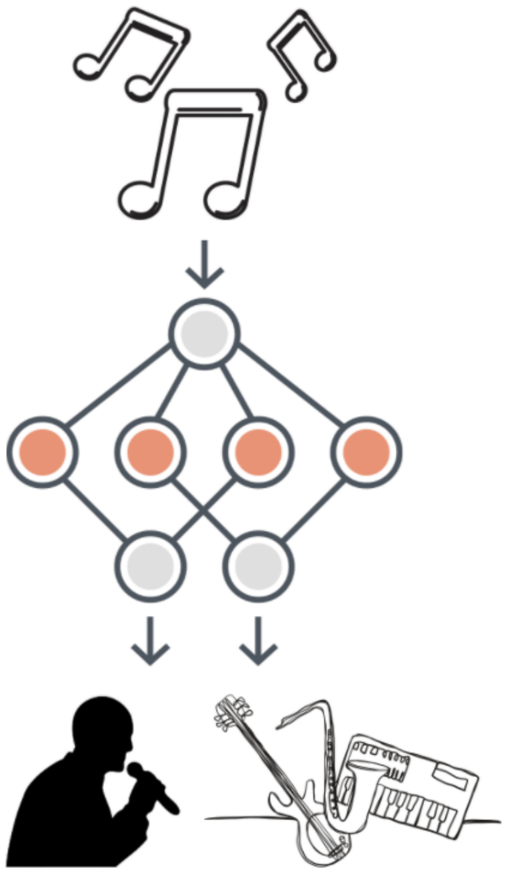
Name: Mohammadreza Pourtorkan

Student ID: 13605486

Supervisor: Roman Kontchakov

Audio Source Separation Using Deep Learning



BSc Computing Project Report, Birkbeck College, University of London, 2022.

Abstract

We propose a deep learning approach to decompose music into its vocals and accompaniments. The audio source separation problem is transformed into an image segmentation problem, and a **U-Net** architecture will be used to identify and separate the component of a music file based on its spectrogram (an image representation of a signal) as the input of the model.

A weighted loss function will be used to balance and change the magnitude of each component in this architecture. The model can not only break down the music into vocals and accompaniments, but it can accomplish multi-source separation to decompose the accompaniments into their sources by modifying the weights of the loss function and the architecture of the model's output layer.

We will investigate and evaluate various models to develop a model architecture that is both accurate and performant.

Table of contents

Chapter 1: Introduction

1.1 Methodology, Approach, Aim and Objectives of This Project

In this project, the U-Net architecture mentioned in [12] will be the main tool in this research and several models will be trained based on that architecture using the musdb18, which is a publicly available dataset.

The project aims to design a source separation deep learning model that takes a song as input and outputs the vocals and accompaniments of the input song. The model should be performant and have acceptable music evaluation metrics. An intuitive UI will also be available for users to interact with the model, upload an input song, and download the separated sources.

Because of the research needed in signal processing, source separation and deep learning areas and the model training time constraints (which could take a few days), having working software at the end of short sprints was not always possible. Therefore, a waterfall software development approach is used instead of using an agile methodology (which was proposed in the proposal). This approach divided the development cycle into several stages, which allowed for better time management and development.

1.2 Blind Source Separation (BSS)

Overview

BSS is a set of algorithms to decompose a mixed-signal into its components having no or very little prior information about the mixing process.

One of the famous examples of this area is The Cocktail Party Problem, where different signals from different sources arrive at the listener. In this situation, the listener’s brain has to estimate and determine the background sounds from the person’s voice he/she is talking.

Applications of BSS

BSS is a popular research topic, and there are several use cases in many domains for an accurate BSS system. Some of the areas mentioned in [1] are:

a) **Biomedical applications** (e.g. electroencephalography (EEG)). In EEG, special sensors record the electrical signals generated by the brain to find any abnormalities in the brain’s activity. BSS algorithms can help remove the artefacts in the recorded signals, which are usually produced by muscular motions [2].

b) **Audio Source Separation** applications include

   i) ***Noise Reduction****:* One can use the source separation algorithm to detect and distinguish background noises from foreground sounds. Therefore, the noise can be silenced, resulting in a clean foreground sound without any noise. Examples of using source separation for noise reduction can be found in [3].

   ii) ***Music Remixing***: Remixers need the individual components of the music (such as the vocals and instruments) to remix a song. The music files' components are usually unavailable, and only the mixture is released by the artists, which makes the remixing options limited. An Audio source separation solution can estimate the individual sources of the music file components and allow the remixers to use and modify the original music mixture. [4] is an example of a source separation approach for remixing music files on the web.

c) **Image Processing**, where an image composed of different photos is the input of the BSS algorithm, and the output is the distinct individual photos.

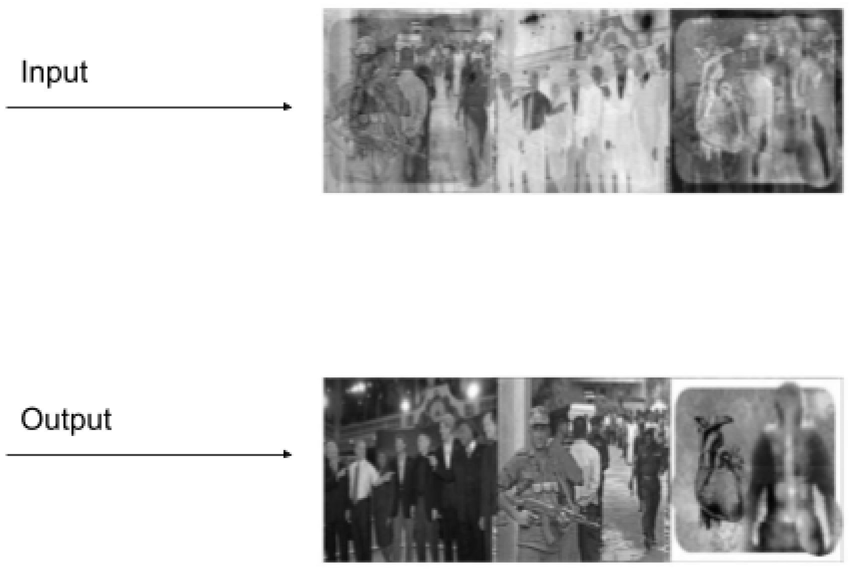


Figure 1. Unmixing images using BSS [5]

1.3 Audio Source Separation and Music Information Retrieval

Audio Source Separation is a part of the **Music Information Retrieval (MIR)** research field, which focuses on retrieving information and extracting features from music pieces. Other research in MIR concentrates on developing music classification systems, music Identification software (e.g. Shazam), music generation, and more.

Audio source separation is challenging since music is typically created by combining various individual components using an unknown mixing procedure. The rest of this section is about the basic mathematical explanations of blind source separation and its difficulties using classical approaches.

Considering $\boldsymbol{M}$ number of microphones in the room and ***N*** number of independent sound-generating sources, we can show the source vector as

$$S = [s_1, s_2, \dots, s_n]^T \quad n = 1, 2, \dots, N \tag{1}$$

and the recorded mixtures vector as

$$X = [x_1, x_2, \dots, x_m]^T \quad m = 1, 2, \dots, M \tag{2}$$

Furthermore, each recorded mixture is calculated as follows:

$$x_i(t) = \sum_{j=0}^{N} (\alpha_{ij} \cdot s_j(t)) + \varepsilon(t) \quad i = 1, 2, \dots, M \tag{3}$$.

Where t is time and ε is any background noise that the microphone might record at a given time t.

*"α which is a high-order **Finite Impulse Response (FIR**) filter that models the room transfer functions between i th sensor and the j th source"* [6] can be represented as

$$\begin{bmatrix} \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1N} \\ \alpha_{21} & \alpha_{22} & \cdots & \alpha_{2N} \\ \vdots & \vdots & & \vdots \\ \alpha_{M1} & \alpha_{M2} & \cdots & \alpha_{MN} \end{bmatrix} \tag{4}$$

The following figure shows the mixing procedure using sources s, mixing functions α and mixed outputs x:

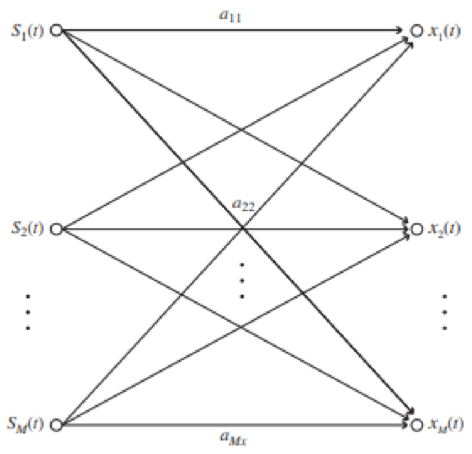


Figure 2. Linear Mixing Process [7]

From equation (3), one can see that each recorded output is a mixture of all the sources modified by the Finite impulse response (FIR) filter plus any background noise. Assuming there are no background noises, the mixing process is performed using the following matrix multiplication:

$$X(t) = A \,.\, S(t) \qquad (5)$$

The problem with equation (5) is that there is no or little information about the matrix A, the mixing matrix applied to the individual sources. Therefore, the BSS method should estimate an unmixing matrix $U_{N\,x\,M}$ to unmix the recorded signals to get an approximate source signal. The unmixing equation is

$$U \,.\, X(t) = \widehat{S}(t) \qquad (6)$$

where $\widehat{S}(t)$ is the estimated source matrix with the shape $N\,x\,1$.

The source signals are not independent in the real world, and one should also consider nonlinear mixtures. Therefore, the mixing equation will be as follow:

$$X(t) = f(S(t)) + N(t) \qquad (7)$$

“where $X(t)$ are M-dimensional observed signal vectors, $S(t)$ are N-dimensional unknown source signal vectors, and $N(t)$ are M-dimensional additive noise vectors that are independent of the source signals. $f : R^{N} \rightarrow R^{M}$ is the unknown reversible nonlinear mixing function.” [7]

In this case, a nonlinear unmixing function should be estimated to calculate the sources by using:

$$\hat{S}(t) = u(X(t)) \qquad (8)$$

where $u: R^{M} \rightarrow R^{N}$ is the nonlinear unmixing function and $\hat{S}(t)$ is the N dimensional output.

