Instrument Identification from Mixed Audio Signals

Dissertation submitted in partial fulfillment of the requirements for the degree of

Master of Technology
in
Computer Science

by
Sagnik Bhowmick
CS2225

under the guidance of
Dr. Sarbani Palit
Computer Vision and Pattern Recognition (CVPR) unit

Indian Statistical Institute Kolkata - 700108 , India
Computer Vision and Pattern Recognition (CVPR) unit
Indian Statistical Institute, Kolkata India - 700108

Sagnik Bhowmick

June 2024

Computer Vision and Pattern
Recognition (CVPR) unit
Indian Statistical Institute,
Kolkata - 700108, India

CERTIFICATE

This is to certify that the dissertation entitled "**Instrument Identification from Mixed Audio Signals**" submitted by **Sagnik Bhowmick** to the Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of Master of Technology in Computer Science is an authentic and genuine record of the research work conducted by the candidate under my supervision and guidance. I affirm that the dissertation has met all the necessary requirements in accordance with the regulations of this institute.

June,2024

Sarbani Palit

Dr. Sarbani Palit
Professor
Compute Vision and Pattern Recognition Unit,
Indian Statistical Institute,
Kolkata-700108

Acknowledgement

My sincere appreciation goes out to Dr. Sarbani Palit, my advisor at the Computer Vision and Pattern Recognition Unit of the Indian Statistical Institute in Kolkata, for her guidance, ongoing support, and inspiration. The course on Digital Signal Processing taught by her, helped me a lot with my project. I want to express my gratitude to Dr. Swagatam Das who took our Machine Learning II course in the third semester. Without that course, it would have been very difficult to grasp the concepts required to complete this project. My sincere gratitude to all of the teachers at the Indian Statistical Institute for their invaluable advice, insights, and instruction, which gave my research a crucial new perspective.Also, I want to thank all the seniors at the SIPL (Signal and Image Processing Lab) lab for their constant mentoring. Finally, I want to express my gratitude to my parents and friends for their unwavering support.

Sagnik Bhowmick

Sagnik Bhowmick
M.Tech (CS), CS2225
Indian Statistical Institute

Declaration

I, Sagnik Bhowmick, a registered M.Tech student in Computer Science at the Indian Statistical Institute, affirm that I have fulfilled all the requirements set forth by the institute for the submission of my dissertation. I hereby declare that the project presented is original and the result of independent investigations and research conducted by myself, and it does not contain any form of plagiarism. The work has contributed to the development of new techniques and approaches. This research has not been previously submitted to any other university or institution in pursuit of a degree, diploma, or any other academic recognition. Furthermore, I declare that any text, diagrams, or materials sourced from external sources, including but not limited to books, journals, and online resources, have been appropriately acknowledged, referenced, and cited to the best of my knowledge and understanding.

[illegible]

Sagnik Bhowmick
M.Tech (CS), CS2225
Indian Statistical Institute

Abstract

Sound source separation has been an active research topic over the years. With the advent of deep learning, there has been many developments in this field. Some early works include the Independent Component Analysis(ICA), the Wave-UNet model with the advent of deep learning. Some recent works include the HTDemucs and Open-Unmix. Here, the work was done on the Open-Unmix architecture. The architecture involves spectrogram calculation using STFT, several Multi layer perceptron layers and three BiLSTM layers with skip connections.
A modified form of this architecture was involved in this project where transformer was used. The result showed a slight increase in the SDR levels and reduced training time.

Contents

List of Figures

Chapter 1

Introduction

Signal source separation, also known as blind source separation or independent component analysis, addresses the challenge of separating mixed signals into their individual sources without prior knowledge of the sources or the mixing process. This problem arises in diverse fields, including audio processing, biomedical signal analysis, and telecommunications. In real-world scenarios, signals captured by sensors or recording devices often contain a mixture of multiple sources, such as simultaneous musical instruments in audio recordings or overlapping brain signals in biomedical data. The goal of source separation is to untangle this mixture and extract the original sources as separate signals. This task presents several challenges, including blindness to source characteristics, underdetermined mixtures, and temporal and spectral overlaps between sources. Approaches to source separation include model-based methods like Independent Component Analysis (ICA) and data-driven techniques such as deep learning, particularly Convolutional Neural Networks (CNNs) and Recurrent Neural Networks

(RNNs). Applications of source separation span various domains, including enhancing speech signals in noisy environments, isolating specific physiological signals in biomedical recordings, and separating different users' signals in telecommunications systems. Continued research and advancements in computational techniques promise further innovations in source separation methodologies and their applications.

Chapter 2

Related Works

2.1 Independent Component Analysis

2.1.1 Motivation

If two microphones record two time signals

$$x_1(t) = a_{11}s_1 + a_{12}s_2....(1)$$

$$x_2(t) = a_{21}s_1 + a_{22}s_2.....(2)$$

where a_{11}, a_{12}, a_{21}, and a_{22} are some parameters that depend on the distances of microphones from the speakers. The estimation of the original signals s_1 and s_2 from only the recorded signals is called the cocktail-party problem. An example is shown below.

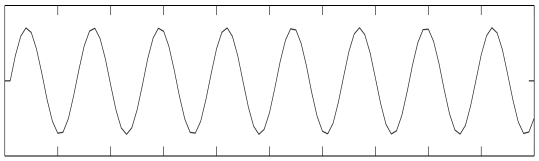


Figure 2.1: Original signal 1

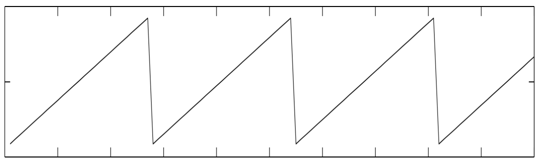


Figure 2.2: Original signal 2

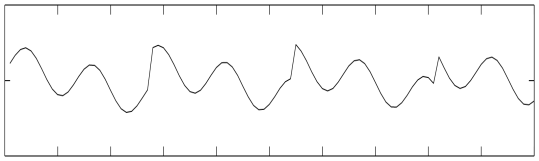


Figure 2.3: Mixed signal 1

2.1.2 Definition of ICA

To rigorously define ICA[2],we can use a statistical latent variables model. Assume that we observe n linear mixtures x_1 ..... x_n of n independent components. Considering a particular instance of time, we can drop the 't' from equations (1) and (2).

