectrogram) algorithms. We will explain their main differences and most common structures below.

2.2.2.1. WAVEFORM ALGORITHMS

Waveform domain algorithms are designed as end-to-end systems. Working in this domain allows modeling phase information, avoiding fixed spectral transformations and low latency calculations.

Most of these algorithms use multiple 1D convolution layers and encoding-decoding structure using down sampling and up sampling. This kind of structure is called U-Net structure. It was introduced in biomedical imaging to improve precision and localization of microscopic images. One example could be the Wave-U-Net neural network.

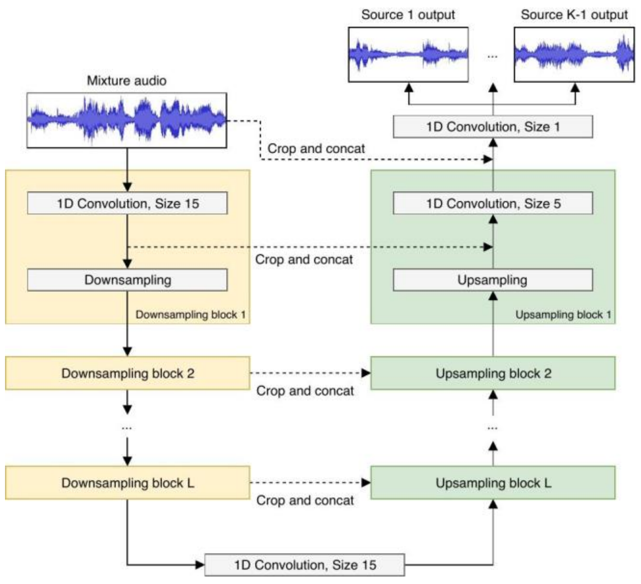


Fig 2.8. Proposed Wave-U-Net structure with K sources and L layers.

We can see that the initial audio mixture is used in the final step through a neural network skipped connection. This provides the magnitude and phase information about the original audio, which is used to obtain final source outputs by applying the mask weights obtained from the network. Most famous algorithms are Demucs, Wave-U-Net and Conv-Tasnet. Despite not being the most extended algorithms, last results show the potential of end-to-end systems in MSS and place these models in the forefront of the MSS list on SISEC18.

2.2.2.2. PRE-PROCESSED WAVEFORM ALGORITHMS

Most common pre-processed waveform algorithms operate on the spectrograms generated by the Short-Time Fourier Transform (STFT). They produce a mask on the magnitude spectrums for each frame and each source. The output audio is generated by running an inverse STFT on the masked spectrograms. However, the STFT output depends on many parameters, such as the size and overlap of audio frames, and it can affect the time and frequency resolution.

As in waveform-domain, these algorithms usually explore U-Net structure networks. But in this case using 2D convolutional layers (e.g. using three dimensional tensors with number of channels, time-steps and frequency bins). One example could be Spleeter by using a U-Net Deep Convolutional Layer from a baseline.

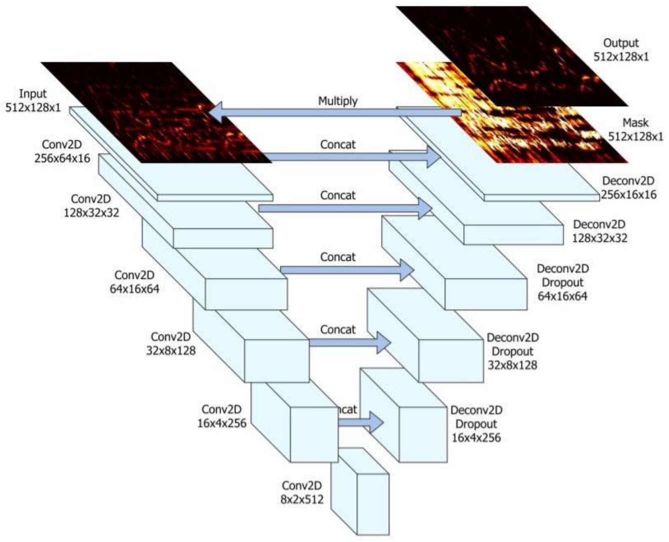


Fig 2.9. Deep U-Net convolutional structure

Most famous algorithms are Spleeter and Open-Unmix. Open-Unmix will be ourproposed approach for the project.

2.2.3. EVALUATION METRICS

Evaluating the results of a music source separation model is a difficult task. However, several objective metrics were developed to facilitate evaluating Blind Audio Source Separation (BASS) algorithms. To this end, the estimated sources $\widehat{s_j}$ (with j = 1…J) are decomposed as:

$$\widehat{s_j} = s_{target} + e_{interf} + e_{noise} + e_{artif}$$

Where s_{target} is a version of the original source, e_{interf} is the interference coming from not desired sources, e_{noise} is the sensor noise and e_{artif} refers to the musical noise self-generated for the separation algorithm.

This decomposition has established the objective metrics and the values we want to maximize:

- SDR (Source to Distortion Ratio), defined as:

$$SDR := 10 * log_{10} \frac{\|s_{target}\|^2}{\| + e_{interf} + e_{noise} + e_{artif}\|^2}$$

- SIR (Source to Interference Ratio), defined as:

$$SIR := 10 * log_{10} \frac{\|s_{target}\|^2}{\|e_{interf}\|^2}$$

- SAR (Source to Artefact Ratio), defined as:

$$SAR := 10 * log_{10} \frac{\|s_{target} + e_{interf} + e_{noise}\|^2}{\|e_{artif}\|^2}$$

2.3. PROGRAMMING ENVIRONMENT

The programing and development of the project has been made using Python and PyTorch.

2.3.1 PYTORCH

PyTorch is an open source machine learning framework that accelerates the path from prototyping to production deployment. The PyTorch Torch package contains data structures for multi-dimensional tensors (data container) and mathematical operations over these are defined. Its principal modules are torch.nn or torch.optim. The nn modules in PyTorch provide us a higher level API to build and train deep network.

CHAPTER 3

SYSTEM METHODOLOGY

The project development has been centered into the optimization of the Spleeter software performance. The system has been trained using the MUSDB18 dataset. The main reasons of the choice will be explained in detail in the next sections.

3.1. DATASET

The SigSep MUSDB18 data set consists of a total of 150 full-track songs of different styles and it includes both the stereo mixtures and the original sources, divided into a training subset and a test subset. All the tracks are stereophonic and encoded at 44,1kHz.

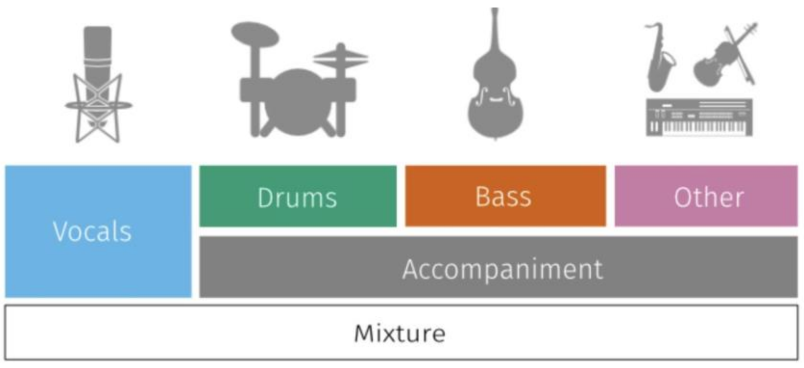


Fig 3.1. MUSDB18 dataset stem distribution.

Its purpose is to serve as a reference database for the design and the evaluation of source separation algorithms. The objective of such signal processing methods is to estimate one or more sources from a set of mixtures, e.g. for karaoke applications. It has been used as the official dataset in the professionally-produced music recordings task for SiSEC 2018.

The database is presented in compressed and uncompressed (high quality) format. The compressed files have a bandwidth limited to 16 kHz. But nevertheless, it seems not affecting to the evaluation performance.

MUSDB18 is actually the most complete database for MSS. Furthermore, all the systems from state-of-art have been using this dataset in order to compare and share their results.

3.2. BASELINE SYSTEM

The proposal approach is based on Open-Unmix Architecture. We will describe its main advantages and the main reasons of this choice below.

3.2.1. OVERALL STRUCTURE

Open-Unmix is an open source MSS software that works with audio spectrograms. Its main advantages are simplicity and fast training.

Its architecture is very simple. It is based on LSTM modules. We will describe its main modules using the Open-Unmix paper description and some considerations.

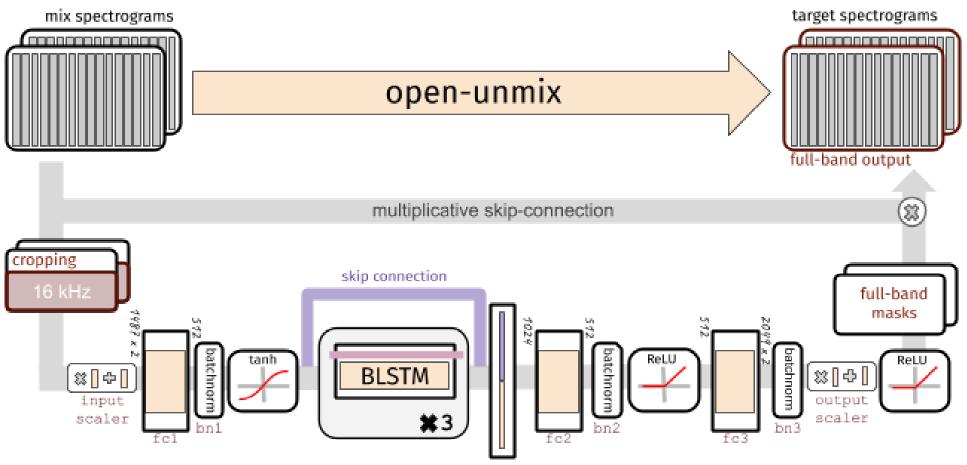


Fig 3.2. Open-Unmix model structure.

INPUT STAGE

Open-Unmix operates in the time-frequency domain to perform its prediction. The input of the model is either:

- Time domain signal tensor (later transformed with STFT)
- Magnitude spectrogram tensor (e.g. when pre-computed and loaded fromdisk)

First of all, the audio input is chunked into 6 seconds excerpts. It guarantees the audio correlation between samples without compromising the computational cost. In one iteration (epoch), 64 samples in random position are selected from each track to ensure a balanced track sampling.

The input spectrogram is cropped at 16kHz (in relation with the compressed MUSDB 18 bandwidth) and standardized using the global mean and standard deviation for every frequency bin across all frames. Furthermore, batch normalization is applied in multiple stages of the model to make the training more robust against gain variation.

DIMENSIONALITY REDUCTION

The LSTM is not operating on the original input spectrogram resolution. In the first step, the network learns to compress the frequency and channel axis of the model to reduce redundancy and make the model converge faster. Fully connected time-distributed layers are used for dimensionality reduction and augmentation, thus encoding/decoding the input and output.

BIDIRECTIONAL LSTM

The core of open-unmix is a three layer bidirectional LSTM network. The fig below shows the structure of a single bidirectional LSTM layer.

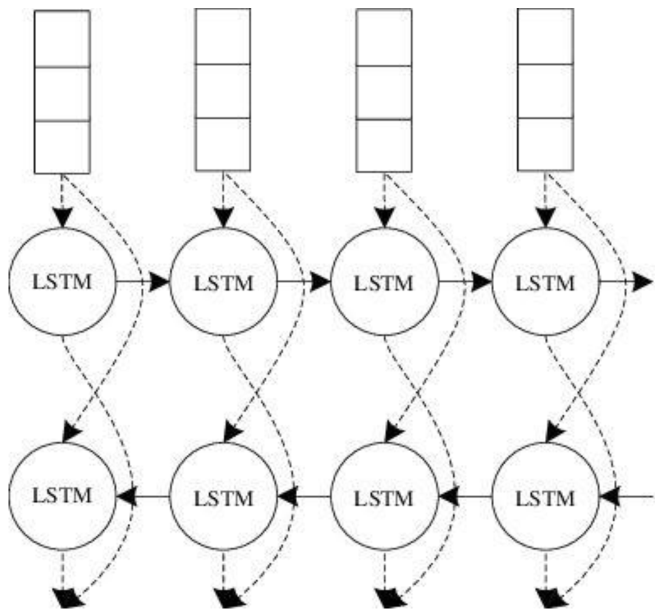


Fig 3.3. Bidirectional LSTM scheme.