As a general rule, based on the number of sources ***N*** and number of microphones ***M***, the following approaches should be used:

a) If ***N* < *M***, the equations of the mixing process are underdetermined, and linear approaches should be used.

b) Systems are usually determined, which means for most systems, the number of microphones recording the sources and the number of sources are the same thus ***N=M*** . In that case, linear filtering approaches can be used.

c) If ***N* > *M***, One should use the nonlinear BSS approaches as the mixing process is overdetermined.

The following figure from [8] is a guide for using appropriate BSS methods:

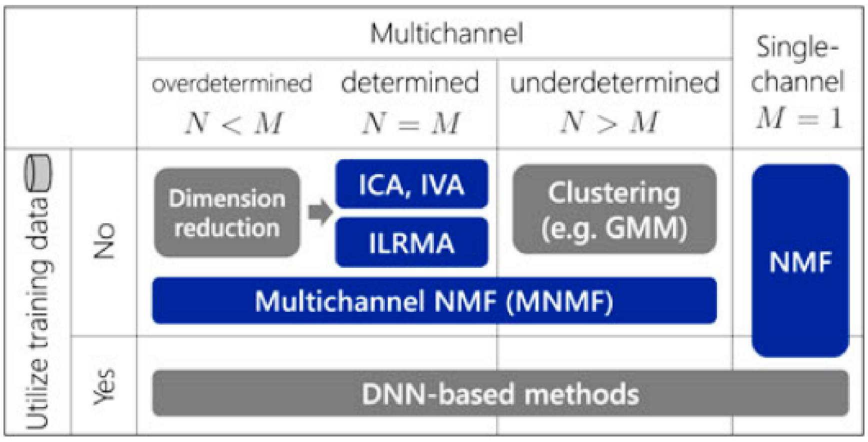


Figure 3. Summary of BSS Approaches [8]

There are many classical and modern approaches to find the unmixing matrix which will be mentioned in the next section.

1.4 Related Works and Approaches

In this section, Some related works and approaches to performing BSS are explained.

Independent Component Analysis

This method extends Principal Component Analysis (PCA) and assumes sources are independent of each other, and the source data has a non-gaussian distribution. ICA method tries to solve equation (6) and find an unmixing matrix $U \simeq A^{-1}$ to estimate the sources. The mathematical explanations of this method are out of the scope of this project, but a thorough explanation can be found at [9]

Non-Negative Matrix Factorisation (NMF)

This method performs the estimation on a non-negative matrix $V_{M\,x\,N}$ and tries to find a combination of two matrices $W_{M\,x\,R}$ and $H_{R\,x\,N}$ where the columns of W (value of R) act as the number of source components and columns of H as activations of each source in the mixture. The following equation is the basis of the NMF:

$$V = WH \qquad (9)$$

For Audio Source Separation, the model's input (matrix V) is the magnitude spectrogram generated by applying the Short-Time Fourier Transform (STFT) on the audio file in the time domain to get a representation of the audio in the time-frequency domain. The values of W and H are iteratively improved to reduce a loss value calculated by a loss function, leading to a more accurate estimation of each source.

The values of W and H are initialised randomly. After minimising the loss value through some iteration, the Inverse Short-Time Fourier Transform (ISTFT) is performed on the result of WH plus the phase information obtained from the STFT of the original signal. This will transform the music from the time-frequency domain to the time domain representation and reconstruct the estimated source.

Machine learning

Machine learning is a field that discusses the approaches machine can take to learn, predict certain results and reach conclusions without being explicitly programmed. Deep learning is a specific part of this field that focuses on how humans learn. Deep learning approaches can be used to imitate the human brain in machines using a network of computational units called artificial neural networks.

A neural network has a set of input, hidden and output nodes and each connection between these nodes carries a certain weight. These weights can be adjusted to minimise the output error (calculated by a loss function) a network produces.

Advances in deep learning and neural networks allowed many researchers to try new approaches in this field of study as neural networks perform well in finding patterns. Some of the best audio source separations systems, such as Spleeter by Deezer and Demucs by Facebook, use deep learning to perform the separations.

Many of these systems use a neural network architecture called U-Nets, which has a contracting and an expansive path. This architecture was initially used for biomedical image segmentation tasks [10], and because of its efficiency and success, it has been used in many different areas.

The input of this network architecture is usually an image representation of the audio file in the time-frequency domain (spectrogram), and the output is usually either a spectrogram mask [11] or spectrogram of each separated source [12]. [13] uses the audio waveform as the input instead of a spectrogram and also outputs the separated wave audios.

1.5 Report Structure

In the following chapters, we dive into the design and structure of the models and try to choose the best preprocessing, training and post-processing approaches that lead to choosing the best model design.

In Chapter 2, an overview of the proposed approach will be discussed and the steps of the process will be explained. Flowcharts and figures will assist the reader in understanding the material.

Chapter 3 is about the implementation and design of the materials in Chapter 2.

In Chapter 4, multiple models that have a slight difference will be trained. The models' accuracies and performances will be evaluated on the test dataset, and the music evaluation metrics of each of the models will be compared.

We conclude the overall Audio Source Separation approach and provide general feedback, improvements and future work that should be undertaken to get a better result in Chapter 5.

Chapter 6 is where we discuss the problems faced during the development, the challenging pieces of work and how we overcome those challenges and the quality of the work.

Chapter 2: Proposed Approach Overview

2.1 What is a U-Net?

U-Net Overview

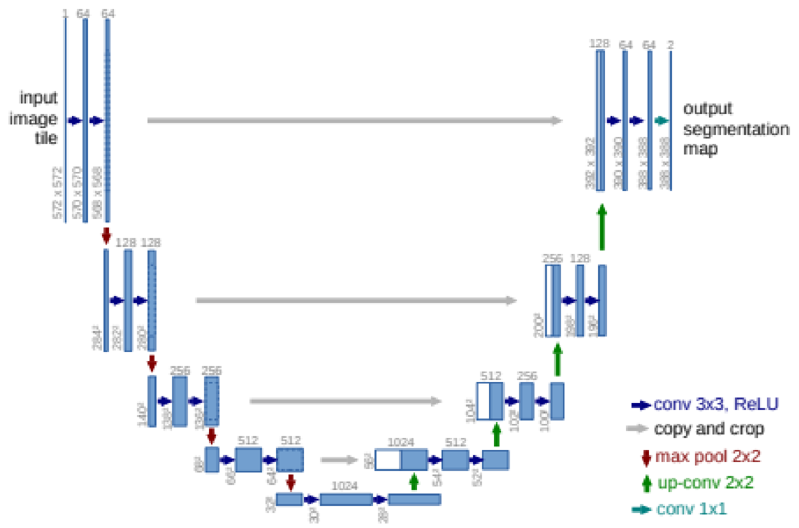


Figure 4. The original U-Net Architecture mentioned in [10]

As mentioned in Section 1.4, U-Net is an appropriate deep learning architecture used by major organisations such as Deezer and Facebook to build the source separation models. The name of the architecture comes from the fact that the network looks like a U (Figure 4).

U-Net is often used for image segmentation, a computer vision task to divide the image into multiple regions by giving each pixel a label; thus, the output is the same image as the input with each pixel classified in a particular class. Pixels that have the same label should have specific characteristics in common.

One of these system applications is analysing and detecting the object in the environment around the self-driving cars. In this system, the cameras constantly take pictures of the environment. The classification system then recognises and classifies each object around the car into different categories like pedestrians, trees, roads, cars, and more.

The original proposed U-Net architecture looks like Figure 4.

U-Net Architecture.

The main feature of this architecture is having a contracting and an expansive path and the skip connections that connect these paths.

Contracting or downsampling path shrinks the image and increases the number of channels while performing convolutional operations to extract essential features of the image. The intuition behind having this path is to have more information per pixel as the number of channels increases while the size of the image decreases.

For example, in Figure 4, the image shape is initially 572x572x1, and it becomes 32x32x1024 at the bottom of the network. This change means that initially, the information is more spread out and each pixel has only one feature channel. However, the information becomes more concentrated as the image size shrinks and the channels increase (1024 feature channels per pixel at the bottom of U-Net). Therefore, the contracting path tries to **increase** the information about **"What"** features are in the image while **reducing "Where"** this information is.

The expansive or upsampling path usually upsamples the image to its original shape (but the shape can also be different depending on the problem) while reducing the number of channels. The upsampling path also performs transposed convolutional operations on the image, which has a trainable kernel and maps each pixel to several pixels while reducing the number of channels. Sometimes, an upsampling layer can be used instead, which does not have a trainable kernel and repeats the rows and columns to increase the input size.

Skip connections between the contracting and expansive layers transfer the extracted features' information to the upsampling layers. These connections localise the information in each layer of the upsampling path, leading to a segmented image as the output containing the extracted features.

In summary, U-Nets components are

a) Contracting convolutional path: Layers in these paths extract specific features from the image while shrinking it in size and increasing the number of channels.

b) Expansive convolutional path: This path up samples the image while reducing the number of channels to reconstruct the original image.

c) Skip connections transfer the extracted information from the contracting path while recreating the image.

U-Nets and CNNs

CNN layers and layers in the contracting path of the U-Nets are mainly convolutional, and max-pooling layers are used to reduce the image size while increasing the number of channels per pixel. This architecture leads to extracting features from the images, like finding the borders of an image or detecting a similar pattern.

The main difference between a U-Net architecture and a CNN model is that the CNN model takes an image as input and produces a class label for the entire image as the output. On the other hand, U-Nets receive an image and produce an image with the same shape with each pixel given a class label. Hence, they perform similar operations, but because U-Nets have an expansive path and skip connections, they are used for segmenting the same input image. In contrast, CNN is best used to classify the entire image.