$$x_j = a_{j1}s_1 + a_{j2}s_2 + ... + a_{jn}s_n.....(3)$$

, for all j.
We assume each mixture x_j is a random variable and is the sum of source random variables. Without loss of generality, we assume that both the mixture variables and the independent components have zero mean. Let

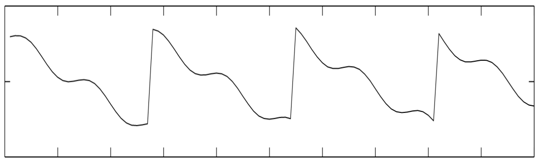


Figure 2.4: Mixed signal 2

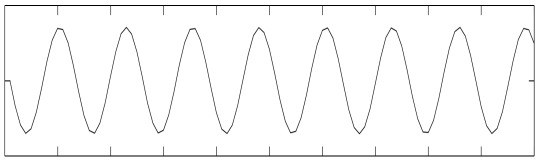


Figure 2.5: Estimated signal 1 from mixture

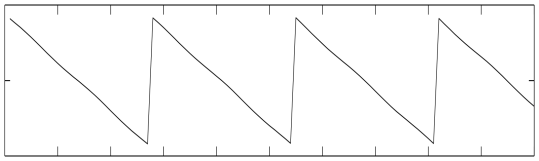


Figure 2.6: Estimated signal 2 from mixture

us denote x the random vector whose elements are the mixtures $x_1 \ldots..x_n$ and let s be the vector with elements $s_1, \ldots, s_n$. Let A be the matrix with elements a_{ij}. Using this vector matrix notation, the mixing model can be written as

$$x = \sum a_i s_i....(4)$$

. The statistical model in Eq.(4) is called independent component analysis, or ICA model. We are provided only with the random vector x, and we estimate s and mixing matrix A using it. We assume $s_i's$ are statistically independent. After estimating matrix A, we compute its inverse W and then obtain the independent component by

$$s = Wx$$

2.1.3 Principles of ICA estimation

The estimation of source signals is based on the concept of the estimated signals being non-gaussian. The Central

Limit Theorem[8], a classical result in probability theory tells that the distribution of a sum of independent random variables tends toward a gaussian distribution under certain conditions. To estimate one of the independent components, we consider a linear combination of the $x_i's$; $y = w^T x = \sum_i w_i x_i$, where w is the vector to estimated. Making a change of variables, $z = A^T w$, we get : $y = w^T x = w^T As = z^T s$. Thus y is a linear combination of s_i's. Since the sum of two or more independent components tend to be more gaussian than the variables themselves, the sum will be least gaussian when only one component of z is non-zero. Thus, we have to estimate a vextor w that maximizes the non-gaussianity of $w^T x$. This would correspond to a z in the transformed co-ordinate system with only one non-zero component. Thus maximizing $w^T x$ gives one of the independent components.

2.1.4 Measures of non-gaussianity

Negentropy

Entropy of a random variable H is defined by:

$$H(Y) = -\sum_i P(Y = a_i) \log P(Y = a_i)$$

where a_i's are the possible values of Y. For continuous valued random variables and vectors with density $f(y)$[3] :

$$H(Y) = -\int f(y) \log f(y) dy$$

.According to a result of information theory, gaussian variable has the largest entropy among all random variables of equal variance. Thus negentropy is defined as follows:

$$J(y) = H(Y_{gauss}) - H(y)$$

. Estimating negentropy using this definition would require us to have an estimate of the pdf. Thefore an approximation of negentropy is required.

Approximation of negentropy

The classical method of approximating negentropy is using higher order moments, for example:[7][6]

$$J(y) \approx 1/12E(y^3)^2 + 1/48kurt(y)^2$$

However, kurtosis being very susceptible to outliers this definition beomes non-robust. So we use the following approximation:[12]

$$J(y) \approx \sum_{i=1}^{p} k_i[E(G_i(y)) - E(G_i(v))]^2$$

Here k_i are some positive constants, and v is a Gaussian random variable with zero mean and unit variance. Variable y is assumed to have zero mean and unit variance. G_i are some non quadratic functions. In case we use only one non-quadratic function, the formula becomes:

$$J(y) \propto [E(G(y)) - E(G(v))]^2....(11)$$

The following choices of G have proved good results:

$$G_1(u) = 1/a_1 \log\cosh(a_1 u), G_2(u) = -\exp(-u^2/2).....(12)$$

where $1 \leq a_1 \leq 2$ is a suitable constant.

2.1.5 Maximum Likelihood Estimation

It is possible to directly formulate the ICA model and then estimate the model by a maximum likelihood method. Denoting by $W = (w_1, w_2, w_3, ..., w_n)^T$ the matrix A^{-1}, the log-likelihood looks like:

$$L = \sum_{i=1}^{T} \sum_{i=1}^{n} \log f_i(w_i^T x(t)) + T \log \det W ...(13)$$

where f_i are density functions of s_i and the x(t), t=1, ...,T are the realizations of x. The term —det W— comes from the rule for linearly transforming random variables and their densities[4]

2.2 Wave-UNet

2.2.1 Introduction

The Wave-U-Net is a one dimensional adaptation of the U-Net[11]. It can separate sources directly in the time domain and can take large temporal dependencies into account. Major changes in the architecture from the U-Net is that it used one dimensional convolutions and it uses linear interpolation to upsample feature maps instead of transpose convolutions.
The Wave-U-Net[12] can process multichannel audio and gives good multi-instrument as well as singing voice separation.