As it is shown above, the information of every input sample is used in LSTM cells to work with the previous or next LSTM cell. Since the model takes information from past and future simultaneously, the model cannot be used in an online/real-time manner.

SKIPPED CONNECTION

Skipped Connections are used in two ways:

- The output to recurrent layers are augmented with their input, and this proved to help convergence.

- The output spectrogram is computed as an element-wise multiplication of the input. This means that the system has to learn how much each TF bin does belong to the target source.

OUTPUT STAGE

After applying the LSTM, the signal is decoded back to its original input dimensionality. In the last steps the output is multiplied with the input magnitude spectrogram, so that the models are asked to learn a mask. In this final step the output signal is synthesized by inverse STFT.

For inference, the signal is post-processed with an implementation of a multichannel Wiener filter that is a very popular way of filtering multichannel audio for several applications. This filtering method assumes you have some way of estimating power or magnitude spectrograms for all the audio sources composing a mixture. Norbert allows us to create a residual model when working with individual sources.

As described in the technical details on Open-Unmix SigSep documentation, different activation functions are used.

- Rectified linear units (ReLU) allow intermediate layers to comprise nonnegative activations.

- Tanh are necessary for good training of LSTM model, notably because they avoid exploding input and output.

- Sigmoid activation is chosen to mimic the way legacy systems take the output as afiltering of the input.

3.2.2. SPECIFICATIES

In this section we will describe the main advantages of the Open-Unmix software and the main reasons why we have chosen it.

The design has been oriented to reach two main objectives:

- To have state-of-art performance
- To be easily understandable (for research purposes) Furthermore, other specificities have been accomplished. The code is:
- Simple to extend: The pre/post-processing, data-loading, training and models part of the code is isolated and easy to replace/update.

- Not a package: The software is composed of largely independent and self-containing parts, keeping it easy to use and easy to change.

- Hackable (MNIST like): Open-unmix mimics the famous MNIST example, including the ability to instantly start training on a dataset that is automatically downloaded.

- Reproducible: Releasing Open-Unmix attempts to provide a reliable implementation sticking to established programming practice.

The determinant factors for choosing this architecture were its training speed, which allowed us to train multiple times and have results within two or three days, and its simplicity, which gave us a concrete vision of all the network hyperparameters and its functions.

3.3. METHODOLOGY

The methodology followed during the project has been based on an alternation between result analysis and improvement proposal and development. After the generation of the first training results the work has been focused in two main blocks: network parameter and hyperparameter tuning and network improvements proposal. All the changes were tested in the same target to really evaluate its impact.

The network training has been done using the UPC Calcula service. This allowed us to streamline the training by using GPU.

We have decided to center the study of the systems in terms of Training/Test loss instead of the metrics explained in 2.2.3. This decision is based on the computational cost and time cost of the metrics calculation and it is also based on the directly proportional relation between SDR and best epoch test loss. This decision has allowed us to increase the numberof experiments and has helped especially in parameter tuning.

The training methodology and the Open-Unmix data processing show some result differences between two identical parameter trainings. For this reason, we have trained each model three times and we have selected the best performance.

3.3.1. PARAMETER TUNING

As it has been seen before, Open-Unmix architecture has some parameters that are interesting to study.

After analyzing the first results, a network diagnostic has shown possible parameter changes. We have studied the algorithm performance under variations of the batch size, the dimensionality reduction in the fully-connected layer (input of the LSTM) and the weight decay for regularization.

We have centered the study on the most common parameters or the parameters that we thought to be most important for the training performance.

3.3.2. IMPROVEMENT PROPOSAL

The improvement proposal and implementation have been conditioned by the Open-Unmix main specificities. We have tried to look for modifications that would not affect directly on thecomputational cost or the understandability of the architecture. For this reason, the work has been centered on searching pre-processing tools to improve the overall system performance.

In pre-processing we have worked to emphasize the selected target in the input of the network. We have tried fixed filters and learnable filters to explore how equalization can affect to a concrete target extraction. We also tested to reduce the frequency threshold of the maximum bandwidth processed by the LSTM (input stage frequency cropping).

CHAPTER 4
SYSTEM IMPLEMENTATION

This section will include the data analysis and findings. We will start with an analysis of the first obtained train results. Then, these results will be used for parameter tuning orientation. Finally, we will analyze the post-processing improvement proposal results.

The default most important training parameters and hyper-parameters are shown in the tablebelow.

Argument	**Description**	**Default**
--batch-size <int>	Batch size has influence on memory usage and performance ofthe LSTM layer	16
--seq-dur <int>	Sequence duration in seconds of chunks taken from the dataset.A value of <=0.0results in full/variable length	6.0
--hidden-size <int>	Hidden size parameter of dense bottleneck layers	512
--lr <float>	learning rate	0.001
--weight-decay <float>	weight decay for regularization	0.0000 1
--bandwidth <int>	maximum bandwidth in Hertz processed by the LSTM. Input andOutput is always full bandwidth!	16000
--nfft <int>	size of fft	4096

Table 4.1. Open-Unmix most important training parameters

After the first training attempts, the first results were generated. We defined next steps after analyzing its performance. Concretely, the iteration/loss graphic.

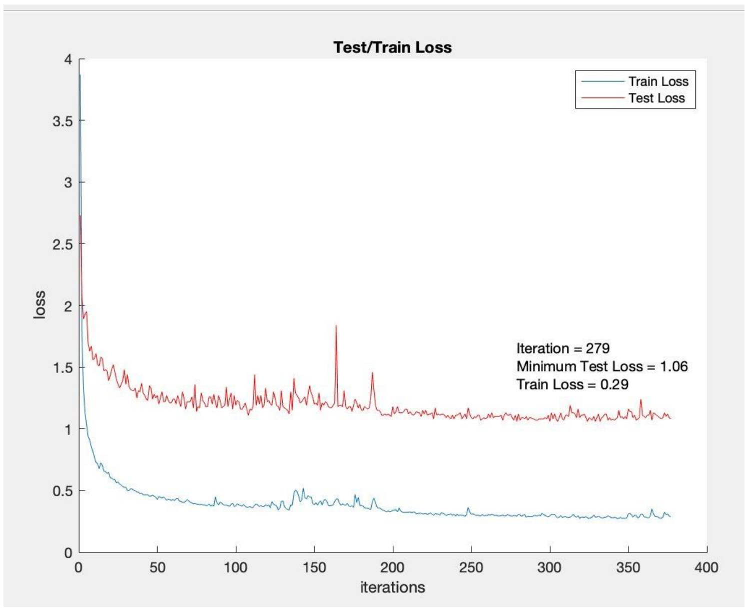


Fig 4.1. First training results (default parameters)

As it can be observed in the fig above, we can assume that the network is suffering a high-variance (overfitting) problem. We can also appreciate a high bias problem, but we decided reducing Test Loss as the main objective, and usually reduce bias involve an increment of variance. As it has been explained before, some actions can be taken to reduce this problem. We have centered on understanding the network parameters and its functions.

In the next sections we will see the performance of parameter tuning and improvement proposal using the network training performance.

4.1. PARAMETER TUNING

In this section we have tested the performance of Open-Unmix model by changing some interesting parameters. The changes were directly oriented to reduce overfitting and to try to reduce the test loss or make test/train loss closer.

BATCH SIZE

There is a tension between batch size and the speed and stability of the learning process. It is also related with the estimation accuracy, so we decided to compare Open-Unmix performances on different batch sizes.

Batch Size	**Total epochs trained***	**Best Epoch Test Loss**
16	481	1.06
32	517	0.98
64	530	0.99
128	327	1.1

Table 4.2. Batch Size system performance comparison

In this case we cannot see relevant difference between trainings in terms of converge speed or stability. Despite of that, we can see an important difference in time per epoch, so increasing batch size accelerates the whole training process. We also have certain advances in terms of Test Loss using 32 or 64 batch size values. Using too low values may affect to the gradient estimation. In contrast, using too high values can produce a degradation in the model generalization ability.

REGULARIZATION PARAMETER TUNING

Regularization is commonly used in NN trainings to reduce overfitting. We tried to increase the weight decay regularization in order to reduce the high-variance problem.

Regularization	**Train Loss**	**Test Loss**	**SDR**
0.00002	0.24	1.02	5.514
0.00004	0.27	1.03	5.466
0.0001	0.35	1.08	5.124
0.001	0.8	1.50	0.548

Table 4.3. Regularization system performance comparison

We can see that the distance between Train Loss and Test (Validation) Loss reduces as regularization increase. The training bias also increase, so regularization tuning it is not useful as an independent measure but it could be interesting when training the system with an augmented dataset, for example.

DIMENSIONALITY REDUCTION

If we analyze the dimensionality reduction in the fully-connected layer (input of the LSTM), we can see that information compression to reduce redundancy and make the model converge faster is not optimal in terms of training results.

LSTM input size	**Total epochs trained ***	**Best Epoch Train Loss**	**Best Epoch Test Loss**
64	433	0.43	1.32
128	638	0.32	1.11
256	649	0.25	1.03
1024	483	0.21	0.97

Table 4.4. Dimensionality reduction system performance comparison

This results show that dimensionality reduction is not affecting or does not seem relevant in terms of convergence velocity. When comparing the Test Loss between the different models,we can affirm that higher input size is a good choice to reduce variance. We can also see that bias decreases when LSTM input size increases.

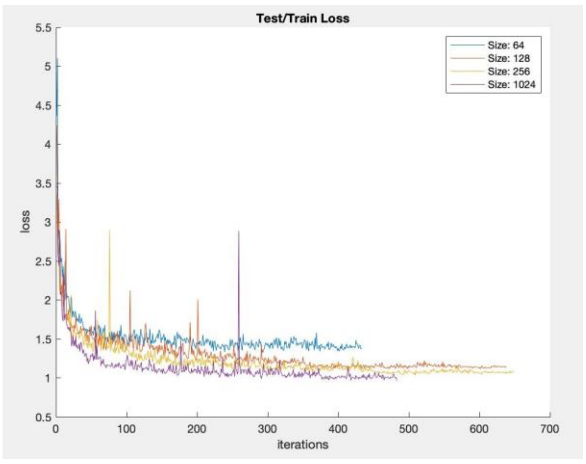


Fig 4.2. Train/Test iteration/loss comparison under dimensionality reduction

4.2. IMPROVEMENT RESULTS

As it has been explained before, the improvements have been focused on filtering for sourceenhancement. We have tried learnable filters of different number of coefficients in the pre-processing stage of the architecture. We implemented a pre-masking stage by assigning learnable weights to the every spectrogram frequency bin. We also analyzed an internal parameter of the network, the frequency bandwidth of the input. We decided to put its results here because it is directly related with spectrum pre-processing.

We have studied the importance of the frequency cropping pre-processing stage. For the voice track we thought that high frequencies were essential for the source extraction. However, results show that we can reduce a lot the frequency bandwidth and inclusive improving Test performance. In the figs below we can see the network performance reducing the bandwidth to 15 kHz, 13 kHz, 11 kHz and 9 kHz (from upper-left corner to lower-right corner).