U-Nets for Audio Source Separation

Because U-Nets are mainly for image segmentation tasks, one can turn the Audio Source Separation problem into an image segmentation one by using the audio representation in the time-frequency domain (spectrograms).

What is a spectrogram and how is it calculated?

Spectrograms are a way to visualise the loudness of different frequencies across time. The spectrograms are calculated using the **Short-Time Fourier Transform (STFT)** of the signal, a *Fourier-related transform used to determine the sinusoidal frequency and phase content of local sections of a signal as it changes over time* [14].

What does STFT do?

The STFT puts the signal frames or samples in different frequency bins by comparing them to pure tones and applying a window function on several frames. **Pure tones** *are the signals with sinusoidal waveforms* [15]. Each frame in the frequency bin is a complex number that consists of magnitude as the real part and phase as the imaginary part. **Magnitude** is the strength of the signal in a given frequency, and **phase** is the shift in time between two sinusoids.

The parameters passed to STFT are frame size (number of frames in the chunk of signal we are applying STFT on), window size (the length of the windowing function in frames, which is usually equal to frame size) and hop size (how many frames should the window shift).

The higher the frame size, the higher the frequency resolution, and the lower the time resolution will be and vice versa. With a higher frame size, the STFT is performed on more frames; thus, the number of different frequencies in a larger number of frames will be more. This leads to higher frequency resolution (more frequency bins to put the frames into) but lowers the time resolution.

The result of STFT, therefore, depends on the parameters passed to it and different parameters can lead to different results. STFT produces a matrix with the shape of $number\ of\ frequency\ bins\ X\ number\ of\ frames$.

Where the first dimension is

$$number\ of\ frequency\ bins\ =\ \lfloor \frac{frame\ size}{2} \rfloor\ +\ 1 \qquad (10)$$

and the second dimension is

$$number\ of\ frames\ =\ \lceil \frac{number\ of\ samples\ in\ signal}{hop\ size} \rceil \qquad (11)$$

In (10), $\lfloor x \rfloor$ is the flooring function (e.g $\lfloor 2.4 \rfloor\ =\ 2.0$) and in (11), $\lceil x \rceil$ is the maths ceiling function which returns the smallest integer which is larger than x. (e.g $\lceil 2.4 \rceil\ =\ 3$)

Spectrogram from STFT

To create spectrograms, the squared magnitude of the STFT result is usually used, and the phase information is ignored. The phase information of each frame is needed to transform the signal more accurately from the time-frequency domain to the time domain using Inverse STFT (ISTFT). Therefore, STFT transforms a signal from a time-domain representation to a time-frequency domain representation.

An example of this shift is:

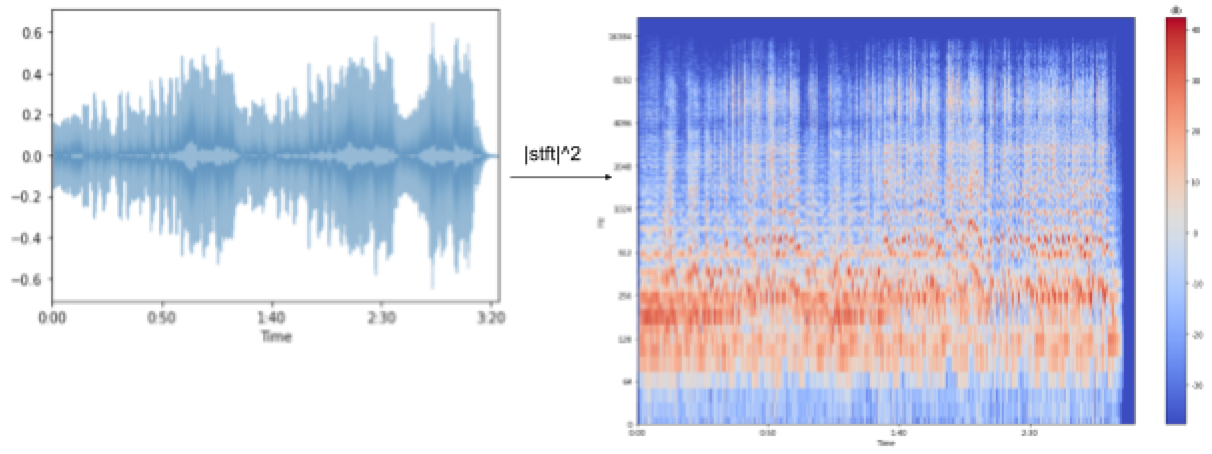


Figure 5. Spectrogram of a time-domain signal using STFT squared magnitude

The redder the colour is on the spectrogram means that the frequency at that given time has more strength.

In general, the steps for creating the spectrograms are:

a) Receive the signal

b) Apply STFT to the signal with appropriate parameters to calculate the complex-valued matrix

c) Calculate $|STFT|^2$ to generate the power spectrogram of the signal (which shows the relative power of each frequency in the signal at any given time)

d) Convert power spectrogram to decibel units. This step is performed because humans do not perceive sounds linearly but the sound is perceived logarithmically (more details in [18]).

   Performing this step produces the spectrogram on a logarithmic scale and the colour intensity of the spectrograms shows how humans actually perceive the audio and the frequencies.

e) Visualise the spectrogram.

Spectrograms as the input of U-Nets in BSS

Spectrograms can be treated as images while giving all the information related to the frequency and loudness of the signal at any given point in time. Therefore, spectrograms can be used as U-Nets' inputs to find similar patterns and regions in the audio. These segments in the spectrogram are the estimates of individual sources that constructed the audio signal, and they can be extracted from the input spectrogram.

The estimated source spectrograms will be extracted from the mixed-signal without having any prior information about the mixing process, which is the definition of BSS.

2.2 Proposed BSS Approach

Now that the reason for using U-Nets and spectrograms of music as the inputs in an Audio Source Separation system is explained, let us have a look at the proposed structure of this report and the proper preprocessing steps.

U-Net Structure

The U-Net used in this system is based on the structure mentioned in [12]. This architecture is based on receiving a spectrogram with shape 1025 x 173 x 1 and producing a spectrogram with shape 1025 x 173 x k where k is the number of sources the system is trying to separate (e.g. in the case of separating the vocals and accompaniments, k will be 2).

The reasoning behind the input spectrogram shape is that the song is divided into two-second-long segments. With a sampling rate of 44100 Hz, there will be a total of 88200 frames in two-second segments. The values used for STFT operations are usually 512 frames for hop size and 2048 frames for window and frame size. Therefore, according to equation (10), we get 1025 for the first dimension and 173 for the second dimension as the spectrogram size of a two-second-long segment.

The reason behind dividing the song into smaller segments and performing STFT on these segments rather than the whole audio is with a smaller number of frames, the frequency does not shift significantly [16], and the change of the signal can be detected more accurately by losing some degree of frequency resolution while gaining a better time resolution [17]. Therefore, a two-second-long segment as the input of the network is a good choice.

The system has a repeating pattern in the contracting and expansive path, which is as follows:

1. **Contracting Path**: In this path, the network has a convolutional block. This block consists of two convolutional layers that perform convolution operations on the input while increasing the number of channels (leading to more information per pixel) and a max-pooling layer that keeps the number of channels while halving the size of the spectrogram.

   This block repeats four times, and at the end of the path, it is followed by two extra convolutional layers, which increase the number of channels to 512.

   Each convolutional layer has a ReLU activation function (which adds non-linearity to the data and helps the networks learn more complex data). Then, a batch normalisation layer normalises the layer's output before passing it to the next layer as input. Normalised data helps the model converge more quickly, leading to a better and faster network.

a) **Expansive Path**: This path also has a repeated structure of an upsampling block which doubles the size while keeping the number of channels (because of concatenation layers). This block is followed by a deconvolutional block that preserves the size and halves the number of channels.

   The upsampling block consists of:

   I. An upsampling layer with a ReLU activation function that halves the number of channels and doubles the spectrogram size.

   II. A batch normalisation layer

   III. A drop out layer that turns off random neurons to avoid overfitting and helps the network learn the patterns and not only the data.

   IV. A concatenation layer that connects this block to the layer in the contracting path that has a similar spectrogram size in the output. This layer adds the number of channels while keeping the size. It transforms the information from a contracting path to an expansive path.

The deconvolutional block is made of two deconvolutional layers each of them followed by:

I. ReLU activation layer

II. Batch normalisation layer

The final layer in this path is a 1 x 1 x k convolutional layer with a ReLU activation function. This layer maps a single pixel and all the channels associated with it to a single pixel with k number of channels and each channel corresponds to one of the estimated sources in the music.

It is easier to think of this layer as a fully connected neural network that receives the channels of the pixel as input and has k neurons. Each neuron produces a single number for the pixel by applying weights and biases to each channel, performs a ReLU function on that number and produces a value for that pixel. This layer produces k number of spectrograms, each with the same spatial dimension as the input.

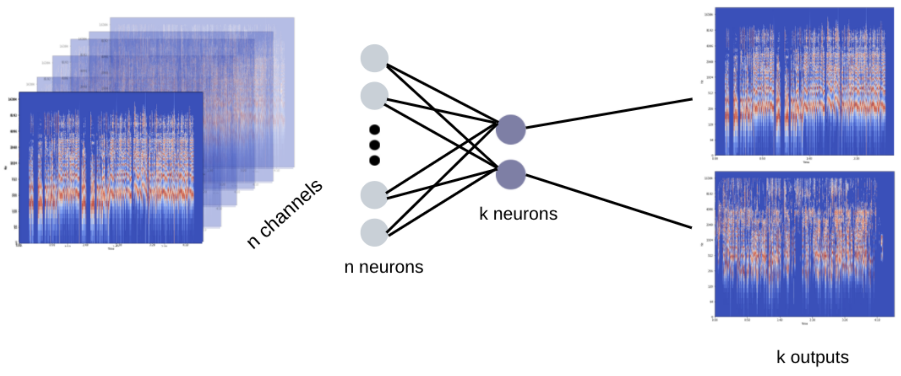


Figure 6. 1x1xk convolutional layer with k = 2

The diagram above illustrates the operation of the 1 x 1 x k layer. Each channel of a pixel in the n channel image is fed into the network. The channels are processed through k neurons, and once all pixels' channels have been passed through, k spectrograms with the exact height and width are generated. Therefore, the network's output is a k-channel picture, with each channel representing an estimated source.