2.2.2 Architecture

The goal is to separate a mixture waveform $M \in [1, 1]^{Lm \times C}$ into K source waveforms $S_1, ..., S_K$ with $S_k \in [1, 1]^{Ls \times C}$ for all $k \in 1, ..., K$, C as the number of audio channels and L_m and L_s as the respective numbers of audio samples. For model variants with extra input context, we have $L_m > L_s$ and make predictions for the centre part of the input. It computes an increasing number of higher-level features on coarser time scales using downsampling (DS) blocks. These features are combined with the earlier computed local, high-resolution features using upsampling (US) blocks, yielding multi-scale features which are used for making predictions. The network has L levels in total, with each successive level operating at half the time resolution as the previous one. For K sources to be estimated, the model returns predictions in the inter-

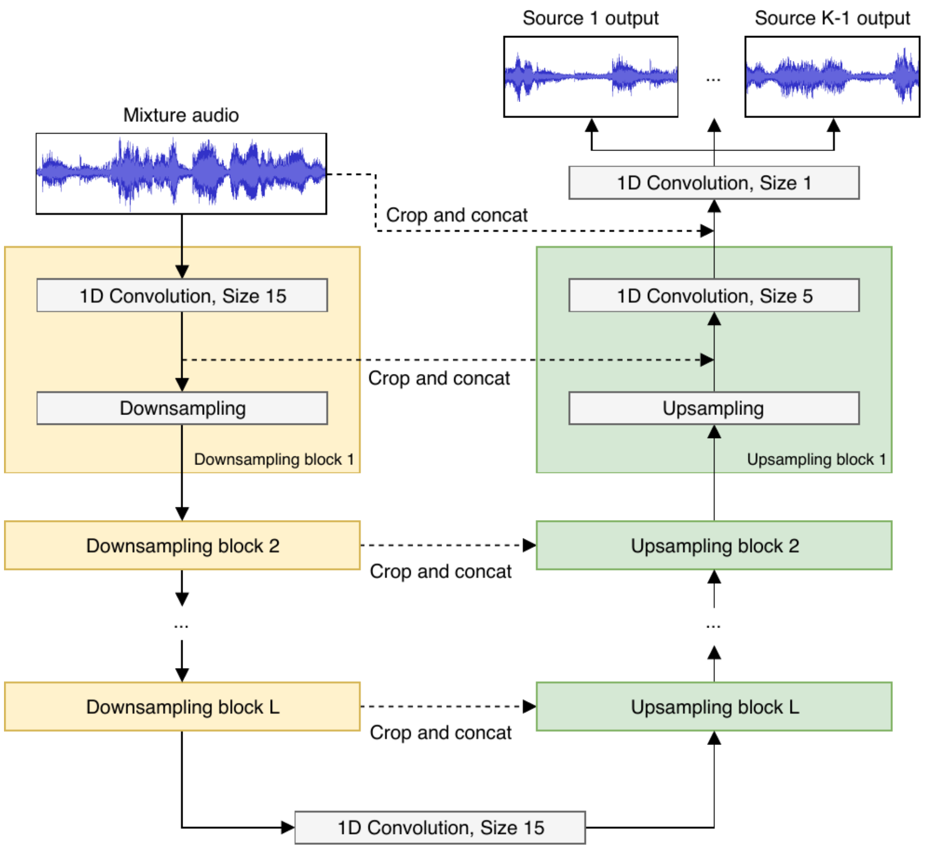
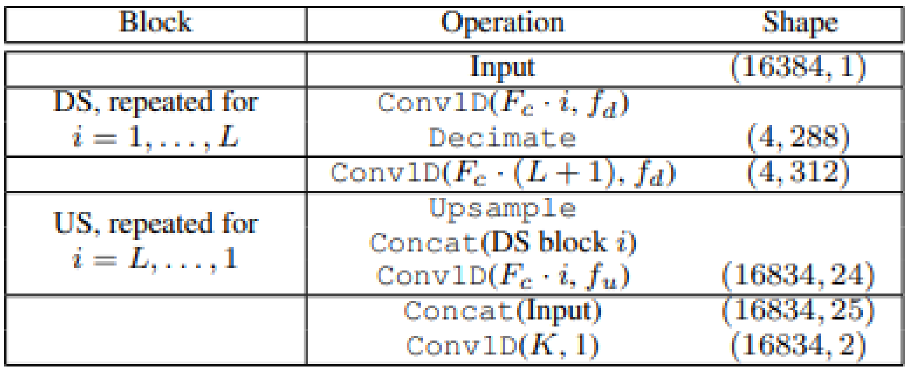


Figure 2.7: Wave UNet architecture

val (1, 1), one for each source audio sample. The detailed architecture is shown in Table 1. Conv1D(x,y) denotes a 1D convolution with x filters of size y. It includes zero-padding for the base architecture, and is followed by a Leaky ReLU activation (except for the final one, which uses tanh). Decimate discards features for every other time step to halve the time resolution. Upsample performs upsampling[5][9] in the time direction by a factor of two, for which we use linear interpolation (see Section 3.1.1 for details). Concat(x) concatenates the current, high-level features with more local features x. In extensions of the base architecture (see below), where Conv1D does not involve zero-padding, x is centre-cropped first so it has the same number of time steps as the current layer. The detailed architecture is shown in Table 1. Conv1D(x,y) denotes a 1D convolution with x filters of size y. It includes zero-padding for the base architecture, and is followed by a Leaky ReLU activation (except for the final one, which uses tanh). Decimate discards features for every other time step to halve the time resolution. Upsample performs upsampling in the time direction by a factor of two, for which we use linear interpolation (see Section 3.1.1 for details). Concat(x) concatenates the current, high-level features with more local features x. In extensions of the base architecture (see below), where Conv1D does not involve zero-padding, x is centre-cropped first so it has the same number of time steps as the current layer



Block	Operation	Shape
	Input	$(16384, 1)$
DS, repeated for $i = 1, \ldots, L$	`Conv1D`$(F_c \cdot i, f_d)$	
	`Decimate`	$(4, 288)$
	`Conv1D`$(F_c \cdot (L+1), f_d)$	$(4, 312)$
US, repeated for $i = L, \ldots, 1$	`Upsample`	
	`Concat`(**DS block** i)	
	`Conv1D`$(F_c \cdot i, f_u)$	$(16834, 24)$
	`Concat`(**Input**)	$(16834, 25)$
	`Conv1D`$(K, 1)$	$(16834, 2)$

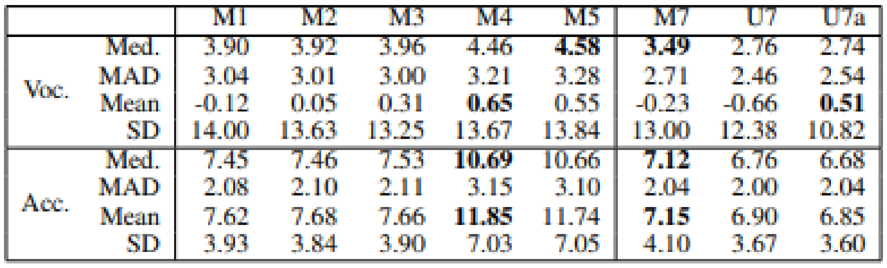
Figure 2.8: Block diagram of the base architecture. Shapes describe the final output after potential repeated application of blocks, for the example of model M1, and denote the number of time steps and feature channels, in that order. DS block i refers to the output before decimation. Note that the US blocks are applied in reverse order, from level L to 1.