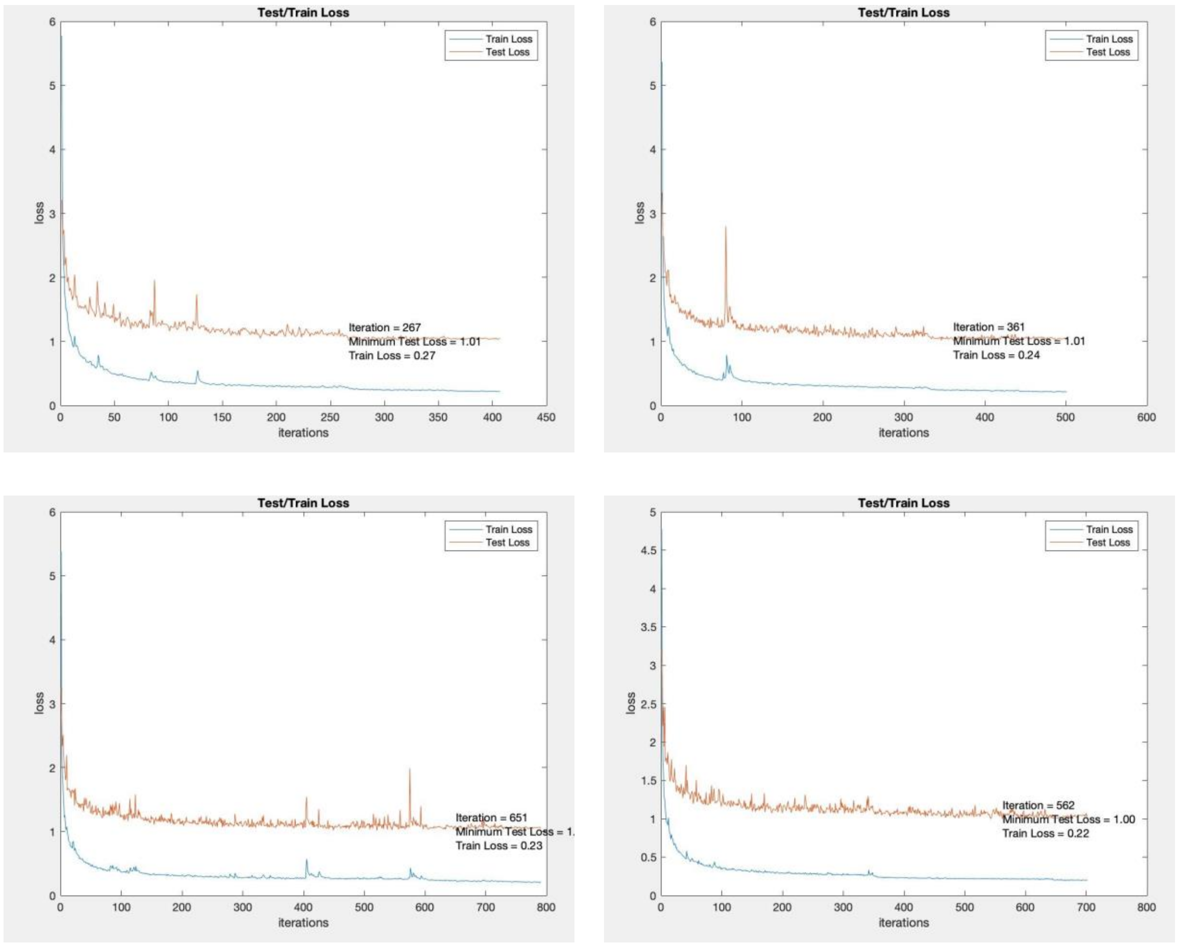


Table 4.5. Train/Test Iteration/Loss after bandwidth variance. Bandwidth: 15 kHz, 13 kHz, 11 kHz and9 kHz (from upper-left corner to lower-right corner)

Before implementing the filters we designed a learnable pre-masking stage. We defined learnable weights applied to every frequency bin of the fft input. As it can be seen below, the training results are similar to other results showed in this section, however the training convergence velocity increases, so we decided not to get stuck into this implementation.

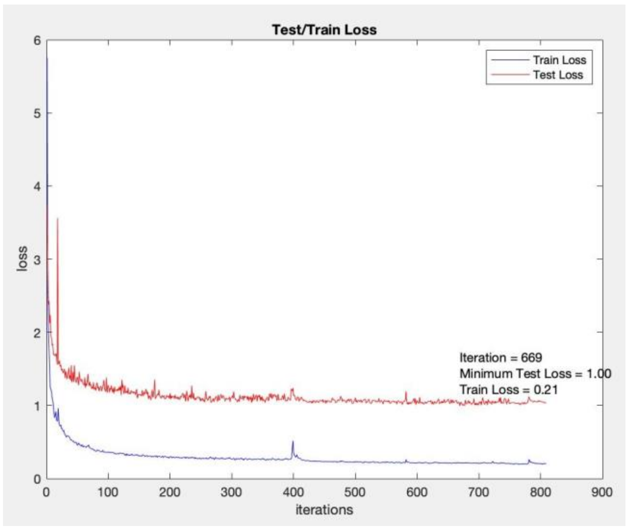
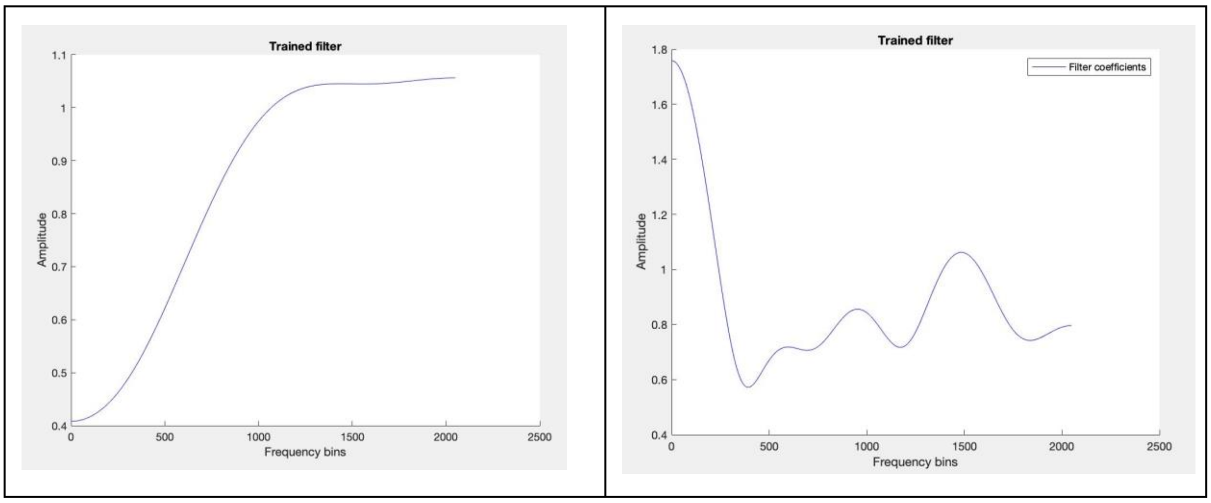
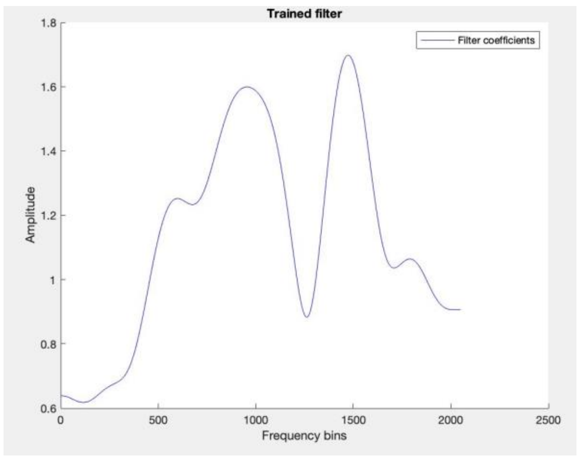


Fig 4.3. Pre-masking model performance

As it is explained in other sections, we have designed a pre-processing learnable filter to try to enhance a concrete target. We have studied its performance with different filter coefficientnumbers.





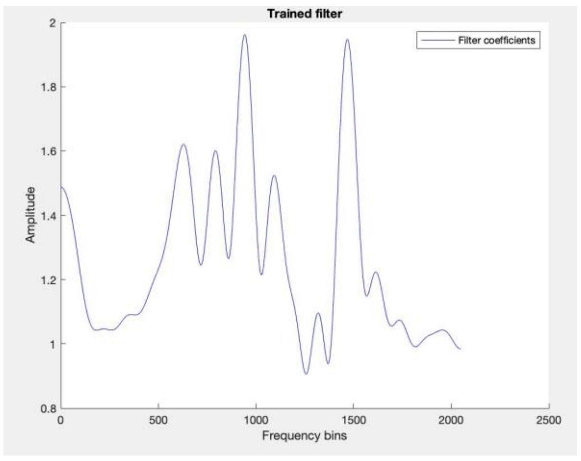


Table 4.6 Learnable filter frequency response of 5, 10, 15 and 35 coefficients (from upper-left cornerto lower-right corner)

Number of filter coefficients	**Best Epoch Train Loss**	**Best Epoch Test Loss**
5	0.29	1.10
10	0.27	1.07
15	0.21	1.00
35	0.21	1.00

Table 4.7. Learnable filter model performance

From the results showed above, we can make some interesting considerations. As the number of coefficients increase, the system seems to have a slightly better performance. Furthermore, increasing a lot the number of coefficients does not provide relevant improvement.

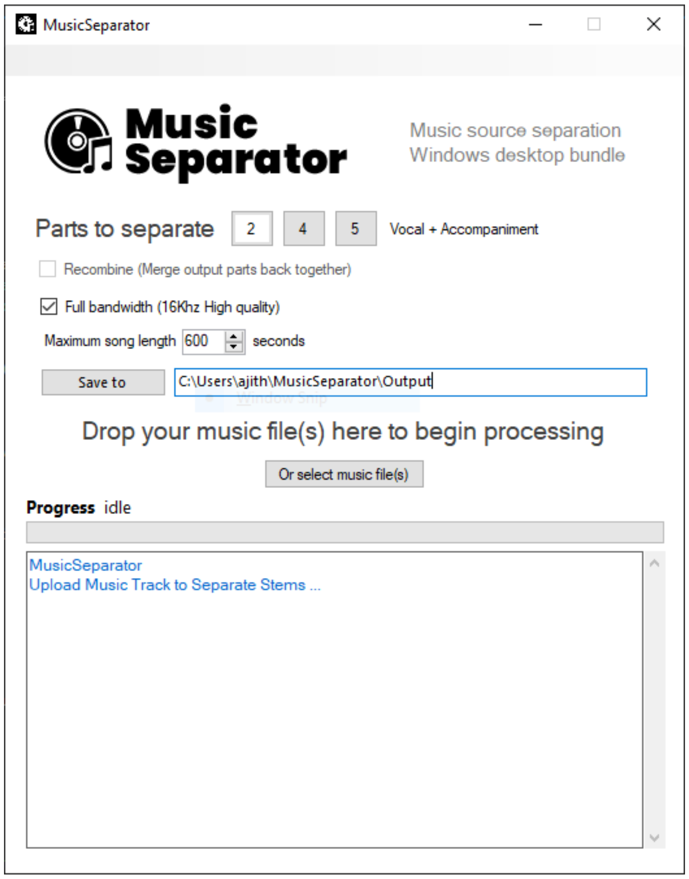
If we analyze the frequency response and of the filters, we can assert that high-pass filtering could be a good option as pre-processing to extract the voice track. This results are comparable with filtering proposal from professional mixing techniques as [20], concretely with the 15 coefficient filter response.

As we can see, the filter response variates a lot between trainings. However, we can see a gap around 12 kHz in all the figs

CHAPTER 5

SYSTEM INTERFACE

5.1. USER INTERFACE DESIGN


MusicSeparator

Music Separator

Music source separation
Windows desktop bundle

Parts to separate 2 4 5 Vocal + Accompaniment

Recombine (Merge output parts back together)

[✓] Full bandwidth (16Khz High quality)

Maximum song length 600 seconds

Save to C:\Users\ajith\MusicSeparator\Output

Drop your music file(s) here to begin processing

Or select music file(s)

Progress idle

MusicSeparator
Upload Music Track to Separate Stems ...

Fig 5.1. GUI for Music Source Separation

The above fig 5.1 represents the GUI for MSS using C# .Net for separating audio track.