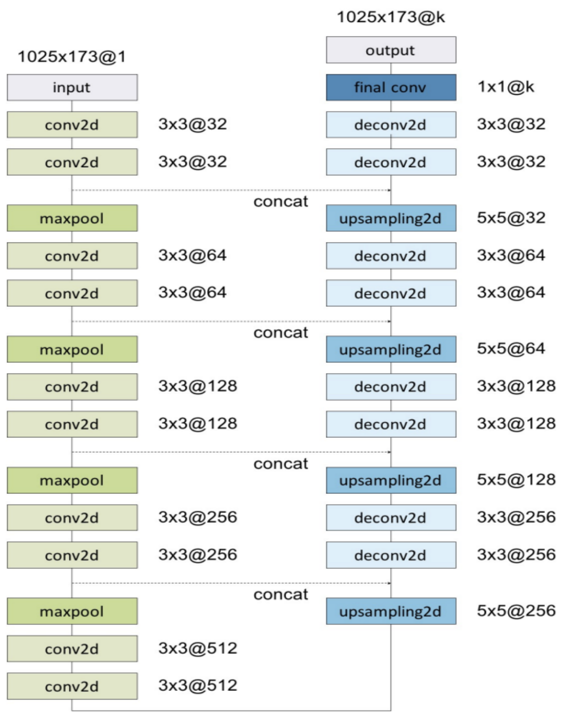


Figure 7. Overall U-Net architecture [12]

The Figure above shows the structure of the proposed network.

Dataset Details

To train a deep learning model, a dataset with a sufficient amount of data is required. For Music Source Separation tasks, musdb18 is usually used, which consists of o150 full-length songs (around ten hours of music). One hundred music recordings are dedicated to training and fifty music to testing; all sampled at 44100 Hz (it means there are 44100 frames per second in the song).

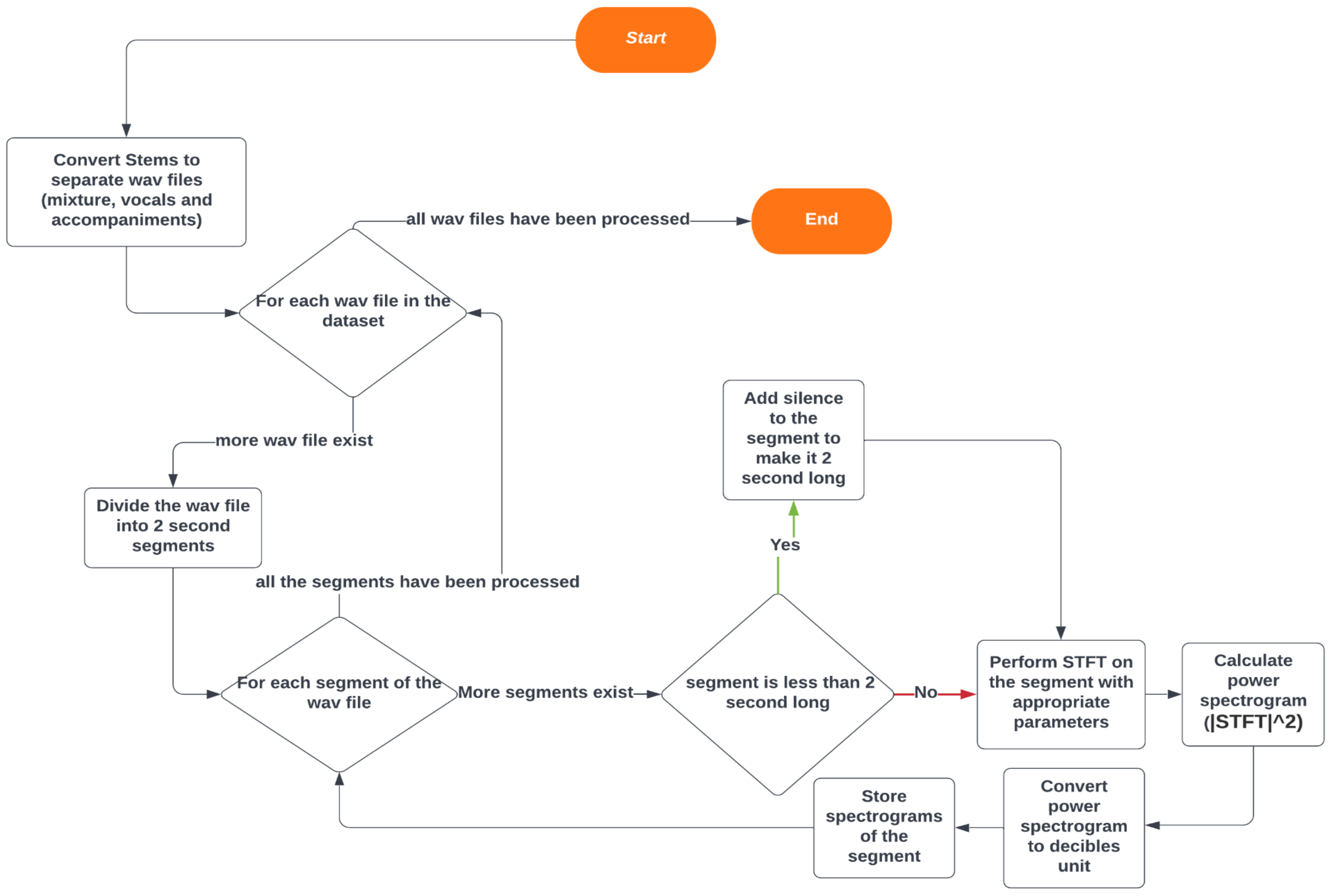
Each piece of music is stored as a stem file (which is a file format to group multiple audio sources) and is made of five different sources, which are:

a) Mixture
b) Vocals
c) Bass
d) Drums
e) Other Accompaniments

In this project, only mixture, vocals and accompaniments (mixture of bass, drums and others) are needed.

Preprocessing

Data preprocessing is a process that converts raw data to a format that is usable by the system. In this project, the deep learning model expects a spectrogram, and the data in the dataset is not in the spectrogram format with the specified size; therefore,



Start
Convert Stems to separate wav files (mixture, vocals and accompaniments)
all wav files have been processed
End
For each wav file in the dataset
more wav file exist
Divide the wav file into 2 second segments
all the segments have been processed
For each segment of the wav file
More segments exist
segment is less than 2 second long
Yes
Add silence to the segment to make it 2 second long
No
Perform STFT on the segment with appropriate parameters
Calculate power spectrogram (|STFT|^2)
Convert power spectrogram to decibles unit
Store spectrograms of the segment

the dataset needs to be preprocessed.

Figure 8. Preprocessing flowchart

The flow chart above lays out all the preprocessing steps that the pieces of music in the musdb dataset should go through.

Converting Stems to Separate Wav Files

As stems are a group of separate audio sources, the first step is to unpack those sources to separate wave files which are mixture.wav, vocals.wav and accompaniment.wav (which is a mixture of bass, drums and other accompaniments) all with 44100 Hz sample rate.

Wave files are the time-domain representation of audio signals and are used to generate spectrograms.

Generating Spectrograms from Wave Files

The input of the proposed network is a 1025 x 173 x 1 spectrogram which is the spectrogram size of two-second-long segments. The spectrograms are based on the result of STFT, and to get the desired shape, appropriate parameters should be passed to the STFT function.

Therefore, the first step to generate the spectrogram is to divide the audio data into two-second long segments. The audio is rarely divisible into two-second segments, and the last segment may have a shorter length. Hence, to make all the segments equally long, data of the shorter segment should be padded with silence (i.e. zeros) to become two seconds long.

The steps of calculating the spectrograms are as mentioned in Section 2.1, but the steps are performed on each segment of the song.

After these calculations, the spectrogram of the mixture segments is stored, and they will be used as the input of the model later.

Input Shape Issues and Proposed Solution

The issue with the mentioned shape of the spectrogram as the input of the U-Net is that the same shape cannot be reconstructed in the upsampling layer. The reason is that max-pooling layers will divide the size in half, and if the dimensions are odd, the shapes will be floored to the nearest even integer.

Block Name	***Input Shape***	***Output Shape***
First convolutional block	*1025 x 173 x 1*	*512 x 86 x 32*
Second convolutional block	*512 x 86 x 32*	*256 x 43 x 64*
Third convolutional block	*256 x 43 x 64*	***128 x 21 x 128***
Forth convolutional block	*128 x 21 x 128*	*64 x 10 x 256*
Centre Convolutional layers	*64 x 10 x 256*	*64 x 10 x 512*
First Upsampling block	*64 x 10 x 512*	***128 x 20 x 256***

Table 1. Problem with odd input shape in U-Net

According to the table above, initial spectrogram shapes will cause a shape mismatch after the first upsampling block. This is because 1025 and 173 are not divisible by 2, and halving the dimensions by the max-pooling layer will cause an issue. To concatenate the channels and transform information, The shapes of the spectrograms (other than the channels) must match, and any mismatching shapes will cause an error.

To fix this issue, it is always recommended to have input shapes that are divisible by 2^n where n is the number of max-pooling layers. This is because, after n max-pooling and upsampling operations, the shapes will always match at any stage (as max-pooling halves the dimensions while upsampling doubles the dimensions).