2.2.3 Architectural Improvement

The baseline model outputs the k source estimates using k convolutional filters and then a tanh nonlinearity. We consider the mixture signal as a sum of its k source signal components. However this constraint is not used. Therefore, we use a difference output layer to constrain the outputs $\widetilde{S}^j$, enforcing $\sum_{j=1}^{k} S^j = M$: only K-1 convolutional filters with a size of 1 are applied to the last feature map of the network, followed by a tanh nonlinearity, to estimate the first K 1 source signals. The last source is then simply computed as $S^k = M - \sum_{j=1}^{k-1} S^j$.

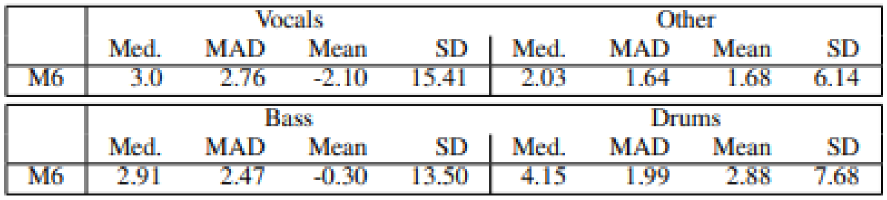
2.2.4 Results

The network was trained on the MUSDB18[10] dataset which consists of 150 songs where 100 songs are in the training set and 50 songs are in the test set. The network is trained over 75 songs and 25 songs are used as validation set. During training, audio excerpts are sampled randomly and inputs padded accordingly for models with input context. As loss, mean squared error (MSE) is used over all source output samples in a batch. ADAM optimizer with learning rate 0.0001, decay rates $\beta1 = 0.9$ and $\beta2 = 0.999$ and a batch size of 16 was used. The results for vocal separation model and the multi-instrument separation are as follows.



		M1	M2	M3	M4	M5	M7	U7	U7a
Voc.	Med.	3.90	3.92	3.96	4.46	**4.58**	**3.49**	2.76	2.74
	MAD	3.04	3.01	3.00	3.21	3.28	2.71	2.46	2.54
	Mean	-0.12	0.05	0.31	**0.65**	0.55	-0.23	-0.66	**0.51**
	SD	14.00	13.63	13.25	13.67	13.84	13.00	12.38	10.82
Acc.	Med.	7.45	7.46	7.53	**10.69**	10.66	**7.12**	6.76	6.68
	MAD	2.08	2.10	2.11	3.15	3.10	2.04	2.00	2.04
	Mean	7.62	7.68	7.66	**11.85**	11.74	**7.15**	6.90	6.85
	SD	3.93	3.84	3.90	7.03	7.05	4.10	3.67	3.60

Figure 2.9: Results for Wave UNet model for Vocal and Instrumental separation



	Vocals				Other			
	Med.	MAD	Mean	SD	Med.	MAD	Mean	SD
M6	3.0	2.76	-2.10	15.41	2.03	1.64	1.68	6.14

	Bass				Drums			
	Med.	MAD	Mean	SD	Med.	MAD	Mean	SD
M6	2.91	2.47	-0.30	13.50	4.15	1.99	2.88	7.68

Figure 2.10: Multi-Instrument separation results

..

Chapter 3

Dataset

The MUSDB18 is a dataset of 150 full length music tracks of different genres with their isolated drums, bass, vocals and others stems. MUSDB18 contains two folders, training set, composed of 100 songs, and a test set, composed of 50 songs. Supervised approaches should be trained on the training set and tested on both sets. All signals are stereophonic and encoded at 44.1kHz. The data from musdb18 is composed of several different sources:

• 100 tracks are taken from the DSD100 dataset, which is itself derived from The 'Mixing Secrets' Free Multitrack Download Library (opens new window).

• 46 tracks are taken from the MedleyDB[1] licensed under Creative Commons (BY-NC-SA 4.0).

• 2 tracks were kindly provided by Native Instruments originally part of their stems pack (opens new window).

• 2 tracks are from the Canadian rock band The Easton Ellises as part of the heise stems remix competition (opens new window), licensed under Creative Commons (BY-NC-SA 3.0). All files from the musdb18 dataset are encoded in the Native Instruments stems format (opens new window)(.mp4). It is a multitrack format composed of 5 stereo streams, each one encoded in AAC @256kbps. These signals correspond to:

0 - The mixture,
1 - The drums,
2 - The bass,
3 - The rest of the accompaniment,
4 - The vocals.
For each file, the mixture correspond to the sum of all the signals.

Tracks properties:
The Track objects which makes it easy to process the audio and metadata in a pythonic way:

• Track.name, the track name, consisting of Track.artist and Track.title.
• Track.path, the absolute path of the mixture which might be handy to process with external applications.
• Track.audio, stereo mixture as an numpy array of shape (nb-samples, 2).
• Track.rate, the sample rate of the mixture.
• Track.sources, a dictionary of sources used for this

track.
• Track.stems, an numpy tensor of all five stereo sources of shape (5, nb-samples, 2). The stems are always in the following order: ['mixture', 'drums', 'bass', 'other', 'vocals'],
• Track.targets, a dictionary of targets provided for this track.
Note that for MUSDB18, the sources and targets differ only in the existence of the accompaniment, which is the sum of all sources, except for the vocals. MUSDB18 supports the following targets: ['mixture', 'drums', 'bass', 'other', 'vocals', 'accompaniment', 'linear-mixture']. Note that some of the targets (such as accompaniment) are dynamically mixed on the fly.

The training was first run on the uncompressed .mp4 stems and later on .wav stems. The training with the latter was much faster almost taking 2 minutes less on an average per epoch.

Chapter 4

Approach and Methodology

4.1 Dataset Acquisition

The MUSDB18 is a dataset of 150 full lengths music tracks (10h duration) of different genres along with their isolated drums, bass, vocals and others stems.