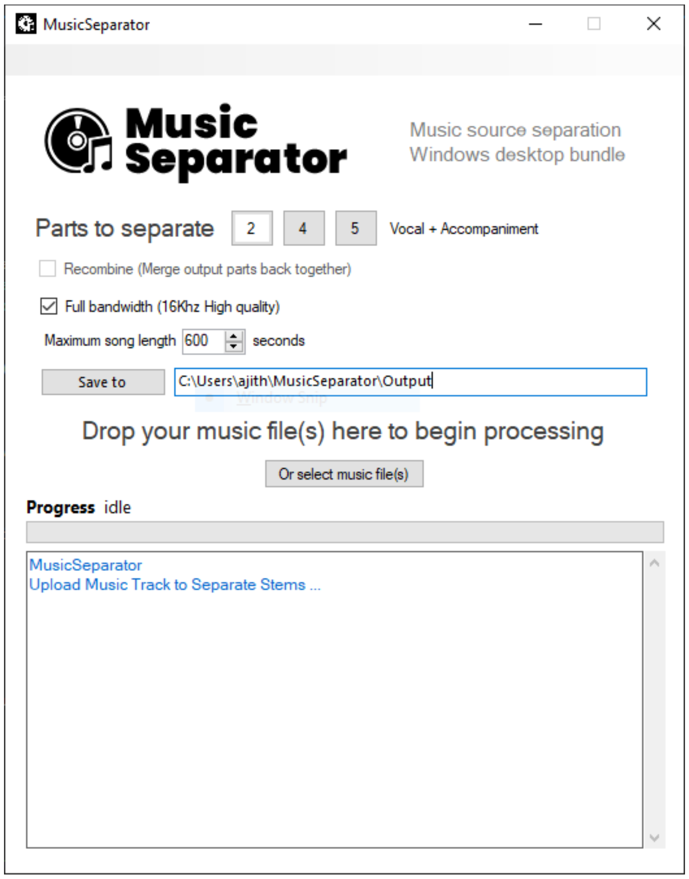
5.2. PARAMETERS IN GUI

5.2.1. PARTS TO SEPARATE

Separate music into stems with the formation of 2 stems,4 stems, 5 stems accordingly.

5.2.1.1. TWO STEM SEPARATION

Two stem separation is a process of separating Vocal and Accompaniment accordingly. Shown in the fig 5.2 below.



MusicSeparator

Music Separator

Music source separation
Windows desktop bundle

Parts to separate 2 4 5 Vocal + Accompaniment

Recombine (Merge output parts back together)

Full bandwidth (16Khz High quality) [✓]

Maximum song length 600 seconds

Save to C:\Users\ajith\MusicSeparator\Output

Drop your music file(s) here to begin processing

Or select music file(s)

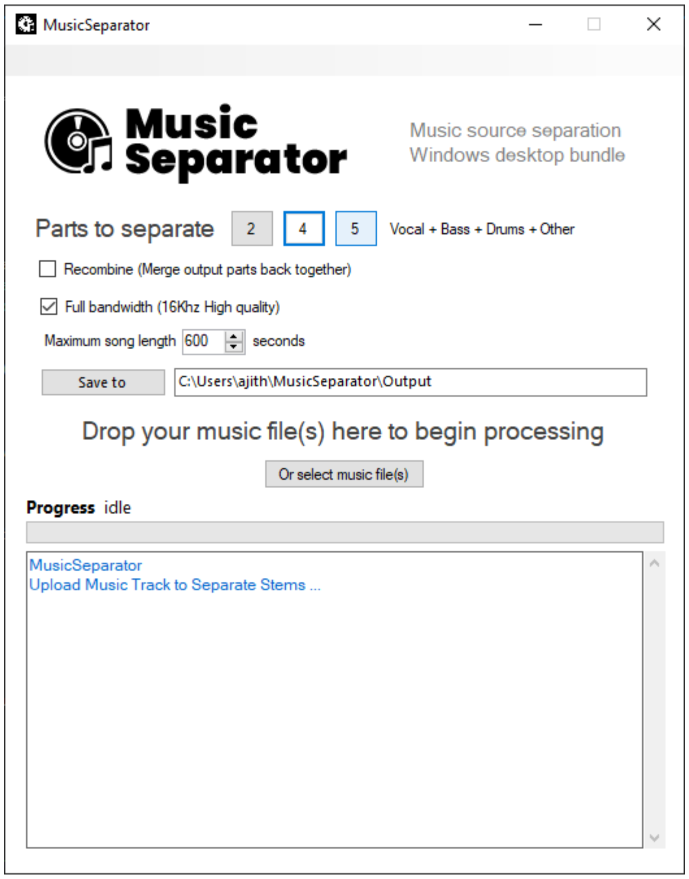
Progress idle

MusicSeparator
Upload Music Track to Separate Stems ...

fig 5.2: Two Stem Separation

5.2.1.2. FOUR STEM SEPARATION

Four stem separation is a process of separating Vocal, Bass, Drums and Others (which are remaining in the given track). Shown in the fig 5.3 below.



Vocal + Bass + Drums + Other

☐ Recombine (Merge output parts back together)

☑ Full bandwidth (16Khz High quality)

Maximum song length 600 seconds

C:\Users\ajith\MusicSeparator\Output

Drop your music file(s) here to begin processing

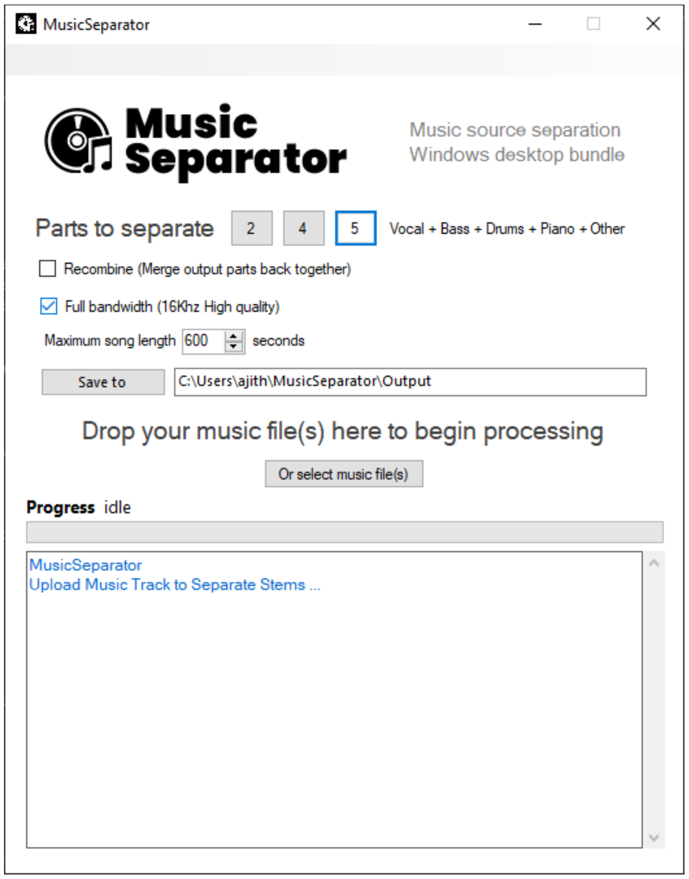
Progress idle

MusicSeparator
Upload Music Track to Separate Stems ...

fig 5.3: Four Stem Separation

5.2.1.3. FIVE STEM SEPARATION

Five stem separation is a process of separating Vocal, Bass, Drums, Piano and Others (which are remaining in the given track). Shown in the fig 5.4 below.



MusicSeparator

Music Separator

Music source separation
Windows desktop bundle

Parts to separate 2 4 [✓ 5] Vocal + Bass + Drums + Piano + Other

☐ Recombine (Merge output parts back together)

☑ Full bandwidth (16Khz High quality)

Maximum song length 600 seconds

Save to C:\Users\ajith\MusicSeparator\Output

Drop your music file(s) here to begin processing

Or select music file(s)

Progress idle

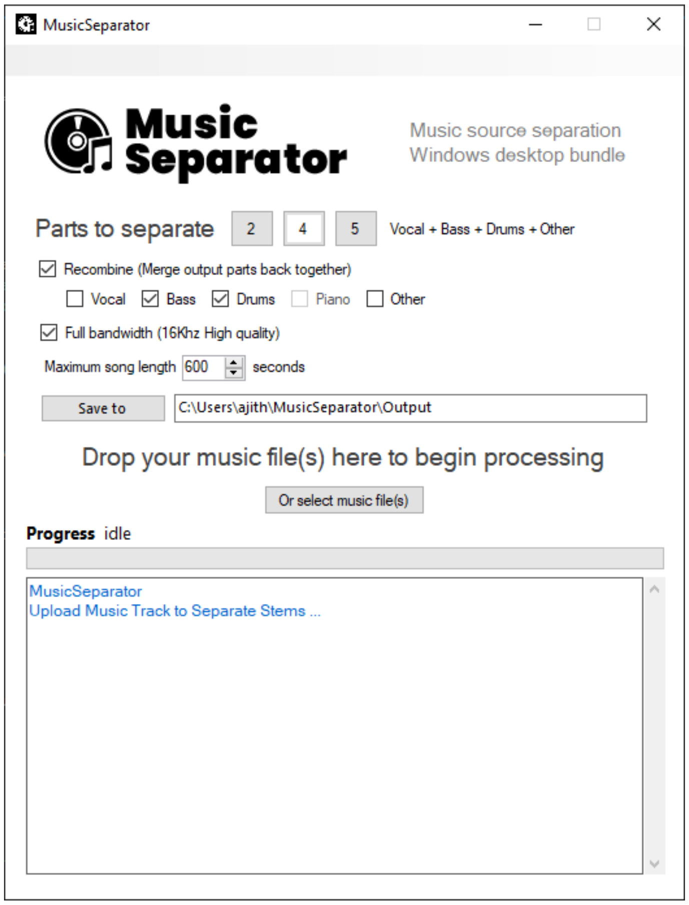
MusicSeparator
Upload Music Track to Separate Stems ...

fig 5.4: 5 Stem Separation

5.2.2. RECOMBINE

Recombine is the feature, which can combine with other stems and save it as a additional stem. Like vocal + drums or drums + bass, etc. Like that we can combine more than 2 stems.

Choose which stems to mix back together (useful if you are learning drums and just want the drums removed from a bunch of songs). Shown in the fig 5.5 below



Vocal + Bass + Drums + Other

☑ Recombine (Merge output parts back together)

☐ Vocal ☑ Bass ☑ Drums ☐ Piano ☐ Other

☑ Full bandwidth (16Khz High quality)

Maximum song length 600 seconds

C:\Users\ajith\MusicSeparator\Output

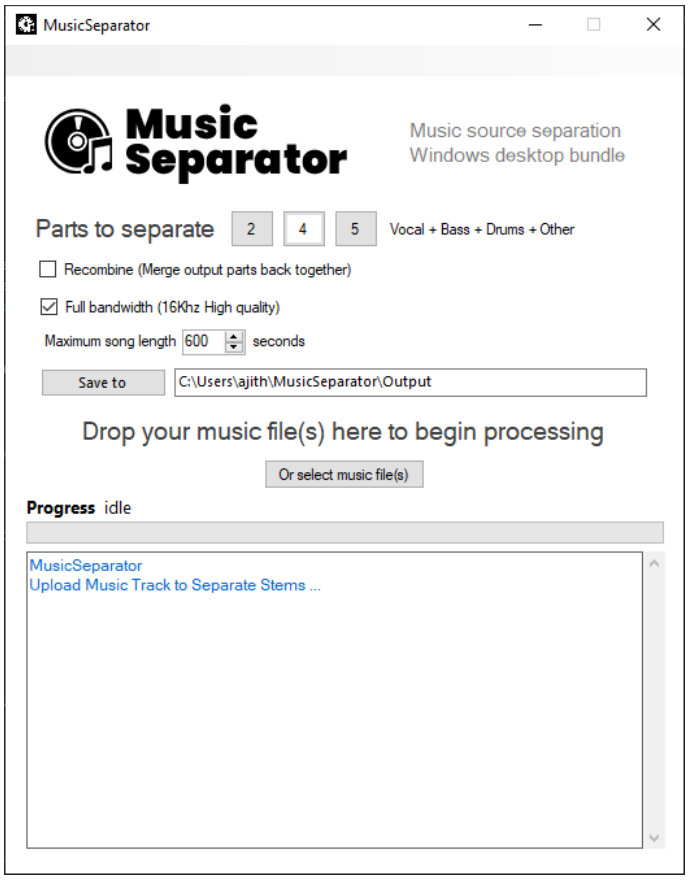
Progress idle

MusicSeparator
Upload Music Track to Separate Stems ...