In the proposed architecture, the network has four max pooling and four upsampling layers. Hence, the input dimensions must be divisible by 2^4. The nearest number greater than 1025, which is divisible by 16, is 1040, and for 173 is 176. To achieve these dimensions, the spectrogram must be padded by zeros before going through the network to achieve these dimensions. The output must be cropped to the original dimensions to produce the same spectrograms as the inputs.

Training the Model

After the preprocessing steps, the data is ready to be used for training the U-Net models. The deep learning models expect several parameters in order to find the patterns and perform the desired task which are:

a) **Epochs**: The number of times the neural network will process the entire dataset. The proposed models will be trained for 40 epochs.

b) **Loss Function and Optimizer**: In neural networks, the aim is to minimise the error between the predictions and the actual values. To do this task, the networks use a function to calculate the prediction error and use an optimiser to optimise the parameters (as the name suggests) and reduce the error.

   The proposed loss function of this system is a function that balances the prediction error of each source by using a weighting parameter. The equation of this function for singing voice separation (separating vocals and accompaniments) is

$$Loss(Singing\ Voice) = \alpha * L(actual\ vocal,\ predicted\ vocal) + (1 - \alpha) * L(actual\ acc,\ predicted\ acc) \quad (12)$$

   where α is a weighting parameter between 0 and 1 which balances the loss function (e.g. $\alpha = 0$ only calculates the error of accompaniment prediction while $\alpha = 1$ calculates the vocal prediction error) and L calculates the absolute pixel-wise difference of the spectrograms (pixel-wise $L1$ loss) [12].

   The weighting parameter allows the model to perform multi-source separation and separate the accompaniments. The equation of a multi-source separation loss function would be:

$$\begin{aligned} Loss(Multi\ Source) = \ & \alpha_1 * L(actual\ vocal,\ predicted\ vocal) + \\ & \alpha_2 * L(actual\ drums,\ predicted\ drums) + \\ & \alpha_3 * L(actual\ bass,\ predicted\ bass) + \\ & \alpha_4 * L(actual\ other\ acc,\ predicted\ other\ acc) \quad (13) \end{aligned}$$

   In (13), the sum of α values must be 1.

One of the most used and effective optimisers in deep learning is the Adam optimiser [19] which allows the model to converge faster compare to other optimisers. This optimiser accepts different parameters such as learning rate and weight decay. The learning rate determines the amount the weights will be updated during each training step, and weight decay exponentially decays the weights toward 0 as the training progresses to stop the weights from growing too large. These values are positive and usually less than 1.

The Adam optimiser used for the models has a learning rate of 0.001 for the first 20 epochs, and then it drops to 0.0001 for the subsequent 20 epochs. The reason is that initially, weights are initialised randomly, which is far from optimal values, but as the training progresses, more minor weight changes are desired because the weights become more sophisticated and more accurate by going through the dataset. Weight decay will also be 10^{-6}.

c) **Batch Size**: This parameter determines how many data items the network should process before updating the weights.

Smaller batch size leads to slower, more frequent and less accurate updates as the model is updated based on smaller numbers of data. The batch size should not be too small or too large, and an appropriate value based on the number of data should be used. The batch size of 8 is used for the training in the proposed models (which means 8 segments of spectrogram per batch) but because the total number of two-second long segments of the mixture files may not be divisible by 8, the last batch may have a smaller size.

After setting the mentioned parameters, the model will be trained on the spectrogram of segments and their accuracy will be validated using the test dataset.

Post-Processing

Individual source estimation is usually imprecise, and artefacts are frequently formed. These artefacts can be heard as background noise, lowering the prediction's quality. As a result, a denoising procedure is desired as a post-processing step to reduce the additional noise in the predicted spectrograms.

A windowing function is proposed to denoise the signal by looping through the signal and moving the window by a fixed number of frames determined by hop size. This function expects several parameters which are:

a) Window size determines the length of the window

b) Hop size determines how many frames should the window move

c) Amplitude envelope determines how the signal amplitude is changing over time. The envelope is calculated using the signal's Hilbert transform magnitude.

d) Amplitude statistical value: a statistical metric calculated from the amplitude envelope which can be either mean or median (mean is slower to calculate and it is affected by the outliers therefore median is recommended)

e) Denoising softness: determines how significant should the denoising threshold be ($threshold\ =\ amplitude\ statistical\ value\ /\ softness$). The larger the softness value means the calculated threshold is smaller and therefore signals with lower amplitudes are muted and vice versa.

Based on these values, the following steps will be performed by the function:

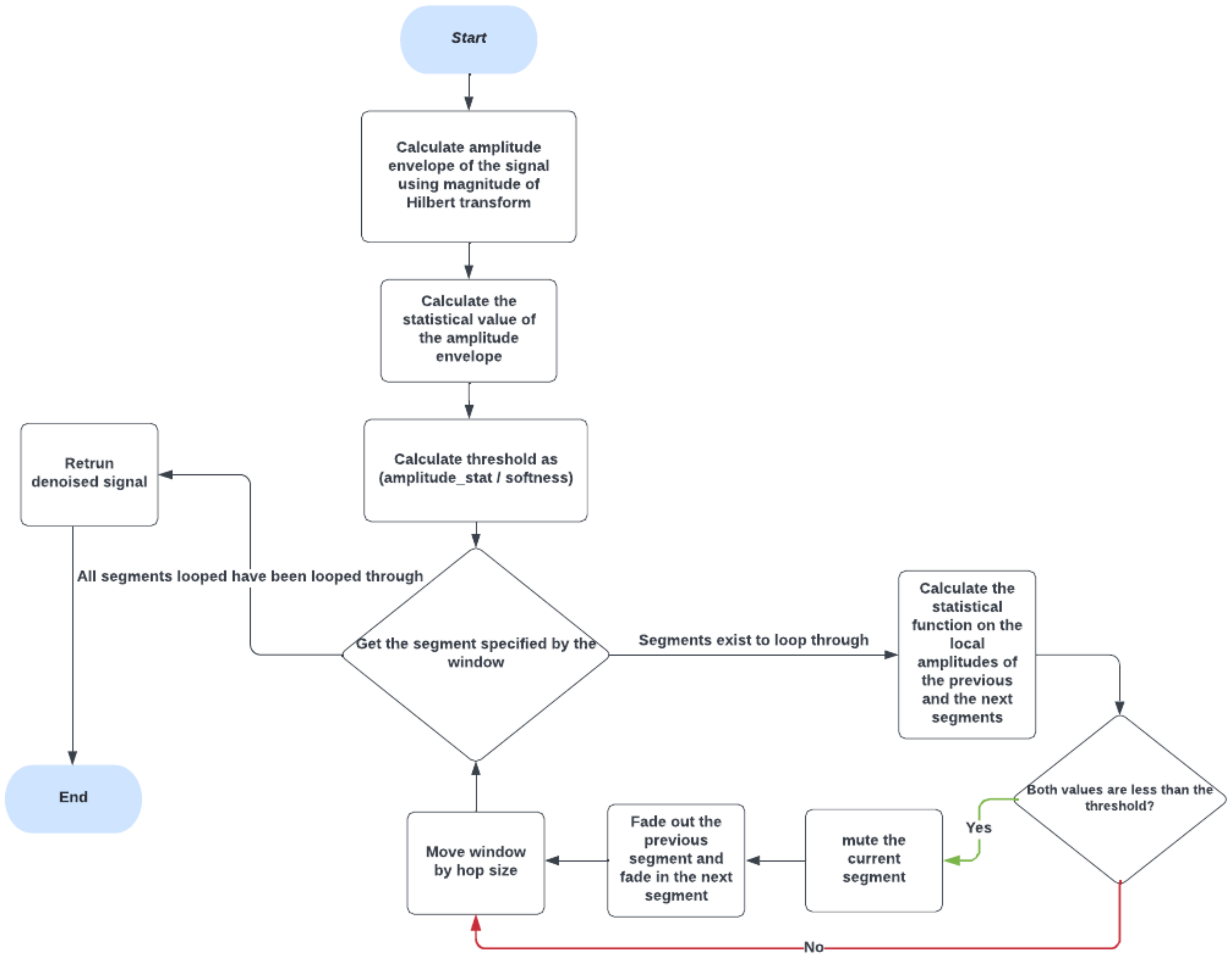

Start
Calculate amplitude envelope of the signal using magnitude of Hilbert transform
Calculate the statistical value of the amplitude envelope
Calculate threshold as (amplitude_stat / softness)
Retrun denoised signal
All segments looped have been looped through
Get the segment specified by the window
Segments exist to loop through
Calculate the statistical function on the local amplitudes of the previous and the next segments
Both values are less than the threshold?
Yes
No
mute the current segment
Fade out the previous segment and fade in the next segment
Move window by hop size
End

Figure 9. The denoising procedure flowchart

With a short window length, if the previous and next segments of a region have statistical values of the local amplitude envelope that are less than the threshold value, the region is most likely a noise artefact. This is because, in the majority of music recordings, each source continuously generates a signal within the same amplitude range, and the overall amplitude of any region of the signal should not be far from a threshold value calculated on the signal's amplitude envelope unless the region is silent or most likely a noise artefact.

This function is mainly effective at removing noises that occur in regions where the estimated source is silent, but it is less effective at removing artefacts that are combined in with the source's sound (for instance, noises that exist concurrently with the singer's vocal sound).

The figure below shows an example of how this function works.