MUSDB18 contains two folders, a folder with a training set: "train", composed of 100 songs, and a folder with a test set: "test", composed of 50 songs. Supervised approaches should be trained on the training set and tested on both sets.

All signals are stereophonic and encoded at 44.1kHz. The data is colected from Sigsep Datasets.
Click here to access the dataset. A python package to parse and process the MUSDB18 dataset, the largest open access dataset for music source separation. The tool was originally developed for the Music Separation task as part of the Signal Separation Evaluation Campaign (SISEC). Training MUSDB18 using open-unmix comes with several design decisions that we made as part

of our defaults to improve efficiency and performance:

- chunking: we do not feed full audio tracks into open-unmix but instead chunk the audio into 6s excerpts (–seq-dur 6.0).
- balanced track sampling: to not create a bias for longer audio tracks we randomly yield one track from MUSDB18 and select a random chunk subsequently. In one epoch we select (on average) 64 samples from each track.
- source augmentation: we apply random gains between 0.25 and 1.25 to all sources before mixing. Furthermore, we randomly swap the channels the input mixture.
- random track mixing: for a given target we select a random track with replacement. To yield a mixture we draw the interfering sources from different tracks (again with replacement) to increase generalization of the model.
- fixed validation split: we provide a fixed validation split of 14 tracks. We evaluate on these tracks in full length instead of using chunking to have evaluation as close as possible to the actual test data.

4.2 Data Preprocessing

The time domain signal is first preprocessed by transforming into its spectrogram. A spectrogram is a visual representation of the spectrum of frequencies in a signal as they vary with time. The spectrogram is the magnitude squared of the STFT:

$$|X[n,k]|^2$$

Next, to reduce the computational load during training, the number of frequency bins are reduced by cropping the spectrogram to 16kHz since the music bandwidth will be within 16kHz and this corresponds to 520 frequency bins in the spectrogram.

4.2.1 Spectrogram and STFT

The Discrete Fourier Transform (DFT) is a mathematical technique used in signal processing and related fields to transform a finite sequence of equally spaced samples of a function into a sequence of coefficients of a finite combination of complex sinusoids, ordered by their frequencies. Essentially, it converts a signal from the time domain to the frequency domain. It is defined as :

$$X[k] = \sum_{n=0}^{n=N-1} x[n] e^{-j\frac{2\pi}{N}kn}$$

Here,

- N : total number of samples in the signal x[n]
- k : component of frequency domain

However, the DFT does not provide information about when the frequencies occur in time. To analyze signals whose properties evolve over time, we can use the Short-Time Fourier Transform (STFT). The STFT gives both temporal and frequency information about the signal. The idea of STFT is to divide the signal into frames of length L. Then apply DFT to each frame to get the frequency information of each frame and compute it using the FFT algorithm. The STFT expression is as follows :

$$X[n,\lambda) = \sum_{m=-\infty}^{\infty} x[n+m]w[m]e^{-j\lambda m}$$

Here,

- n : location in time
- λ : frequency component
- m : sample in the current frame starting from n

Sampling the frequency at $\lambda_k = 2\pi/Nk$ such that N¿L :

$$X[n,k] = \sum_{m=0}^{L-1} x[n+m]w[m]e^{-j\frac{2\pi}{N}kn}$$

The kth frequency bin involves x[n], x[n+1], x[n+2],....., x[n+L-1].
The window function we use here is the Hanning window defined as :

$$w[n] = 0.5(1-\cos(\frac{2\pi n}{M-1}))$$

where n=0,1.....M-1. Generally, window size = frame size.

• Hop size = number of samples slid to right to get the new frame.
• Number of frequency bins = $\frac{framesize}{2} + 1$
• Numer of frames = $\frac{No.of samples - framesize}{hopsize} + 1$
The spectrogram is the magnitude squared of the STFT:

$$|X[n,k]|^2$$

4.3 Architecture of Open-Unmix[13]

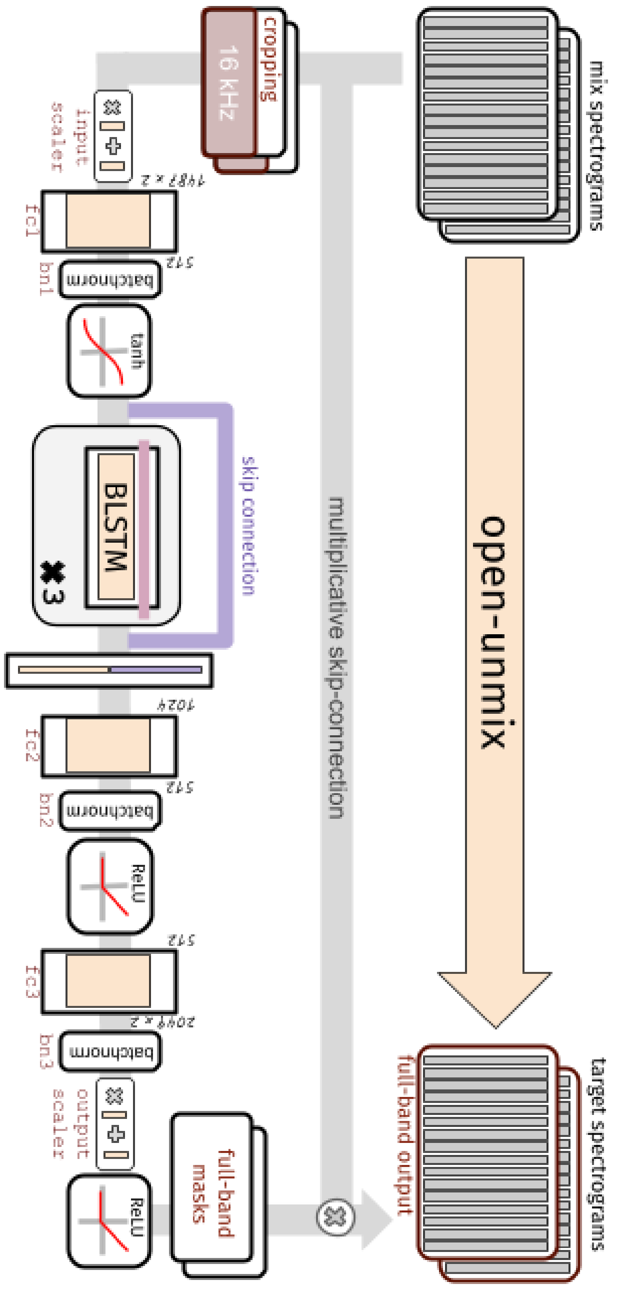


Figure 4.1: Open-Unmix Archicture

After the spectrograms are genrated, the number of frequency bins are limited 16 kHz. The input then passes through the Input Scaler component:

4.3.1 Input Scaler:

The input scaler standardizes each frequency bin such that the data in each frequency bin has $mean = 0$ and $standard - deviation = 0$. The output of this layer is fed into a fully connected layer fc1.