fig 5.5: Recombine

5.2.3. BANDWIDTH

Bandwidth is a option to choose a 15Kz frequency of high quality output. An audio signal which has a bandwidth of 15 kHz is to be transmitted using digital modulation. Wideband codecs have a typical sample rate of 15 kHz. A rate of 15,000 samples per second is equal to 15,000 Hz (or 15 kHz). Shown in the fig 5.5 below


MusicSeparator

Music Separator

Music source separation
Windows desktop bundle

Parts to separate [2] [✓ 4] [5] Vocal + Bass + Drums + Other

☐ Recombine (Merge output parts back together)

☑ Full bandwidth (16Khz High quality)

Maximum song length 600 seconds

Save to C:\Users\ajith\MusicSeparator\Output

Drop your music file(s) here to begin processing

Or select music file(s)

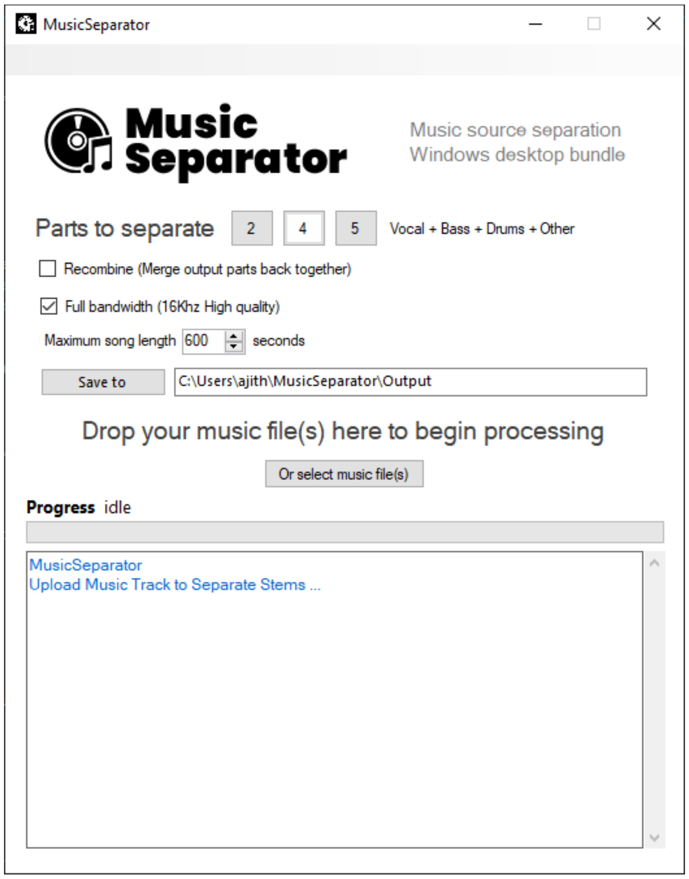
Progress idle

MusicSeparator
Upload Music Track to Separate Stems ...

fig 5.5: Bandwidth

5.2.4. MAXIMUM SONG LENGTH

Maximum song length is a option to adjust the length of the audio track. 500 seconds (10 minutes) is a default song length (as shown in fig 5.7). .


MusicSeparator

Music Separator

Music source separation
Windows desktop bundle

Parts to separate 2 4 5 Vocal + Bass + Drums + Other

☐ Recombine (Merge output parts back together)

☑ Full bandwidth (16Khz High quality)

Maximum song length 600 seconds

Save to C:\Users\ajith\MusicSeparator\Output

Drop your music file(s) here to begin processing

Or select music file(s)

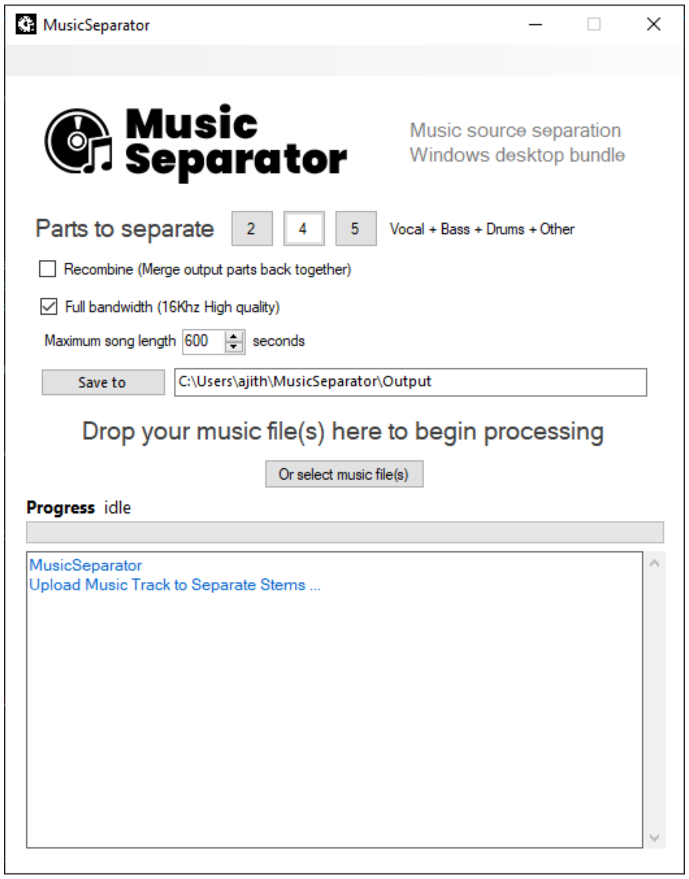
Progress idle

MusicSeparator
Upload Music Track to Separate Stems ...

fig 5.7: maximum song length

5.2.5. SAVE TO

Save to (as shown in fig 5.8) is a option to save the output in the user directory path (as shown in fig 5.9).


MusicSeparator

Music source separation
Windows desktop bundle

Parts to separate 2 4 5 Vocal + Bass + Drums + Other

☐ Recombine (Merge output parts back together)

☑ Full bandwidth (16Khz High quality)

Maximum song length 600 seconds

Save to C:\Users\ajith\MusicSeparator\Output

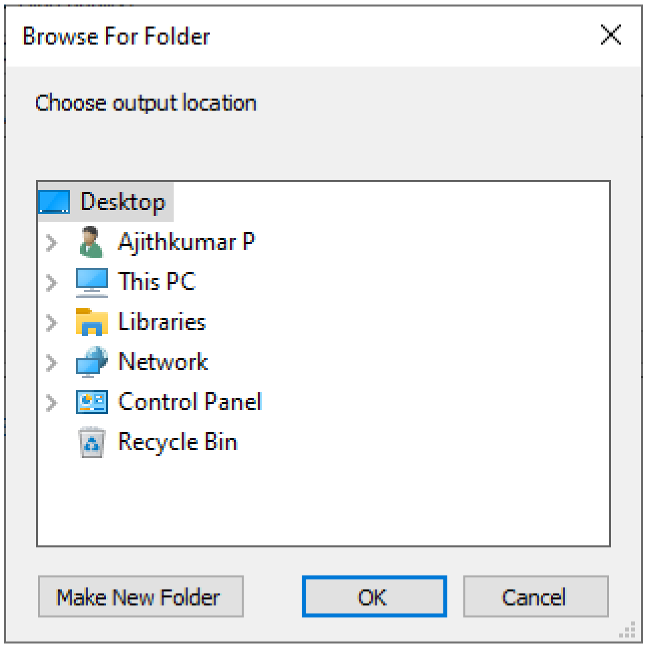
Drop your music file(s) here to begin processing

Or select music file(s)

Progress idle

```
MusicSeparator
Upload Music Track to Separate Stems ...
```

fig 5.8: save to


Browse For Folder

Choose output location

fig 5.9: save to (directory)

5.2.5. UPLOAD MUSIC FILES

User can upload the music file by in two ways :

1) Select music file(s). (as shown in fig 5.10).

2) Drag and Drop your music file(s). (as shown in fig 5.11)

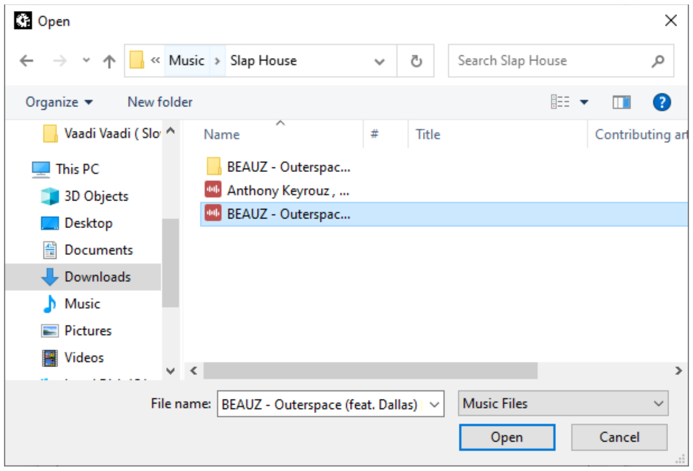


fig 5.10: upload music file(s)

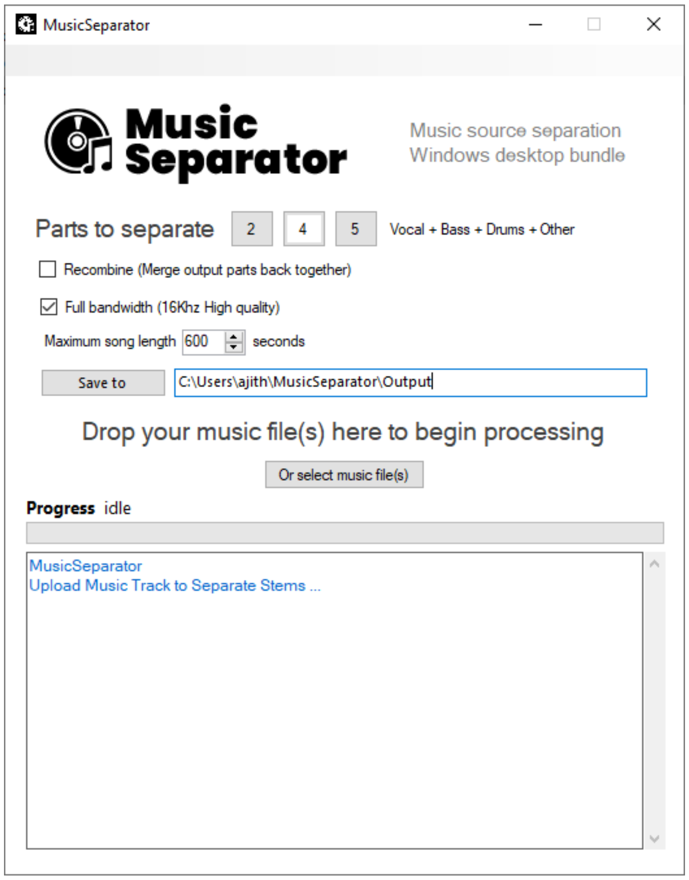
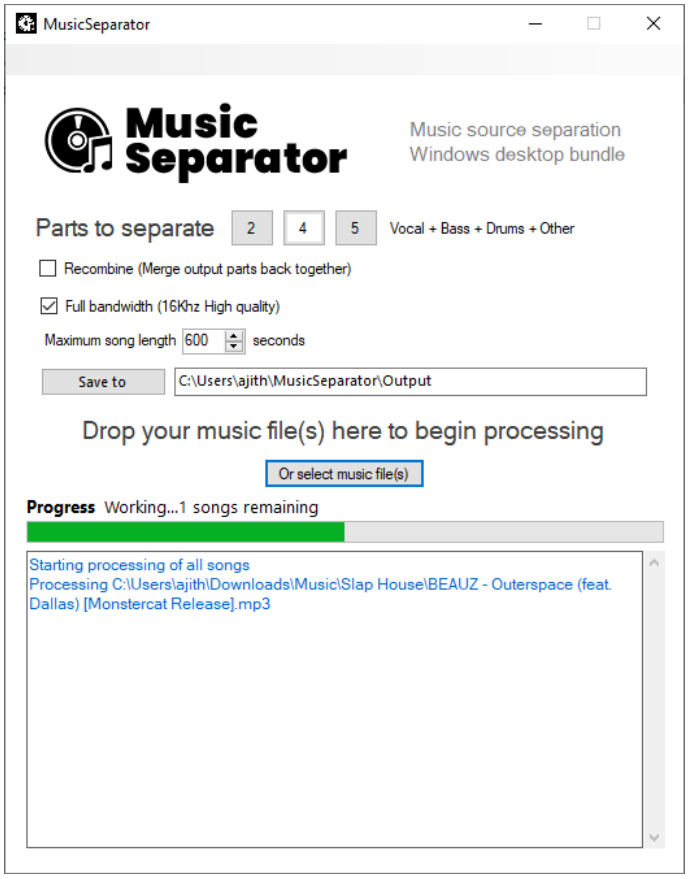