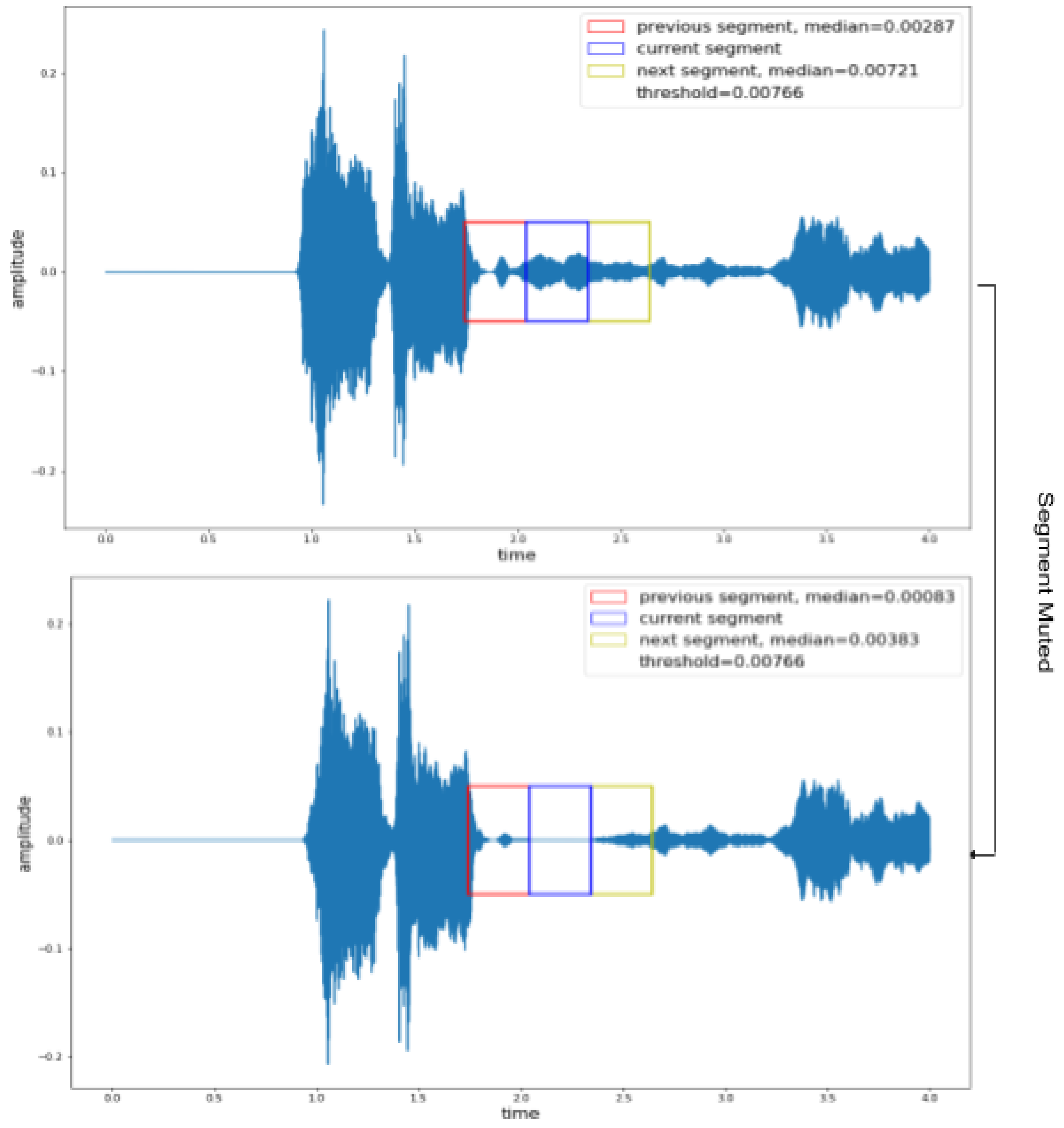


Figure 10. Signal before and after denoising

The current segment is muted in the above figure because the local amplitude medians of the preceding and following segments are both less than the threshold amplitude median, implying that the examined segment is noise. The preceding segment fades out and the subsequent segment fades in. Some of the faded noises that remain will be removed as well due to segment overlap caused by the hop size being smaller than the window size. As a result, the fading effects will end up making the denoising more realistic and the windowing function eliminates some of the artefacts from the signal.

The signal before and after performing the denoising function can be seen below:

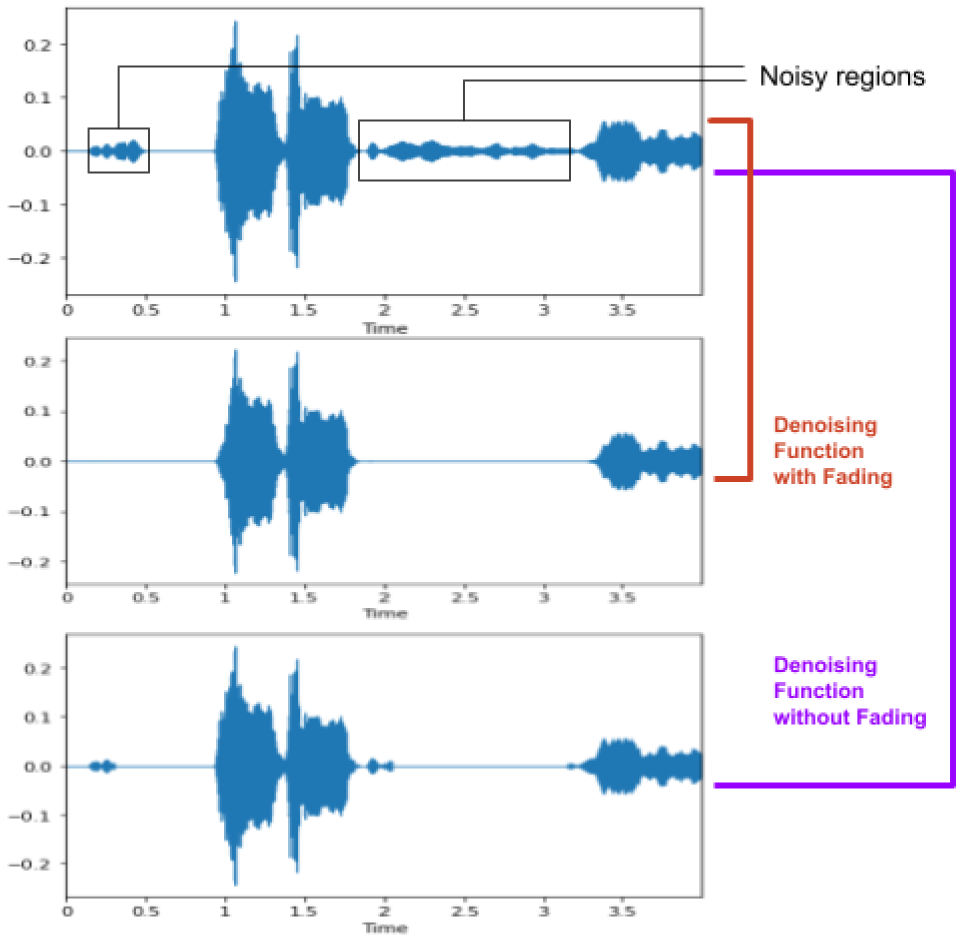


Figure 11. Denoising with and without segment fading

Although this function is not perfect, with the appropriate window size and softness parameter values, it is capable of removing some noise artefacts from estimated signals.

A low pass and high pass filter can also be used to perform the denoising function. The low pass filter allows only frequencies below a frequency threshold, while the high pass filter allows frequencies above the threshold. These filters are typically applied in the time-frequency domain and are capable of removing specific frequencies from the signal.

There is no additional post-processing required on the estimated signals other than the ones mentioned previously, which attempt to improve the signal's quality.

Chapter 3: Implementation

3.1 Tools and Technologies

Programming Language

Python was used to develop all parts of the proposed system. It has a simple syntax and great community support that accelerates the development. There are also many libraries available to use for developing deep learning applications.

Development Environment

Deep learning models require a large number of mathematical operations, such as tensor multiplication, to be trained. While CPUs are adequate for simple programming tasks such as operating system and spreadsheet management, GPUs are recommended for performing the operations due to their ability to perform parallel computations more efficiently with the help of Compute Unified Device Architecture (CUDA), thereby speeding up training.

Google Colab is a cloud-based Jupyter notebook environment that provides free GPUs for deep learning research. Additionally, Google Drive is easily integrated with Colab, which simplifies the process of storing and retrieving datasets and models.

Libraries

Multiple libraries were used in the project development, including (but not limited to) the following:

a) **TensorFlow** is a Google-developed framework for deep learning.

b) **Librosa** is a dedicated Python library for signal and audio processing.

c) **NumPy** is a Python library for performing operations on multidimensional arrays.

d) **SciPy** is a package for performing scientific and technical operations.

e) **Matplotlib** which is a data visualisation and plotting library.

3.2 System Design

Classes

System followed the OOP paradigm and the functionality of the system was divided between multiple classes. The class diagram below shows the classes in the system.