4.3.2 fc1

Input dimension for this layer is : (nb_samples, nb_channels*nb_bins)
Output dimensions : (nb_samples, hidden size)
The hidden size is the dimension of the bottleneck layer and here it is set as 512. Next, the output of this layer is batch-normalised.

4.3.3 bn1

The output of the fc1 layer is passed through the bn1 layer where it is batch-normalised.

Batch Normalization

Batch normalization operates by normalizing the output of a layer by subtracting the batch mean and dividing by the batch standard deviation. This is applied independently to each feature in the layer. Additionally, batch normalization introduces learnable parameters (scale and shift) to allow the model to adapt the normalized output to the specific task, increasing the representational power of the network.

The benefits of batch normalization are multifold. First and foremost, it helps mitigate the internal covariate shift, leading to faster convergence during training. By keeping the activations within a certain range, batch normalization allows for more stable gradients, which in turn accelerates the training process. Furthermore, batch normalization acts as a form of regularization, reducing the reliance on techniques like dropout and enabling the use of higher learning rates without risking divergence.

4.3.4 tanh **activation layer**

The output of the batch norm layer of dimension 512 then passes through a layer of tanh activation tanh is an activation function used in neural networks. It is defined as :

$$f(x) = \frac{e^x - e^-x}{e^x + e^-x}$$

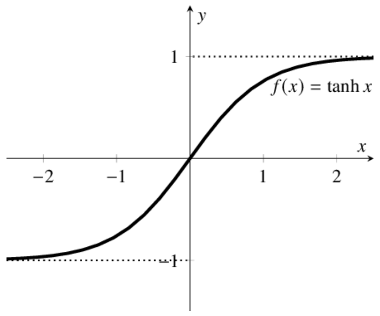


Figure 4.2: tanh curve

4.3.5 BiLSTM layers

Next, the output of tanh layer is fed to a series connection of 3 consecutive BiLSTM's. The BiLSTM is a type of recurrent neural network(RNN) that processes sequential data in both forward and backward directions. It involves LSTM's with bidirectional processing so that it can capture both past and future context of the input sequence. Components and functionality of BiLSTM :
LSTM

LSTM is a type of RNN that introduces memory cells and gating mechanisms to selectively retain and forget information over time.

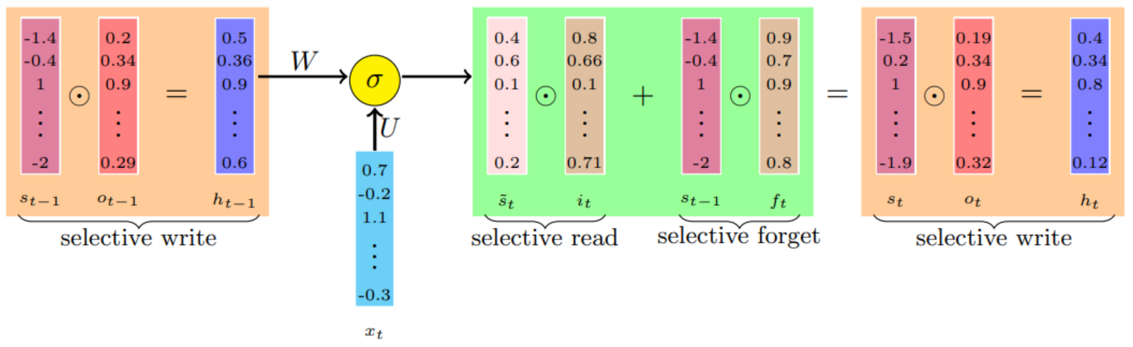


Figure 4.3: LSTM Architecture

The full set of equations for LSTM are as follows :
Gates

- $o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$
- $i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$
- $f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$

States

- $\tilde{s} = (Wh_{t-1} + Ux_t + b)$
- $s_t = f_t \cdot s_{t-1} + i_t \cdot \tilde{s}$
- $h_t = o_t \cdot (s_t)$

$rnn_{out} = h_t$

4.3.6 concat

There exist skip connections from the output of tanh layer to the output of BiLSTM layers. The output from the tanh layer is concatenated with the output from BiLSTM layers. So the dimension of this concat layer is (1024,1)

4.3.7 fc2

Next, the output from the previous layer passes through a fully connected network fc2 where the output of this layer is of dimension (512, 1). The output of fc2 is batch normalised again and fed to a ReLU activation layer keeping the dimension same as (512, 1).

4.3.8 fc3

The output from the last layer is passed through another fully connected layer fc3 such that the output of this layer is (nb_frames * nb_samples, nb_output_bins * nb_channels). This layer is again batch normalised and scaled as earlier.

4.3.9 Activation layer

Finally the network passes through a ReLU activation layer where the input spectrograms before cropping are concatenated to give the full band output.

4.4 Training

The network was trained on the MUSDB dataset.The train folder contains 100 songs. Certain design decisions were made for efficient performance:

- chunking : Instead of training with entire songs, it is trained with 6s audio excerpts.
- Track sampling : To avoid bias for continuous excerpts, the 6s excerpts are chosen randomly from one track in one epoch. Specifically, 64 samples are chosen from each track randomly in each epoch, the batch size in training being 64.
- source augmentation: we apply random gains between 0.25 and 1.25 to all sources before mixing. Furthermore, we randomly swap the channels the input mixture.
- random track mixing: for a given target we select a random track with replacement. To yield a mixture we draw the interfering sources from different tracks (again with replacement) to increase generalization of the model.