MusicSeparator
Music Separator
Music source separation
Windows desktop bundle
Parts to separate
2
4
5
Vocal + Bass + Drums + Other
Recombine (Merge output parts back together)
Full bandwidth (16Khz High quality)
Maximum song length
600
seconds
Save to
C:\Users\ajith\MusicSeparator\Output
Drop your music file(s) here to begin processing
Or select music file(s)
Progress
idle
MusicSeparator
Upload Music Track to Separate Stems ...

fig 5.11: drag and drop music file(s)

5.2.7. PROGRESS BAR

Progress bar shows the progress of the project (as shown in fig 5.12) and the status shows below the progress bar (as shown in fig 5.13).



Parts to separate 2 4 5 Vocal + Bass + Drums + Other

☐ Recombine (Merge output parts back together)

☑ Full bandwidth (16Khz High quality)

Maximum song length 600 seconds

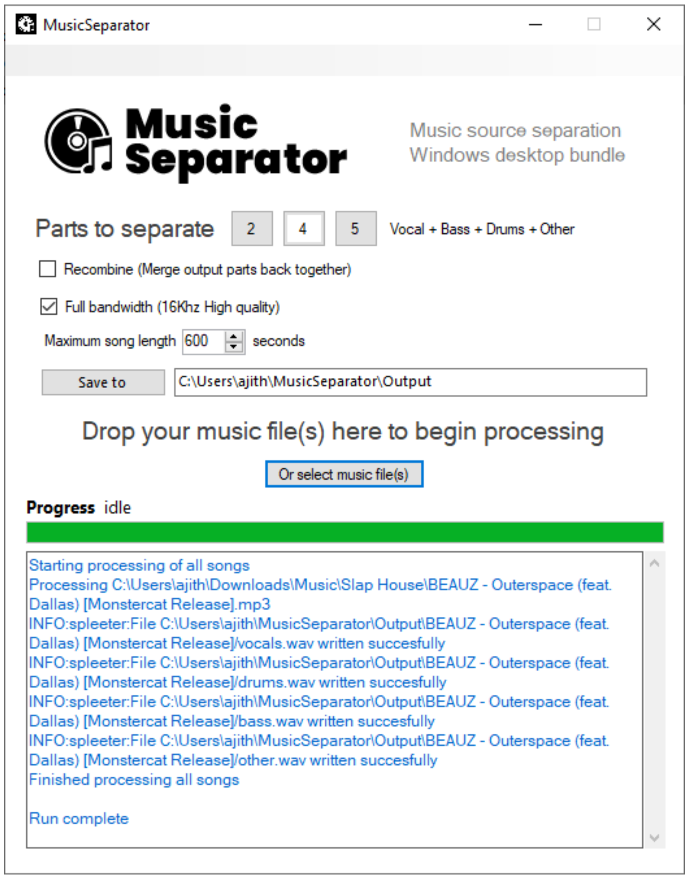
C:\Users\ajith\MusicSeparator\Output

Drop your music file(s) here to begin processing

Progress Working...1 songs remaining

```
Starting processing of all songs
Processing C:\Users\ajith\Downloads\Music\Slap House\BEAUZ - Outerspace (feat.
Dallas) [Monstercat Release].mp3
```

fig 5.12: progress bar.



Parts to separate 2 4 5 Vocal + Bass + Drums + Other

[ ] Recombine (Merge output parts back together)

[✓] Full bandwidth (16Khz High quality)

Maximum song length 600 seconds

Save to C:\Users\ajith\MusicSeparator\Output

Progress idle

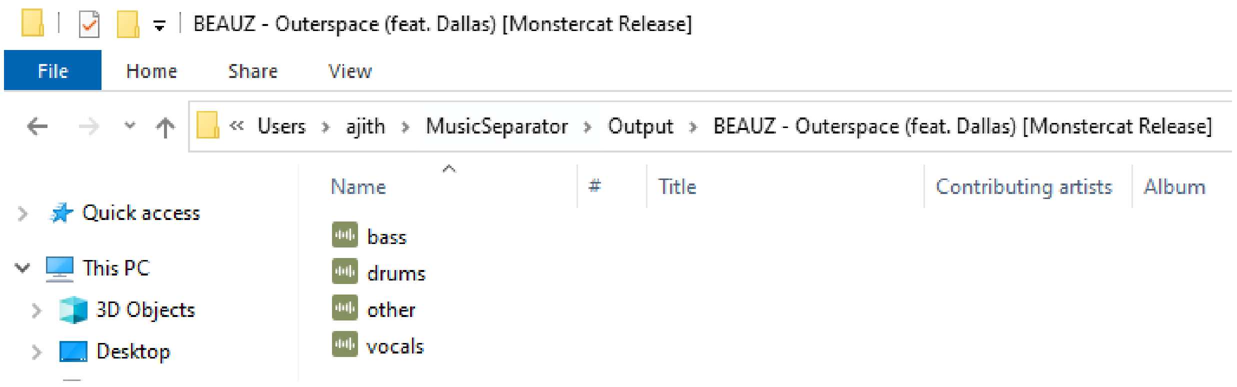
```
Starting processing of all songs
Processing C:\Users\ajith\Downloads\Music\Slap House\BEAUZ - Outerspace (feat.
Dallas) [Monstercat Release].mp3
INFO:spleeter:File C:\Users\ajith\MusicSeparator\Output\BEAUZ - Outerspace (feat.
Dallas) [Monstercat Release]/vocals.wav written succesfully
INFO:spleeter:File C:\Users\ajith\MusicSeparator\Output\BEAUZ - Outerspace (feat.
Dallas) [Monstercat Release]/drums.wav written succesfully
INFO:spleeter:File C:\Users\ajith\MusicSeparator\Output\BEAUZ - Outerspace (feat.
Dallas) [Monstercat Release]/bass.wav written succesfully
INFO:spleeter:File C:\Users\ajith\MusicSeparator\Output\BEAUZ - Outerspace (feat.
Dallas) [Monstercat Release]/other.wav written succesfully
Finished processing all songs

Run complete
```

fig 5.13: status

5.3. OUTPUT

The separated stems saves in the given user directory path accordingly (as shown in the fig 5.14).


BEAUZ - Outerspace (feat. Dallas) [Monstercat Release]
File
Home
Share
View
Users › ajith › MusicSeparator › Output › BEAUZ - Outerspace (feat. Dallas) [Monstercat Release]
Name
#
Title
Contributing artists
Album
Quick access
This PC
3D Objects
Desktop

bass
drums
other
vocals

fig 5.14: output

CHAPTER 6
CONCLUSION AND FUTURE SCOPE

6.1 CONCLUSION

Improving Open-Unmix performance has been a difficult task to manage. All the modifications were carefully studied and chosen in order to improve the system performance without compromising the training speed and the overall system simplicity and understandability. Therefore, finding and implementing modifications has been limited by the system main advantages.

The result comparison with the state-of-art is affected by the performance of the system post-processing Norbert Wiener Filter. As explained in 3.2.1, Norbert filter works with all the individual stems obtained from the mix (vocals, bass, drums and others). When the filter does not have other stem information, it uses an internal approximation of the mix without the voice. This adaptation makes the metrics differ from the metrics found in the state-of-art of the project. So centering the trainings on the voice track has make the performance of the Norbert filter poorer. This is the one of the reasons because of we have used the training loss/iteration metric to compare the different system performances.

Some implementations and purposes have been excluded from the results because it directly affected to the characteristics explained above. A clear example would be a pre-processing temporal filtering design that dragged the training time from two days to morethan a month and a half. The sequential filtering was specially slow in GPU. We tried to move it into the pre-processing stage but it did not perform as expected in terms of training loss and speed.

Moreover, parameter tuning is a very complex task. Finding the optimal values of the parameters chosen has been a challenging task. Parallel training has helped to reduce the waiting time. Nevertheless, it has been difficult to control the performance when changing the different parameters (two or more parameters) in the same training example. Helpfully, Open-Unmix is a relatively small network and the internal network parameters to tune were not too hard to study separately. We have centered the study on the most common parameters or the parameters that we thought to be most important for the training performance.

The results show a slightly upgrade in source separation performance by using filters. In spite of that, more accurate filtering or parametric equalization could work better for this concrete problem. As it is mentioned in 4.2, reducing the frequency bandwidth could help with source extraction in tracks like bass or voice, but with tracks with a very wide frequency range such as drums it will not. Other alternatives such as panning coefficients or other stereo audio based source enhancement methods could be implemented for a better separation. Music repetitive structure has also been used to help source separation.

Filtering would be an interesting option to try in the post-processing stage to reduce final audio artificial interference. Furthermore, different spectrogram signal reconstruction, such as Griffin&Lim, or more complex structures, such as wavenets, could perform better obtaining the final unique source audio.

Up to now, end-to-end systems seem to perform better in MSS than pre-processing waveform algorithms. This could change introducing some improvements or exploring different features. Concretely, in spectrogram based algorithms efforts are made to use the FFT phase information as an input or extra feature of the system to help with the source extraction.

It is well known that this kind of problems depend a lot on how much data are available to train the system. So database augmentation is a must when we think on the next steps. Various techniques have been followed in the state-of-art of this project, such as pitch modification or time stretching. Another option would be a multiple dataset adaptation or directly obtaining new tracks in a further research task.

The main objectives of the project have been accomplished. Focusing on the voice track hashelped in the result comparison but has also compromised the global vision of source separation algorithms on other tracks like bass or drums. Taking a chance on open source algorithm implementations has been a good choice. Open source community has helped alot in terms of research and information accessibility.

6.2 FUTURE SCOPE

The developed algorithm and methods can be applied for intelligent music separation application platform, singer identification, musical information retrieval, melody extraction and music instrument separation.

Musical Source Separation is a challenging research area with numerous real-world applications. Due to both the nature of the musical sources and the very particular processes used to create music recordings, MSS has many unique features and problems which make it distinct from other types of source separation problems. This is further complicated by the need to achieve separations which sound good in a perceptual sense.

While the quality of MSS has greatly improved in the last decade, several critical challenges remain. Firstly, audible artifacts are still produced by most algorithms. Possible research directions to reduce artifacts include the use of phase retrieval techniques to estimate the phase of the target source, the use of feature representations that better match human perception, allowing models to concentrate on the parts of the sounds that are most relevant for human listeners, and the exploration of MSS systems that model the signal directly in time domain as waveforms.

REFERENCES

[1] Cano, E., Fitzgerald, D., Liutkus, A., Plumbley, M., Robert-Stöter, F. (2019). Musical Source Separation: An Introduction. IEEE Signal Processing Magazine, 36 (1), 31-40.