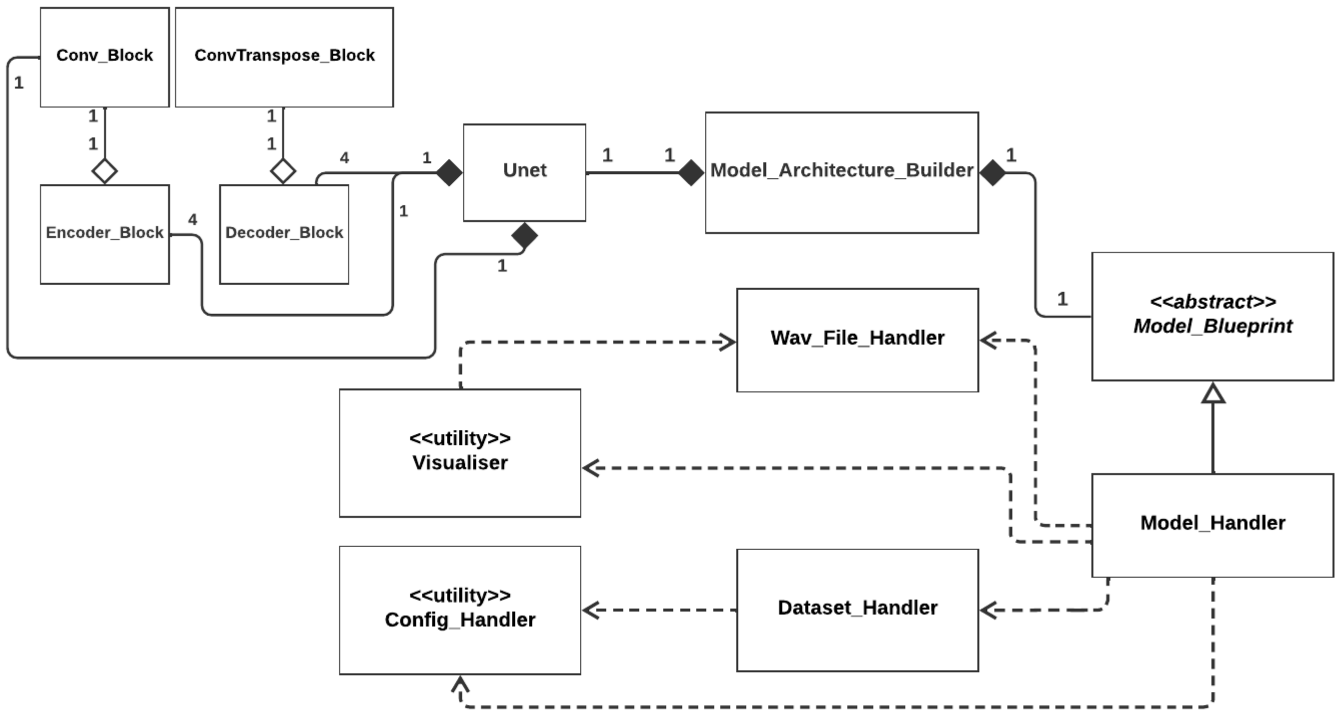


Figure 12. System Class Diagram

According to the figure above, the classes of the system are

a) **Config_Handler**: A utility class responsible for the configuration of the system such as initialising and making directories, storing paths to different project folders and initialising the project.

b) **Dataset_Handler**: This class is responsible for all the operations related to the dataset such as downloading the dataset, converting stems to wave files, saving the wave files and more.

c) **Visualiser**: Another utility class responsible for all the visualising tasks such as visualising spectrograms and waveforms of a signal.

d) **Model_Architecture_Builder**: This class is responsible for creating and returning the U-Net model with appropriate input and output shapes.

e) **Model_Blueprint**: An abstract class that has a subset of a model's core functionalities, such as generating training data, training the model, and predicting outputs. This class defines the layout of a model and must be implemented by a subclass.

f) **Model_Handler**: A model based on Model_Blueprint which has all the methods fully developed and is ready to be trained (or ready to predict and evaluate if it is already trained)

g) **Wav_File_Handler**: This class is responsible for all the operations on audio signals such as segmenting and denoising the signal, generating spectrograms, transforming a predicted signal from time-frequency domain to time-domain and more.

h) **Conv_Block**: This class creates the convolutional blocks of the U-Net architecture mentioned in Section 2.2.

i) **Encoder_Block**: Combines the Conv_Block and a max-pooling layer to create one encoder block of the U-Net's contracting path.

j) **ConvTranspose_Block**: Creates the deconvolutional blocks of the U-Net's expansive path.

k) **Decoder_Block**: Combined the ConvTranspose_Block with an upsampling convolutional layer to create one decoder block of U-Net's expansive path.

l) **Unet**: responsible for creating and returning U-Net architecture using an input layer, four Encoder_Blocks, one Conv_Block (the centre block of the U-Net) and four Decoder_Blocks.

Please refer to the GitHub repository for more documentation and testing of the classes mentioned here.

Workflow

In order to train a model on the musdb dataset, the system has the following workflow on a high level:

a) Downloading the musdb dataset, converting the stems to wav files and storing the required files (mixture.wav, vocals.wav and accompaniment.wav) for each song into project folder (if not already exists).

b) Creating the model architecture using Model_Architecture_Builder class.

c) Creating the model using Model_Handler class and the previously defined architecture.

d) Calling the training method of the model with appropriate loss function, epochs, batch size, learning rate and more.

e) Preprocessing the data and using the stored spectrograms of the segments

f) Training the model using the training data generator for the specified number of epochs.

g) Storing the model after training is complete

In the next chapter, the system and workflow mentioned here will be used to train several models with different parameters and the results of their predictions will be compared.

Chapter 4: Experiments and Comparisons

4.1 Trained Models and Parameters

To compare the different approaches and training methods, four models were developed with different loss functions (as mentioned in [12]). The parameters used for training these models are explained below.

Training Data

The training data for each model is the two-second long segment spectrograms of each song in the training directory of musdb dataset (as mentioned in [12]). To increase the randomness of the data and avoid the model to overfit on a certain order of segments, the music orders are shuffled in every epoch and the batch of data in each step consists of randomly shuffled spectrogram segments of a song.

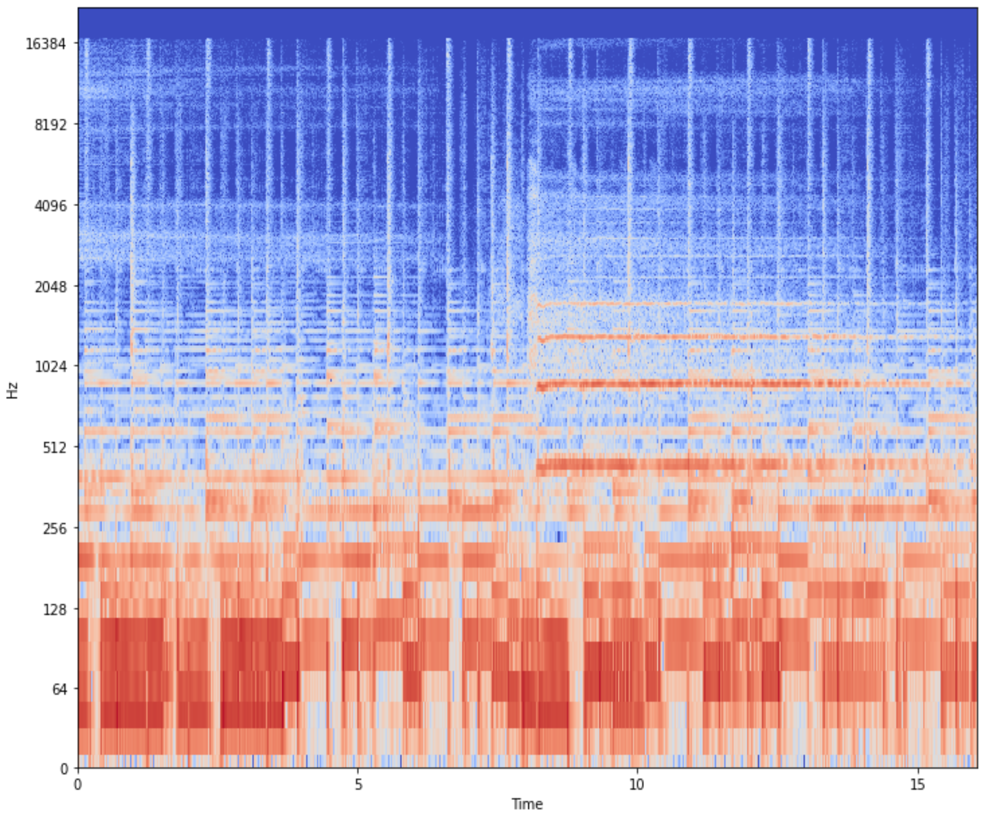


Figure 13. A non-shuffled batch of a song

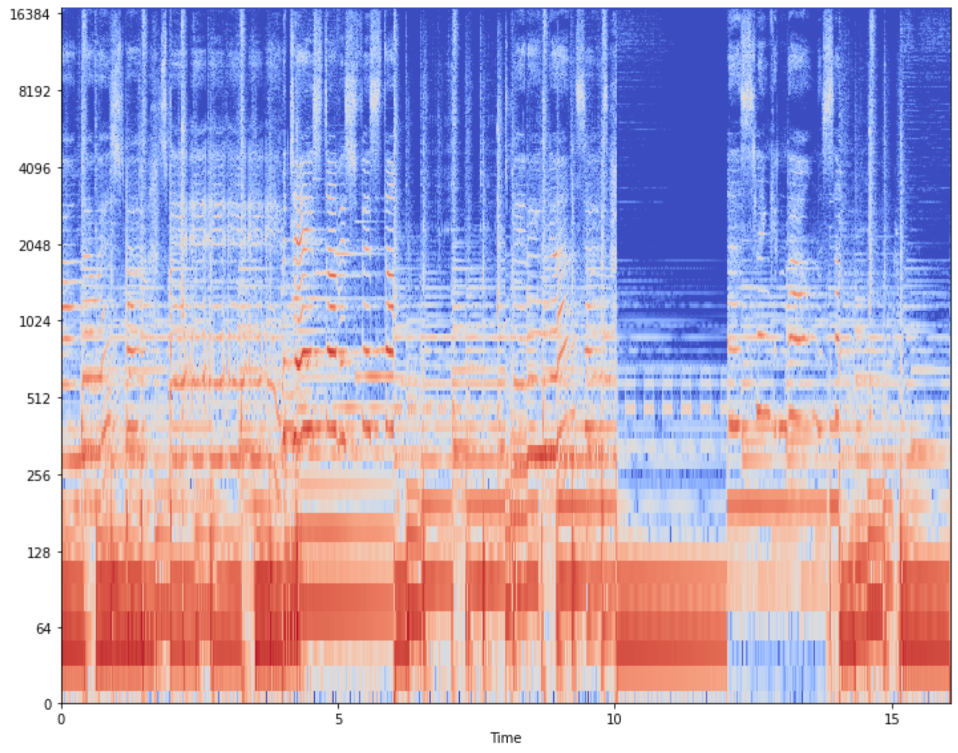


Figure 14. A shuffled batch of a song

The shuffled batch allows the model to learn the true underlying patterns without depending on the sequence of segments in a certain order. In general, more randomness in training data is usually preferred to avoid the model overfitting.