Also, for faster loading and processing, the mp4 dataset was converted to wav files from command prompt com-

mands.
Each model (vocals, bass, drums and others) were trained for 100 epochs, keeping the batch size as 16 and other parameters as follows:

- STFT window length in samples : 4096
- STFT hop length in samples : 1024
- learning rate : 0.001
- bandwidth for LSTM processing : 16kHz
- number of workers for data loading : 16

The same parameters were used for both the base model and the model where the BiLSTM layers were replaced by a transformer.
The time taken on average for each epoch for the base model was 20:04 minutes.
The time taken on average for each epoch for the transformer model was 17:29 minutes.

4.5 Architectural modification

The modification done to the baseline model was that the three sequential BiLSTM layers were removed and replaced by a transformer which was trained in the same manner as mentioned before. The transformer model premises itself upon attention, the technique of masking input data to emphasize certain features. In a single training epoch in a transformer, the input and output data follow the steps shown in Fig. 4.4.

Key advantages of Transformers over BiLSTM are:

- Parallelization : Transformers enable efficient parallelization which allows processing of multiple tokens simultaneously. The speed is incresed even with the help of GPUs compared to sequential processing of BiLSTM blocks.
- Long range dependencies : Transformer can handle long range dependencies in the input since it processes using attention mechanism which relates any two token in the input and is not dependent on the distance between them.
  BiLSTMs on the other hand, due to the sequential nature of their architecture, cannot capture long range dependencies and has the problem of vanishing gradients.
- Scalability ; Transformers are more scalable to very large datasets and models. The architecture of transformers, particularly the use of self-attention and feed-forward layers, allows for easier scaling compared to BiLSTM.

4.6 Loss function

The L2 loss function was used for training. The loss function is calculated between the predicted spectrograms and ground truth spectrograms. The initial positional encoding algorithm creates a vector of position values for each sample. The multi-head attention layers then learn different features from the input data in parallel, accentuating certain features for processing and comparison; this modified data is added to the original and then normalized before moving to the next layers. The training time is reduced and also SDR values show slight improvement in this case.

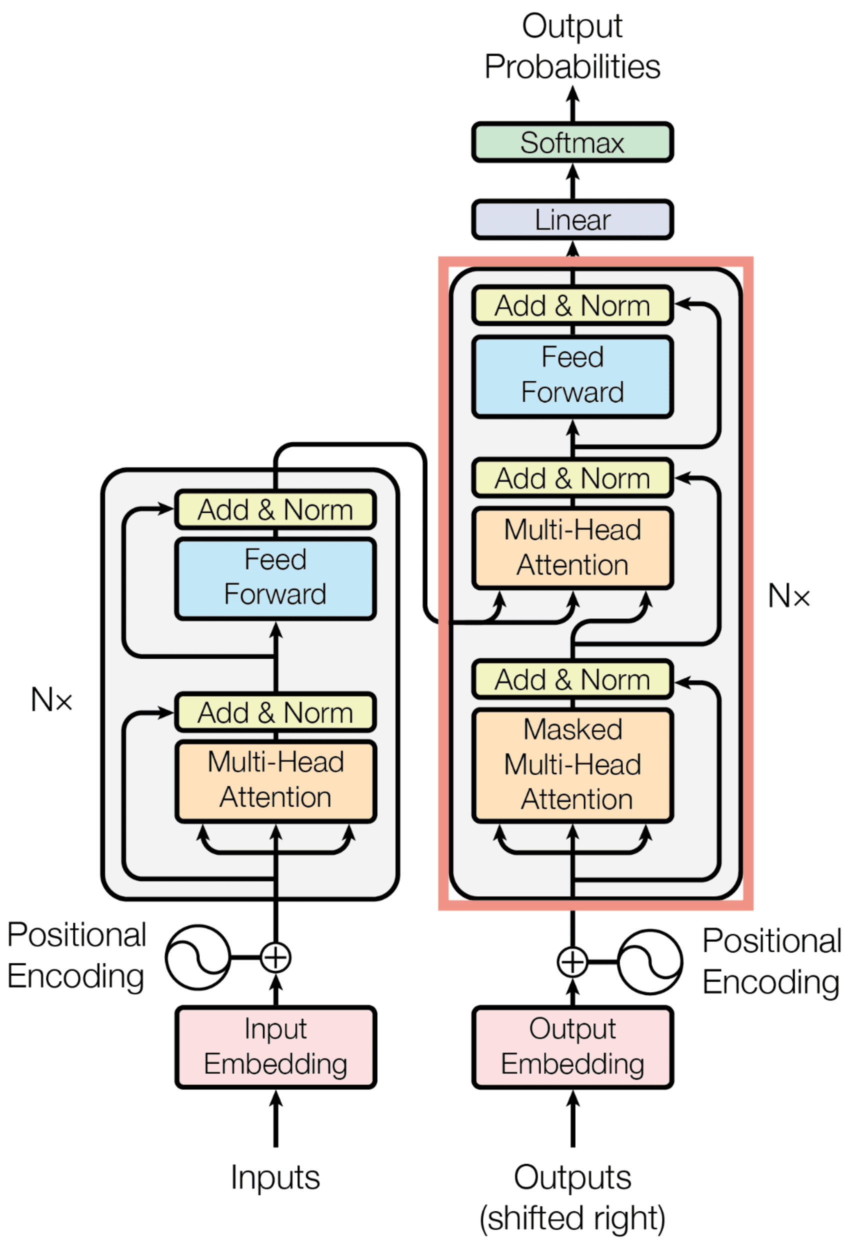


Figure 4.4: Transformer architecture

Chapter 5

Results and Discussions

5.1 Museval

The Museval python package is used to evaluate the test set of MUSDB dataset.
The package supports various data formats commonly used in source separation tasks, such as audio files in WAV format and multi-channel audio signals.The package includes implementations of various evaluation metrics commonly used in the evaluation of music source separation algorithms. These metrics may include signal-to-distortion ratio (SDR), signal-to-interference ratio (SIR), signal-to-artifact ratio (SAR).

5.1.1 Signal-to-distortion ratio

The Signal-to-distortion ratio is a metric to evaluate the quality of separated audio signals in the field of audio source separation. It gives a measure of distortion present in the separated signals compared to the original source signals. Mathematically, SDR is defined as the ratio of the energy of the original source signals to the energy of the distortion introduced during separation. A higher SDR value indicates a better separation quality, as it implies that the energy of the separated signals is closer to that of the original sources. SDR is calculated using the following formula:

$$SDR = 10\log_{10}\frac{\sum_t s_{original}(t)^2}{\sum_t (s_{original}(t)^2 - s_{separated}(t)^2)}$$

where :

- $s_{original}(t)$ is the original signal at time t.
- $s_{separated}(t)$ is the signal estimated at time t.