[2] Abdi. H., Williams, L.J. (2010). Principal component analysis. Wiley Interdisciplinary Reviews: Computational Statistics, 2 (4), 433-459.

[3] Inderjit S. Dhillon and Suvrit Sra. (2005). Generalized nonnegative matrix approximations with Bregman divergences. In Proceedings of the 18th International Conference on Neural Information Processing Systems (NIPS'05) (283–290). Cambridge, MA, USA: MIT Press.

[4] Stöter, E., Liutkus, A., Ito, N. (2018). 14th International Conference on Latent Variable Analysis and Signal Separation (LVA/ICA 2018).

[5] Quiza, R., Davim, J. (2009). Computational modeling of machining systems.

[6] Nicholson, C. (n.d.). A Beginner's Guide to Neural Networks and Deep Learning. Available: https://pathmind.com/wiki/neural-network [Accessed: 1 June 2020]

[7] Saha, S. (2016). A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 [Accessed: May 2020]

[8] Olah, C. (2015). Understanding LSTM Networks. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ [Accessed: May 2020]

[9] Microsoft Ventures. (2014). Intro to Machine Learning. Available: https://www.slideshare.net/microsoftventures/microsoft-ventures-workshop [Accessed: 10 June 2020]

[10] Stoller, D., Ewert, S., Dixon, S. (2018). Wave-u-net: A multi-scale neural network for end-to-end audio source separation. Proc. Int. Soc. Music Inf. Retrieval, (334–340).

[11] Défossez, A., Usunier, N., Bottou, L., Bach, F. (2019). Music Source Separation in the Waveform Domain. Available: https://hal.archives-ouvertes.fr/hal-02379796/document [Accessed: April 2020]

[12] Luo, Y., Mesgarani, N. (2019). Conv-TasNet: Surpassing Ideal Time–Frequency Magnitude Masking for Speech Separation. IEEE/ACM Transactions on Audio, Speech, and Language Processing, 27.8 (1256–1266).

[13] Hennequin, R., Khlif, A., Voituret, F., Moussallam, M. (2019). SPLEETER: A FAST AND STATE-OF-THE-ART MUSIC SOURCE SEPARATION TOOL WITH PRE-TRAINED MODELS. Available: https://archives.ismir.net/ismir2019/latebreaking/000036.pdf [Accessed: April 2020].

[14] Jansson A., Humphrey E. J., Montecchio, N., Bittner R. M., Kumar, A., Weyde, T. (2017). Singing Voice Separation with Deep U-Net Convolutional Networks. ISMIR, (745-751).

[15] Stöter F., Uhlich, S., Liutkus, A., Mitsufuji, Y. (2019). Open-Unmix - A Reference Implementation for Music Source Separation. Available : https://sigsep.github.io/open-unmix/ [Accessed: March-June 2020]

[16] Vincent, E., Gribonval R., Fevotte, C. (2006). Performance measurement in blind audio source separation," in IEEE Transactions on Audio, Speech, and Language Processing, 14-4 (1462-1469).

[17] Rafii, Z., Liutkus, A., Stöter, F., Mimilakis, S. I., Bittner, R. (2017) MUSDB18 - a corpus for music separation. https://sigsep.github.io/datasets/musdb.html [Accessed: March-June 2020]

[18] Stöter, F., Liutkus, A., Inria and LIRMM. (2019). Open-Unmix - A Reference Implementation for Music Source Separation. Available: https://sigsep.github.io/open-unmix/#using-the-pytorch-version [Accessed: March-June 2020]

[19] Hu, Z., Ma, X., Liu, Zhengzhong H., Xing, E. (2016). Harnessing Deep Neural Networks with Logic Rules. https://arxiv.org/pdf/1603.06318.pdf [Accessed: April 2020]

[20] Moxey, J. (n.d.). Songstuff Music Resources - EQ Frequencies. Available: https://www.songstuff.com/recording/article/eq_frequencies/ [Accessed: June 2020]

[21] Avendano, C. (2003). Frequency-domain source identification and manipulation in stereo mixes for enhancement, suppression and re-panning applications. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics. (55-58).

[22] Rafii, Z., Pardo, B. (2011). A simple music/voice separation method based on the extraction of the repeating musical structure. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). (221-224).

[23] Oord, A. V. D., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., Kavukcuoglu, K. (2016). WaveNet: a generative model for raw audio. https://arxiv.org/pdf/1609.03499.pdf [Accessed: May 2020].

APPENDICES

This section will include the Source Code develop of a GUI for a MSS. C# .Net is used to develop a GUI for a MSS for frontend.

CODING FORM

```
using System.*;
namespace MusicSeparator
{
   public partial class Form1 : Form
   {
      private string stem_count = "2";
      private string mask_extension = "average";
      private string storage = ""
      private string path_python = "";    //needs to be the SpleeterGUI folder, not python
      private string current_songname = "";
      private int files_remain = 0;
      private List<string> files_to_process = new List<string>();
      private Boolean run_silent = true;
      private String gui_version = "";
      IDictionary<string, string> langStr = new Dictionary<string, string>();
      public Form1()
      {  InitializeComponent();}
      private void Form1_Load(object sender, EventArgs e)
      { this.AllowDrop = true;
         this.DragEnter += new DragEventHandler(Form1_DragEnter);
         this.DragDrop += new DragEventHandler(Form1_DragDrop);}
      private void Form1_Shown(object sender, EventArgs e)
      { LoadStuff();}
      public void LoadStuff()

      void Form1_DragEnter(object sender, DragEventArgs e)
      {if (e.Data.GetDataPresent(DataFormats.FileDrop)) e.Effect = DragDropEffects.Copy;}
      void Form1_DragDrop(object sender, DragEventArgs e)
```

```
        {
            if (files_remain == 0)
            {
                textBox1.Text = "";
                if (txt_output_directory.Text == "")
                {
                    MessageBox.Show(langStr["output_message"]);
                    return;
                }
                string[] files = (string[])e.Data.GetData(DataFormats.FileDrop);
                files_remain = 0;
                foreach (string file in files)
                {
                    files_to_process.Add(file);
                    files_remain++;
                }
                textBox1.AppendText(langStr["starting_all"] + "\r\n");
                progressBar1.Maximum = files_remain + 1;
                progressBar1.Value = 0;
                progress_txt.Text = langStr["starting"] + "..." + files_remain + " " +
langStr["songs_remaining"];
                next_song();
            }
            else
            {
                System.Media.SystemSounds.Asterisk.Play();
            }
        }

        private void next_song()
        {
            //begins the spleeting function on the next song in the queue
            if (files_remain > 0)
            {
                run_silent = false;
                //string pyPath = storage + @"\python\python.exe";
                string pyPath = path_python + @"\python.exe";
                string filename = files_to_process[0];
                progressBar1.Value = progressBar1.Value + 1;
                System.IO.File.WriteAllText(storage + @"\config.json", get_config_string());
                textBox1.AppendText(langStr["processing"] + " " + filename + "\r\n");
                progress_txt.Text = langStr["working"] + "..." + files_remain + " "+
langStr["songs_remaining"];
                ProcessStartInfo processStartInfo = new ProcessStartInfo(pyPath, @" -W ignore -m spleeter
separate  -o " + (char)34 + txt_output_directory.Text + (char)34 + " -d " + (duration.Value).ToString() +
" -p " + (char)34 + storage + @"\config.json" + (char)34 + " " + (char)34 + filename + (char)34);
                processStartInfo.WorkingDirectory = storage;
                processStartInfo.UseShellExecute = false;
                processStartInfo.ErrorDialog = false;
                processStartInfo.RedirectStandardOutput = true;
                processStartInfo.RedirectStandardError = true;
                processStartInfo.CreateNoWindow = true;
                files_to_process.Remove(filename);  } }
        private void Button2_Click(object sender, EventArgs e)
        {
            //prompt user for output folder
            var folderBrowserDialog1 = new FolderBrowserDialog();
            folderBrowserDialog1.ShowNewFolderButton = true;
            folderBrowserDialog1.Description = langStr["set_output"];
            DialogResult result = folderBrowserDialog1.ShowDialog();
            if (result == DialogResult.OK)
            {
                txt_output_directory.Text = folderBrowserDialog1.SelectedPath;
                Properties.Settings.Default.output_location = txt_output_directory.Text;
                Properties.Settings.Default.Save();
            }
```

```
    else
    {
        txt_output_directory.Text = "";
    }
}
private void parts_btn2_Click(object sender, EventArgs e)
{
    //set the stem mode to 2
    parts_label.Text = langStr["vocal_accompaniment"];
    parts_btn2.UseVisualStyleBackColor = false;
    parts_btn4.UseVisualStyleBackColor = true;
    parts_btn5.UseVisualStyleBackColor = true;
    stem_count = "2";
    update_checks();
}

private void parts_btn4_Click(object sender, EventArgs e)
{
    //set the stem mode to 4
    parts_label.Text = langStr["vocal_bass_drums_other"];
    parts_btn2.UseVisualStyleBackColor = true;
    parts_btn4.UseVisualStyleBackColor = false;
    parts_btn5.UseVisualStyleBackColor = true;
    stem_count = "4";
    update_checks();
}
private void parts_btn5_Click(object sender, EventArgs e)
{
    //set the stem mode to 5
    parts_label.Text = langStr["vocal_bass_drums_piano_other"];
    parts_btn2.UseVisualStyleBackColor = true;
    parts_btn4.UseVisualStyleBackColor = true;
    parts_btn5.UseVisualStyleBackColor = false;
```

```
        stem_count = "5";
        update_checks();
    }
        if (stem_count == "2")
        {
            chkRecombine.Checked = false;
            chkRecombine.Enabled = false;
            pnlRecombine.Height = 20;
            pnlMain.Location = new Point(12, 182);
            this.Height = 677;
        }
        else
        {
            chkRecombine.Enabled = true;

            if (chkRecombine.Checked)
            {
                pnlRecombine.Height = 50;
                pnlMain.Location = new Point(12, 202);
                this.Height = 697;
            }
            else
            {
                pnlRecombine.Height = 20;
                pnlMain.Location = new Point(12, 182);
                this.Height = 677;
                chkRPartVocal.Checked = false;
                chkRPartBass.Checked = false;
                chkRPartDrums.Checked = false;
                chkRPartPiano.Checked = false;
                chkRPartOther.Checked = false;
            }
            switch (stem_count)
            {
               case "4":
                  chkRPartVocal.Enabled = true;
                  chkRPartBass.Enabled = true;
                  chkRPartDrums.Enabled = true;
                  chkRPartPiano.Enabled = false;
                  chkRPartOther.Enabled = true;
                  break;
               case "5":
                  chkRPartVocal.Enabled = true;
                  chkRPartBass.Enabled = true;
                  chkRPartDrums.Enabled = true;
                  chkRPartPiano.Enabled = true;
                  chkRPartOther.Enabled = true;
                  break;}}  }
                  }}}
```

