

BACHELOR THESIS
COMPUTING SCIENCE



RADBOUD UNIVERSITY

**A Component Analysis of Neural
Network Architectures for Spectral
based Audio Source Separation**

Author:

Bas van der Linden
s4597516

First supervisor/assessor:

Prof. David van Leeuwen
d.vanleeuwen@science.ru.nl

Second supervisor:

Nik Vaessen
nik.vaessen@ru.nl

Second assessor:

Prof. Djoerd Hiemstra
hiemstra@cs.ru.nl

January 17, 2022

Abstract

When an advancement is made in state-of-the-art performance for the audio source separation task, it is often the case that an entirely new architecture is published along with it. Even though these models contribute greatly to advancing the field, it can be difficult to determine what components of a new architecture make it perform better than the rest. The goal of this thesis is to analyze some of the basic methods that architectures can implement to improve audio source separation performance.

We looked at various relevant optimizers to find that RMSprop and Adam are both optimizers that we recommend for our task. We found that increasing the depth of feature extracting architectures such as an auto encoder can negatively impact the results, but if skip connections are added, deepening the network greatly improves performance. Furthermore, we saw that using mask synthesis over spectrogram synthesis improves both training stability and performance. We tested the use of dynamic mixing of source signals to create additional samples, but our measurements showed that this form of data augmentation negatively impacted our results, likely due to the large number of samples added to the training data. Finally, we compared using depth-wise separable convolutions instead of standard convolutions to show the trade-off between model size and performance.

Contents

1	Introduction	3
2	Preliminaries	7
2.1	The Fourier Transform	7
2.2	Short-Time Fourier Transform	10
2.3	Constant Overlap Add (COLA)	11
2.4	Depth-wise Separable Convolutions	12
3	Research	14
3.1	The Setup	14
3.1.1	The Dataset	14
3.1.2	Data Preparation	15
3.1.3	Audio Source Separation using Spectrograms	17
3.1.4	Model Evaluation	18
3.1.5	Network and Parameters	18
3.2	The Research	19
3.2.1	Research Preparation	20
3.2.2	Optimizer Comparison	20
3.2.3	Depth of Auto-encoders and UNets	22
3.2.4	Spectral Synthesis and Mask Synthesis	24
3.2.5	Data Augmentation using Dynamic Mixing	28
3.2.6	Depth-wise Separable Convolutions	30
4	Related Work	32
4.1	TRU-Net	32
4.2	Multi Modal Audio Source Separation	32
4.3	MMDenseNet	32

4.4 Spleeter	33
5 Conclusions	34
A Appendix	41

Chapter 1

Introduction

Audio Source Separation is the task of isolating a signal from a mixture of multiple signals. The individual signals that are heard in the mixture are called sources because that signal comes from a single source. Sources can be anything from vocals, and instruments to background noise. There are many useful application of audio source separation, such as the practical application of removing background noise from voice calls, or fun applications like being able to create instrumental tracks for karaoke, given any song. In Figure 1.1 we see a graphical illustration of what audio source separation is.

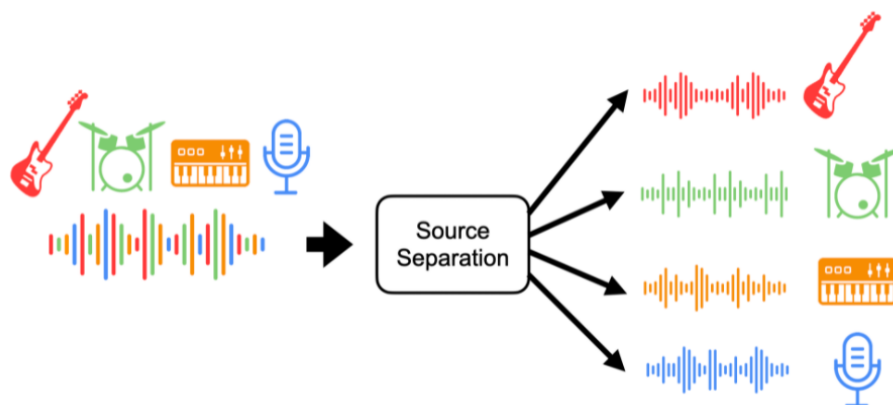


Figure 1.1: This figure illustrates what can be classified as a mix of sources, and what the separation of those sources entails [8].