5.1.2 Signal-to-Interference Ratio(SIR)

SIR measures the ratio between the power of the true source signal and the power of the interference from other sources.

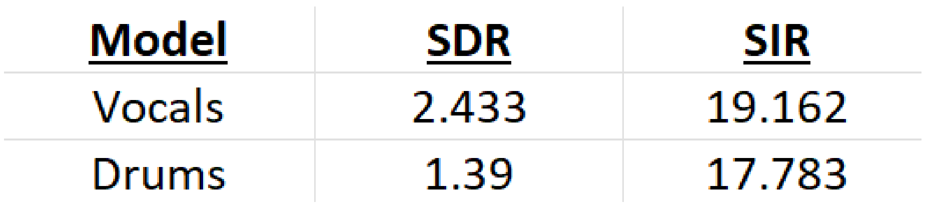
$$SIR = 10\log_{10}(\frac{||s_{true}||^2}{||e_{interf}||^2})$$

where:

- s_{true} is the original signal
- s_{interf} is the interference from all other signals

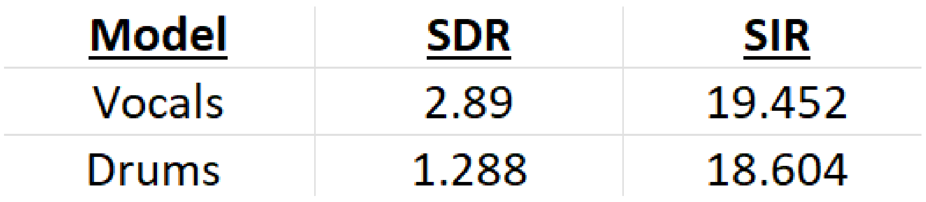
5.2 Performance of the models

Both the model were evaluated by executing the evaluate.py file and evaluated on the MUSDB18 test set where the audio files were already converted to .wav format. The directories where the evaluation results and the output file(predicted stem files) will be saved are provied in the command prompt while running evaluate.py.



Model	SDR	SIR
Vocals	2.433	19.162
Drums	1.39	17.783

Figure 5.1: Evaluation results with baseline model



Model	SDR	SIR
Vocals	2.89	19.452
Drums	1.288	18.604

Figure 5.2: Evaluation results with transformer model

Upon evaluation of the 'drums' and 'vocals' models for both the baseline and transformer architecture, it is seen that the SDR for 'vocals' was better for the transformer model than the baseline model. However, the SDR for 'drums' was a little less for the transformer model.
The SIR for both 'vocals' and 'drums' were seemed to increase in the transformer model compared to the BiLSTM model.

Chapter 6

Conclusion and Future Work

The Open-Unmix works with spectrograms and gives good results amonf state of the art architectures. Slight modifiction was made on this architecture to include a transformer and slight improvements were made on the SDR metric.
Several scopes of modifications to this architecture as well as dataset can be made for trying out better results. The code in the github repo has the following structure :

- data.py : includes different torch datasets that can be used to train Open-Unmix
- train.py : contains the code necessary to train the model
- model.py : contains the code where the architecture is defined
- test.py : has the code to unmix data from test datset

- eval.py : contains code to run evaluation using the museval package
- utils.py : contains tools for audio and metadata loading,etc.

Few modifications that can be made are:

- **Jointly train targets**
  Open Unmix is designed in a way such that multiple targets can be trained separately. This allows:

  - training can be distributed by running training of multiple models in parallel.
  - at test time the selection of different models can be adjusted for specific applications

  1. data.py can be modified where the dataset can be modified easily.
  2. train.py can be updated such that the loss functions can be changed

- **End to End Time Domain Models**
  The model can be converted to directly run from time domain input to time domain output if the STFT steps are removed.

Bibliography

[1] Rachel M Bittner, Justin Salamon, Mike Tierney, Matthias Mauch, Chris Cannam, and Juan Pablo Bello. Medleydb: A multitrack dataset for annotation-intensive mir research. In ISMIR, volume 14, pages 155–160, 2014.

[2] Pierre Comon. Independent component analysis, a new concept? Signal processing, 36(3):287–314, 1994.

[3] Thomas M Cover. Elements of information theory. John Wiley & Sons, 1999.

[4] Fourth Edition, Athanasios Papoulis, and S Unnikrishna Pillai. Probability, random variables, and stochastic processes. McGraw-Hill Europe: New York, NY, USA, 2002.

[5] Po-Sen Huang, Minje Kim, Mark Hasegawa-Johnson, and Paris Smaragdis. Singing-voice separation from monaural recordings using deep recurrent neural networks. In ISMIR, pages 477–482, 2014.

[6] Aapo Hyvärinen. New approximations of differential entropy for independent component analysis and

projection pursuit. Advances in neural information processing systems, 10, 1997.

[7] M Chris Jones and Robin Sibson. What is projection pursuit? Journal of the Royal Statistical Society: Series A (General), 150(1):1–18, 1987.

[8] Sang Gyu Kwak and Jong Hae Kim. Central limit theorem: the cornerstone of modern statistics. Korean journal of anesthesiology, 70(2):144, 2017.

[9] Santiago Pascual, Antonio Bonafonte, and Joan Serra. Segan: Speech enhancement generative adversarial network. arXiv preprint arXiv:1703.09452, 2017.

[10] Zafar Rafii, Antoine Liutkus, Fabian-Robert Stöter, Stylianos Ioannis Mimilakis, and Rachel Bittner. Musdb18-a corpus for music separation. 2017.

[11] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18, pages 234–241. Springer, 2015.

[12] Daniel Stoller, Sebastian Ewert, and Simon Dixon. Wave-u-net: A multi-scale neural network for end-to-end audio source separation. CoRR, abs/1806.03185, 2018.

[13] Fabian-Robert Stöter, Stefan Uhlich, Antoine Liutkus, and Yuki Mitsufuji. Open-unmix-a reference

implementation for music source separation. <u>Journal of Open Source Software</u>, 4(41):1667, 2019.