Loss Function

Each of the four models used the same loss function of equation (12) but with a different weighting factor (α).

$\alpha = 1.0$ was used to train a vocal separator as the loss function only calculates the loss of vocals. $\alpha = 0.0$ means the loss function only calculates the accompaniment loss thus the models with this loss function are supposed to separate only the vocals of the songs. $\alpha = 0.5$ was used to separate from both vocals and accompaniments by putting the same weight on the loss value of both sources and finally, $\alpha = 0.707$[1] was used to put more weight on the vocals' loss as in many songs, vocals usually have lower volume than accompaniments and putting more weight on the vocals'

[1] This value was calculated using α∗(the mean of 2-norm of vocal) = (1 − α)∗(the mean of 2-norm of accompaniment). [12]

loss should separate them better than using the same weights on both sources. The chosen values are based on [12].

The models were trained for 40 epochs with a learning rate of 0.001 for the first 20 epochs and 0.0001 for the rest.

4.2 Comparision

After training the models, they were evaluated on the test set using a signal difference function defined at [20] (with modifications). This function calculates the difference between the reference and the estimated signals in time and time-frequency domains.

The mean and median of these metrics were calculated for all the songs in the test dataset. Lower numbers show less difference and more similarity between reference and estimated signal. If two signals are exactly the same, the value will be 0.

Table below shows the calculated metrics:

	Mean Vocal Difference	***Median Vocal Difference***	***Mean Accompaniment Difference***	***Median Accompaniment Difference***
1. $\alpha = 1.0$ ***(vocal separator)***	*916.072*	*826.469*	*815.277*	*801.687*
2. $\alpha = 0.0$ ***(accompaniment separator)***	*1051.922*	*966.136*	*788.230*	*787.648*
3. $\alpha = 0.50$ ***(equal source separator)***	*894.879*	*831.509*	*790.867*	*795.772*
4. $\alpha = 0.707$ ***(more weight on vocals)***	*1055.377*	*818.539*	*790.717*	*798.284*

Table 2. The signal difference metrics of predicted and actual sources in the test dataset

According to the table, model 1 has a lower median difference than models 3 and 4. The median is larger than model 4 but the mean is significantly lower which shows the model is not as affected by the outliers as the model 4. Therefore, model 1 has the best performance at separating the vocals using both difference metrics

Model 2 had the worst vocal separation performance as the loss value of vocals was not used in the training loss function because of the value of $\alpha = 0.0$.
This model had the best performance in separating the accompaniments using both metrics.

Model 3 had a good performance on both separating the vocals and the accompaniments. Both median and mean values are close which shows the model was not affected by the outliers as much as other models.

Model 4 had a similar performance at separating the accompaniments as model 3 and overall good performance at separating the vocals. The only downside of this model was the effect of outliers on separating the vocals which are determined by the high mean vocal difference value.

The estimated accompaniments have less difference to their reference signals compared to vocals which is mainly due to the higher number of instrumental sources in a song. This will increase the similarity and reduces the difference.

These metrics prove the same results as [12] by using different evaluation metrics. Model 1 performs the best vocal separation and Model 2 performs the best accompaniment separation.

Chapter 5: Conclusion

In this research project, the basics of BSS and different BSS approaches were discussed. The deep learning approach to BSS was explained in-depth and several segmentation models were trained using U-Net architecture. The models were trained using a balanced loss function to output estimates of multiple sources and the overall performance of the models was acceptable.

Pre-processing steps such as segmenting the data, generating spectrograms of audio files and shuffling the data randomly to train more generalised models were discussed. Post-processing techniques such as a windowing denoise function were introduced to improve the quality of the estimations by removing some of the artefact noises.

Results of [12] were also concluded using a different set of evaluation metrics and the best models for the separation operations were introduced.

The source codes for this project are available in the GitHub repository and the tested scripts can be found there.

Future work

The models' performance could be improved using other preprocessing techniques such as data augmentation. This will allow the model to be trained on a larger dataset for a longer time. The model can also generalise more using the augmented training data.

To improve the quality of outputs, better denoising functions should be examined to remove the artefact noises from the estimations. Also, many other deep learning approaches such as using U-Nets with binary masks or Reinforcement learning can be used to improve the performance of the outputs.

The same architecture can also perform multi-source separation by using a different loss function (13). This will be an area of research and improvement in the future.

Chapter 6: Review

This research was very challenging and really interesting at the same time. The idea of having a system to learn patterns without any human interaction and perform the source separation which is a mathematically challenging task at an acceptable level shows the applications of deep learning.

The main challenge of this project was learning the concepts. BSS is an active area of research and very theoretical. Processing the audio signals and learning about sound was another interesting aspect of the project.

Another issue with this approach was the training time of the models. Each of the models mentioned in Chapter 4 was trained for one or 2 days which can lead to issues when having a fixed deadline to meet.

The metrics used to evaluate the results of the models are different from the BSS evaluation toolkit (e.g SDR, SIR, SNR and more) which is due to the fact that these metrics only produce an acceptable value if the predictions perfectly align with the reference signal. A difference in one frame of the data causes these metrics to show very bad results. The U-Net architecture developed used a different system of paddings compared to the one mentioned in [12]. These paddings that were used to match the input and output shape are most probably the reason behind frame mismatching thus leading to a bad evaluation. The frame mismatch issue should be fixed in the next version to allow for better evaluation.

In conclusion, this project taught me many new concepts of deep learning and what techniques to use to improve the model accuracy. It was a great research topic to read about.

References

[1] Comon, P. and Jutten, C., 2010. *Handbook of blind source separation*. Amsterdam: Elsevier.

[2] Roy, V. and Shukla, S., 2019. Designing Efficient Blind Source Separation Methods for EEG Motion Artefact Removal Based on Statistical Evaluation. *Wireless Personal Communications*, 108(3), pp.1311-1327.

[3] Bendoumia, R., 2019. Efficient Sparse Blind Source Separation Algorithm for two-Channel Acoustic Noise Reduction. *Algerian Journal of Renewable Energy and Sustainable Development*, [online] 01(01), pp.31-40. Available <https://ajresd.univ-adrar.edu.dz/index.php?journal=AJRESD&page=article&op=view&path%5B%5D=30&path%5B%5D=23> [Accessed 28 March 2022].

[4] Roma, G., Simpson, A., Grais, E. and Plumbley, M., 2016. Remixing musical audio on the web using source separation. [online] Available <https://smartech.gatech.edu/bitstream/handle/1853/54602/WAC2016-93.pdf?sequence=1&isAllowed=y> [Accessed 27 March 2022].

[5] Gleich, D., 2022. *Principal Component Analysis and Independent Component Analysis with Neural Networks* [image] Available at: <https://www.cs.purdue.edu/homes/dgleich/projects/pca_neural_nets_website/> [Accessed 31 March 2022].

[6] Mitianoudis, N. and Davies, M., 2004. Audio source separation: solutions and problems. *International Journal of Adaptive Control and Signal Processing*, 18(3), pp.299-314.

[7] Yu, X., Hu, D. and Xu, J., 2014. *Blind source separation*. Singapore: John Wiley et Sons Singapore Pte. Ltd.

[8] Sawada, H., Ono, N., Kameoka, H., Kitamura, D. and Saruwatari, H., 2019. A review of blind source separation methods: two converging routes to ILRMA originating from ICA and NMF. *APSIPA Transactions on Signal and Information Processing*, 8(1).

[9] Tharwat, A., 2020. Independent component analysis: An introduction. *Applied Computing and Informatics*, 17(2), pp.222-249.

[10] Ronneberger, O., Fischer, P. and Brox, T., 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation.

[11] Jansson, A., Humphrey, E., Montecchio, N., Bittner, R., Kumar, A. and Weyde, T., 2017. Singing Voice Separation With Deep U-Net Convolutional Networks.

[12] Oh, J., Kim, D. and Yun, S., 2018. Spectrogram-channels u-net: a source separation model viewing each channel as the spectrogram of each source.

[13] Stoller, D., Ewert, S. and Dixon, S., 2018. Wave-U-Net: A Multi-Scale Neural Network for End-to-End Audio Source Separation.

[14] En.wikipedia.org. 2022. *Short-time Fourier transform - Wikipedia*. [online] Available at: <https://en.wikipedia.org/wiki/Short-time_Fourier_transform> [Accessed 2 April 2022].

[15] En.wikipedia.org. 2022. *Pure tone - Wikipedia*. [online] Available at: <https://en.wikipedia.org/wiki/Pure_tone#:~:text=In%20psychoacoustics%20and%20signal%20processing,phase%2Dshift%2C%20and%20amplitude.> [Accessed 2 April 2022].

[16] mathworks. n.d. *Spectrogram Computation in Signal Analyzer*. [online] Available at: <https://www.mathworks.com/help/signal/ug/spectrogram-computation-in-signal-analyzer.html> [Accessed 5 April 2022].

[17] Yao, S., Piao, A., Jiang, W., Zhao, Y., Shao, H., Liu, S., Liu, D., Li, J., Wang, T., Hu, S., Su, L., Han, J. and Abdelzaher, T., 2019. STFNets: Learning Sensing Signals from the Time-Frequency Perspective with Short-Time Fourier Neural Networks. [online] Available at: <https://arxiv.org/pdf/1902.07849.pdf> [Accessed 5 April 2022].

[18] Pigeon, D., n.d. *The non-linearities of the Human Ear*. [online] Audiocheck.net. Available at: <https://www.audiocheck.net/soundtests_nonlinear.php> [Accessed 5 April 2022].

[19] P. Kingma, D. and Lei Ba, J., 2014. ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION. [online] Available at: <https://arxiv.org/pdf/1412.6980.pdf> [Accessed 7 April 2022].

[20] Stack Overflow. 2022. *Similarity between two signals: looking for simple measure*. [online] Available at: <https://stackoverflow.com/questions/20644599/similarity-between-two-signals-looking-for-simple-measure/68219235#68219235> [Accessed 20 April 2022].