When given a mixture of audio sources, we can define that mixture as the sum of all the sources. We define the mixed source signal as a function of time $y(t)$, as the sum of N source signals $x_n(t)$, where $n = 1 \dots N$. We can write this down as Equation 1.1 [8].

$$y(t) = \sum_{i=0}^N x_i(t) \quad (1.1)$$

In order to perform the separation of the signals x_n from the mixture y we will use a neural network. To have a neural network perform the separation we use supervised learning by training the network using a large amount of example songs where the separate sources are available. To train our models we used the MUSDB18 dataset [30], containing 150 songs and their separated sources. Using this data, the characteristics of the source signal can be learned by the neural network and be used to recognize the source for separation.

In the last decades we have seen lots of improvement in the audio source separation space. Before the popularization of deep neural networks (DNNs), various approaches to tackle source separation were introduced. These include local Gaussian modeling [7, 11], non-negative factorization [22, 20, 26], kernel additive modeling [24], and a combination of multiple approaches [28, 23, 10]. Currently, DNNs are the dominant method of achieving state-of-the-art performance in source separation tasks. In earlier work, feed forward fully connected neural networks (FNNs) were used [27, 38]. These would take multiple frames as input, concatenated, to utilize temporal context. These standard FNN were soon replaced by long short-term memory (LSTM) based models, that were able to model a longer context, as in [39]. LSTM based models showed performance increase over the previous FNNs, but they suffer from longer training times. Longer training times are detrimental for testing architecture changes or re-training the model for a new purpose, making it more difficult to find good models.

With the research being done on image classification using convolutional neural networks (CNNs) [21, 32, 35, 19, 36], preprocessing the audio signal using the Short-Time Fourier Transform (STFT) has proven to be a successful way to perform audio source separation using CNNs. CNNs are able to take long contexts, similar to LSTMs. The receptive field of a CNN is increased as the network deepens. However, modeling a larger context required deeper models, which suffer from vanishing gradients, leading to a decrease in performance. CNN based architectures such as UNets, ResNet [13] and densely connected CNNs [37] alleviated this problem by adding skip connections to join layers in the network. This allowed both the original input to be passed further down the network to preservation of details, and for the gradient to flow to earlier layers, directly bypassing the layers in between.

Throughout the years of research in the field of audio source separation we have seen many different approaches and model architectures. We have end-to-end models that work in the amplitude-time domain [33], and we have seen models that work in the frequency-time domain that use the STFT as preprocessing step. When a new model gets released that outperforms the current state-of-the-art, a comparison is given between the new model and previously viable models, to show the improvement in performance. The downside of these comparisons is that we do not get to see what design choices improved the model itself, but only as a whole when compared to other solutions. In this thesis we will focus on CNNs trained to perform audio source separation in frequency-time domain. We use the STFT on a segment of audio to transform it to a spectrogram which is then used as input to the network. The network is then also trained to synthesis the spectrogram of the target source as output. We will be using a UNet architecture to see how different design choices and techniques effect audio source separation performance of the architecture.

In Chapter 2 we go over the preliminary knowledge to aid in the understanding of our research. We explain the STFT and its subsidiary, the Fourier Transform (FT), which we use to process our data, in Sections 2.1 and 2.2. In Section 2.3 we go over the Constant Overlap Add (COLA) condition that is required for the STFT and its inverse to be lossless. We finalize the Chapter of preliminaries by explaining what depth-wise separable convolutions are in order to compare them with standard convolutions. We follow up the Chapter on preliminaries with Chapter 3 describing our research comparing design choices for spectrogram based audio source separation. We first look at what optimizer is best for audio source separation in Section 3.2.2. In Section 3.2.3 we discuss the effect of skip connections in a comparison between Auto Encoders (AEs) and UNets at various depths. Furthermore we discuss the approach of soft mask synthesis and how it compares to directly synthesising the source spectrogram in Section 3.2.4. Section 3.2.5 discusses the use of dynamic mixing as a data augmentation strategy. Finally, Section 3.2.6 goes over the effect of replacing standard convolution layers with depth-wise separable convolutions and what impact this has on model size and performance. After our research, we move on to Chapter 4 covering related work in the field of audio source separation. We end our thesis with a discussion about research we have done and the conclusions we have made, in Chapter 5.