CODING FORM DESIGN

```
namespace MusicSeparator
{
   partial class Form1
   {
      private System.ComponentModel.IContainer components = null;
      protected override void Dispose(bool disposing)
      {
         if (disposing && (components != null))
         {
            components.Dispose();
         }
         base.Dispose(disposing);
      }
```

```
        private void InitializeComponent()
        {
            System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(Form1));
            this.fileToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.exitToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.mnuLanguage = new System.Windows.Forms.ToolStripMenuItem();
            this.advancedToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.setPythonPathToolStripMenuItem1 = new System.Windows.Forms.ToolStripMenuItem();
            this.helpToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.helpFAQToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.checkSpleeterGUIUpdateToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.spleeterupgradeToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.toolStripMenuItem3 = new System.Windows.Forms.ToolStripSeparator();
            this.spleeterGithubPageToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.makenItSoToolStripMenuItem1 = new System.Windows.Forms.ToolStripMenuItem();
            this.btnSaveTo = new System.Windows.Forms.Button();
            this.txt_output_directory = new System.Windows.Forms.TextBox();
            this.chkFullBandwidth = new System.Windows.Forms.CheckBox();
            this.textBox1 = new System.Windows.Forms.TextBox();
            this.lblDroptext = new System.Windows.Forms.Label();
            this.lblSlogan1 = new System.Windows.Forms.Label();
            this.progressBar1 = new System.Windows.Forms.ProgressBar();
            this.lblSlogan2 = new System.Windows.Forms.Label();
            this.lblPartsTitle = new System.Windows.Forms.Label();
            this.parts_btn2 = new System.Windows.Forms.Button();
            this.parts_btn4 = new System.Windows.Forms.Button();
            this.parts_btn5 = new System.Windows.Forms.Button();
            this.pnlMain = new System.Windows.Forms.Panel();
            this.pnlRecombine = new System.Windows.Forms.Panel();
            this.chkRPartOther = new System.Windows.Forms.CheckBox();
            this.chkRPartPiano = new System.Windows.Forms.CheckBox();
            this.chkRPartDrums = new System.Windows.Forms.CheckBox();
            this.chkRPartBass = new System.Windows.Forms.CheckBox();
            this.chkRPartVocal = new System.Windows.Forms.CheckBox();
            this.pictureBox1 = new System.Windows.Forms.PictureBox();
            this.stems4 = new System.Windows.Forms.RadioButton();
            this.stems2 = new System.Windows.Forms.RadioButton();
            this.stems5 = new System.Windows.Forms.RadioButton();
            this.menuStrip1 = new System.Windows.Forms.MenuStrip();
            this.pnlMain.SuspendLayout();
            ((System.ComponentModel.ISupportInitialize)(this.duration)).BeginInit();
            this.pnlRecombine.SuspendLayout();
            ((System.ComponentModel.ISupportInitialize)(this.pictureBox1)).BeginInit();
            this.SuspendLayout();
            // btnSaveTo
            this.btnSaveTo.AccessibleDescription = "Choose folder to save separated files to";
            this.btnSaveTo.AccessibleName = "Save to";
            this.btnSaveTo.AccessibleRole = System.Windows.Forms.AccessibleRole.PushButton;
            this.btnSaveTo.Font = new System.Drawing.Font("Segoe UI", 8.25F);
            this.btnSaveTo.ForeColor = System.Drawing.Color.Black;
            this.btnSaveTo.Location = new System.Drawing.Point(15, 67);
            this.btnSaveTo.Name = "btnSaveTo";
            this.btnSaveTo.Size = new System.Drawing.Size(97, 22);
            this.btnSaveTo.TabIndex = 8;
            this.btnSaveTo.Text = "Save to";
            this.btnSaveTo.UseVisualStyleBackColor = true;
            this.btnSaveTo.Click += new System.EventHandler(this.Button2_Click);
            // txt_output_directory //
            this.txt_output_directory.AccessibleDescription = "shows output directory location";
            this.txt_output_directory.AccessibleName = "output directory display";
            this.txt_output_directory.AccessibleRole = System.Windows.Forms.AccessibleRole.StaticText;
            this.txt_output_directory.Font = new System.Drawing.Font("Segoe UI", 8.25F);
            this.txt_output_directory.ForeColor = System.Drawing.Color.Black;
            this.txt_output_directory.Location = new System.Drawing.Point(118, 67);
            this.txt_output_directory.Name = "txt_output_directory";
            this.txt_output_directory.Size = new System.Drawing.Size(364, 22);
            this.txt_output_directory.TabIndex = 9;
            // chkFullBandwidth //
            this.chkFullBandwidth.AccessibleName = "full bandwidth";
            this.chkFullBandwidth.AccessibleRole =
            this.chkFullBandwidth.AutoSize = true;
            this.chkFullBandwidth.Checked = true;
            this.chkFullBandwidth.CheckState = System.Windows.Forms.CheckState.Checked;
            this.chkFullBandwidth.ForeColor = System.Drawing.Color.Black;
            this.chkFullBandwidth.Location = new System.Drawing.Point(15, 12);
            this.chkFullBandwidth.Name = "chkFullBandwidth";
            this.chkFullBandwidth.Size = new System.Drawing.Size(191, 17);
            this.chkFullBandwidth.TabIndex = 7;
            this.chkFullBandwidth.Text = "Full bandwidth (16Khz High quality)";
            this.chkFullBandwidth.UseVisualStyleBackColor = true;
            this.chkFullBandwidth.CheckedChanged += new
            // parts_btn2 //
            this.parts_btn2.AccessibleDescription = "Separate song in 2 parts";
            this.parts_btn2.AccessibleName = "Two parts";
            this.parts_btn2.AccessibleRole = System.Windows.Forms.AccessibleRole.PushButton;
            this.parts_btn2.Cursor = System.Windows.Forms.Cursors.Hand;
            this.parts_btn2.ForeColor = System.Drawing.Color.Black;
            this.parts_btn2.Location = new System.Drawing.Point(173, 127);
            this.parts_btn2.Name = "parts_btn2";
            this.parts_btn2.Size = new System.Drawing.Size(34, 29);
            this.parts_btn2.TabIndex = 3;
            this.parts_btn2.Text = "2";
            this.parts_btn2.UseVisualStyleBackColor = false;
            this.parts_btn2.Click += new System.EventHandler(this.parts_btn2_Click);
            // parts_btn4 //
            this.parts_btn4.AccessibleDescription = "Separate song in 4 parts";
            this.parts_btn4.AccessibleName = "four parts";
            this.parts_btn4.AccessibleRole = System.Windows.Forms.AccessibleRole.PushButton;
            this.parts_btn4.Cursor = System.Windows.Forms.Cursors.Hand;
            this.parts_btn4.ForeColor = System.Drawing.Color.Black;
            this.parts_btn4.Location = new System.Drawing.Point(213, 127);
            this.parts_btn4.Name = "parts_btn4";
            this.parts_btn4.Size = new System.Drawing.Size(34, 29);
            this.parts_btn4.TabIndex = 4;
            this.parts_btn4.Text = "4";
            this.parts_btn4.UseVisualStyleBackColor = true;
            this.parts_btn4.Click += new System.EventHandler(this.parts_btn4_Click);
            // parts_btn5 //
            this.parts_btn5.AccessibleDescription = "Separate song in 4 parts";
            this.parts_btn5.AccessibleName = "five parts";
            this.parts_btn5.AccessibleRole = System.Windows.Forms.AccessibleRole.PushButton;
            this.parts_btn5.Cursor = System.Windows.Forms.Cursors.Hand;
            this.parts_btn5.ForeColor = System.Drawing.Color.Black;
            this.parts_btn5.Location = new System.Drawing.Point(253, 127);
            this.parts_btn5.Name = "parts_btn5";
            this.parts_btn5.Size = new System.Drawing.Size(34, 29);
            this.parts_btn5.TabIndex = 5;
            this.parts_btn5.Text = "5";
            this.parts_btn5.UseVisualStyleBackColor = true;
            this.parts_btn5.Click += new System.EventHandler(this.parts_btn5_Click);
            // stems4 //
            this.stems4.Location = new System.Drawing.Point(0, 0);
            this.stems4.Name = "stems4";
            this.stems4.Size = new System.Drawing.Size(104, 24);
            this.stems4.TabIndex = 35;
            // stems2 //
            this.stems2.Location = new System.Drawing.Point(0, 0);
            this.stems2.Name = "stems2";
            this.stems2.Size = new System.Drawing.Size(104, 24);
            this.stems2.TabIndex = 36;
            // stems5 //
            this.stems5.Location = new System.Drawing.Point(0, 0);
            this.stems5.Name = "stems5";
            this.stems5.Size = new System.Drawing.Size(104, 24);
            this.stems5.TabIndex = 34;
            // chkRPartOther //
            this.chkRPartOther.AutoSize = true;
            this.chkRPartOther.ForeColor = System.Drawing.Color.Black;
            this.chkRPartOther.Location = new System.Drawing.Point(255, 26);
            this.chkRPartOther.Name = "chkRPartOther";
            this.chkRPartOther.Size = new System.Drawing.Size(52, 17);
            this.chkRPartOther.TabIndex = 5;
            this.chkRPartOther.Text = "Other";
            this.chkRPartOther.UseVisualStyleBackColor = true;
            // chkRPartPiano  //
            this.chkRPartPiano.AutoSize = true;
            this.chkRPartPiano.ForeColor = System.Drawing.Color.Black;
            this.chkRPartPiano.Location = new System.Drawing.Point(197, 26);
            this.chkRPartPiano.Name = "chkRPartPiano";
            this.chkRPartPiano.Size = new System.Drawing.Size(53, 17);
            this.chkRPartPiano.TabIndex = 4;
            this.chkRPartPiano.Text = "Piano";
            this.chkRPartPiano.UseVisualStyleBackColor = true;
            // chkRPartDrums //
            this.chkRPartDrums.AutoSize = true;
            this.chkRPartDrums.ForeColor = System.Drawing.Color.Black;
            this.chkRPartDrums.Location = new System.Drawing.Point(136, 26);
            this.chkRPartDrums.Name = "chkRPartDrums";
            this.chkRPartDrums.Size = new System.Drawing.Size(56, 17);
            this.chkRPartDrums.TabIndex = 3;
            this.chkRPartDrums.Text = "Drums";
            this.chkRPartDrums.UseVisualStyleBackColor = true;
```

```
// chkRPartBass //
this.chkRPartBass.AutoSize = true;
this.chkRPartBass.ForeColor = System.Drawing.Color.Black;
this.chkRPartBass.Location = new System.Drawing.Point(82, 26);
this.chkRPartBass.Name = "chkRPartBass";
this.chkRPartBass.Size = new System.Drawing.Size(49, 17);
this.chkRPartBass.TabIndex = 2;
this.chkRPartBass.Text = "Bass";
this.chkRPartBass.UseVisualStyleBackColor = true;
// chkRPartVocal //
this.chkRPartVocal.AutoSize = true;
this.chkRPartVocal.ForeColor = System.Drawing.Color.Black;
this.chkRPartVocal.Location = new System.Drawing.Point(24, 26);
this.chkRPartVocal.Name = "chkRPartVocal";
this.chkRPartVocal.Size = new System.Drawing.Size(53, 17);
this.chkRPartVocal.TabIndex = 1;
this.chkRPartVocal.Text = "Vocal";
this.chkRPartVocal.UseVisualStyleBackColor = true;
// Form1 //
this.AllowDrop = true;
this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.BackColor = System.Drawing.Color.White;
this.ClientSize = new System.Drawing.Size(522, 628);
this.Controls.Add(this.menuStrip1);
this.Controls.Add(this.pnlRecombine);
this.Controls.Add(this.pnlMain);
this.Controls.Add(this.parts_label);
this.Controls.Add(this.parts_btn5);
this.Controls.Add(this.parts_btn4);
this.Controls.Add(this.parts_btn2);
this.Controls.Add(this.lblPartsTitle);
this.Controls.Add(this.lblSlogan2);
this.Controls.Add(this.lblSlogan1);
this.Controls.Add(this.pictureBox1);
this.Controls.Add(this.stems5);
this.Controls.Add(this.stems4);
this.Controls.Add(this.stems2);
this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedSingle;
this.Icon = ((System.Drawing.Icon)(resources.GetObject("$this.Icon")));
this.Margin = new System.Windows.Forms.Padding(2);
this.MaximizeBox = false;
this.Name = "Form1";
this.SizeGripStyle = System.Windows.Forms.SizeGripStyle.Hide;
this.Text = "MusicSeparator";
this.Load += new System.EventHandler(this.Form1_Load);
this.Shown += new System.EventHandler(this.Form1_Shown);
this.pnlMain.ResumeLayout(false);
this.pnlMain.PerformLayout();
((System.ComponentModel.ISupportInitialize)(this.duration)).EndInit();
this.pnlRecombine.ResumeLayout(false);
this.pnlRecombine.PerformLayout();
((System.ComponentModel.ISupportInitialize)(this.pictureBox1)).EndInit();
this.ResumeLayout(false);
this.PerformLayout();   }
```