Chapter 2

Preliminaries

In this chapter we will discuss the preliminary knowledge. Throughout this thesis the input to a model is a spectral representation of the audio. The audio is converted from a waveform to an 2D (frequency, time) representation, using the *Short-Time Fourier Transform* (STFT). The STFT applies the *Fourier Transform* (FT) on multiple overlapping time frames across the given audio. For a better understanding of the information that a spectrogram contains, we explain what the FT is in Section 2.1 and what the STFT is in Section 2.2. Next, we go over Constant Overlap Add (COLA) condition required for the STFT and its inverse to be lossless in Section 2.3. We conclude the preliminary knowledge Chapter with Section 2.4, explaining how depth-wise separable convolutions work for our comparison with standard convolutions.

2.1 The Fourier Transform

We explain the Fourier Transform (FT) by sketching the following scenario. Suppose we have an environment with two sources that create a sound, and both sources produce that sound with a constant volume and frequency. When recording both sources separately, we would get a measurement showing two pure sinusoids. For the sake of this example let *source one* produces sinusoid 1 shown in red (Figure 2.1), at a frequency of 500Hz, and *source two* produces sinusoid 2 shown in blue (Figure 2.1), at a frequency of 750Hz.

Since these sources produce sound in the same environment it would be difficult to record one of them without interference from the other. For sim-

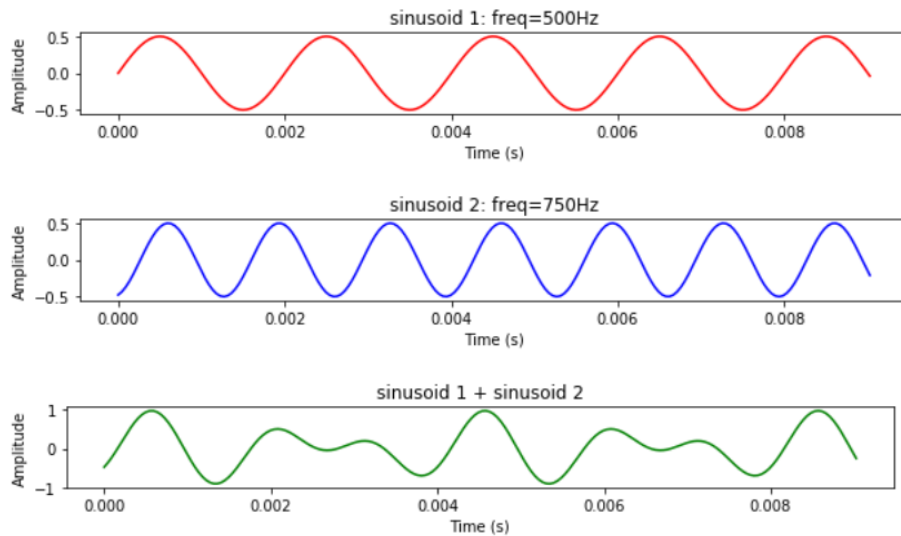


Figure 2.1: Two pure sinusoids and their super-positon

licity sake, we say that both sources are equally audible from the position of the measuring instrument. If this is the case, the sound that would be measured in the current environment would be the superposition of sinusoid 1 and 2, shown in green (Figure 2.1). The superposition is no longer a pure sinusoid. As we increase the number of sources that produce pure sinusoids, this impurity will further be amplified. The superposition of all the sinusoids would look increasingly random and it would be more difficult to distinguish what feature of the superposed waveform is caused by what signal.

To solve this problem and decompose the superposed waveform into the pure frequencies that made it, we can use the *Fourier Transform* (FT). The FT is used to transform a finite sequence, of a function of time, to a sequence of a complex valued function of frequency. Mathematically the FT of a time signal $y(t)$, where ξ is frequency and t is time, is given by

$$\text{FT}(\xi) = \int_{-\infty}^{\infty} y(t)e^{-2\pi it\xi} dt \quad (2.1)$$

For source separation, it is important to note that the FT is a linear transformation. This is helpful because this means we can take the FT of a mixture of noise and a source, and subtract the FT of the noise in order to get the FT of the source.

$$\text{FT}(y_1 + y_2) = \text{FT}(y_1) + \text{FT}(y_2) \quad (2.2)$$

After performing the FT on a signal, we are able to pick out and manipulate the frequencies that are of interest to us. Once the frequency domain changes have been made we can go back to the time domain with the *inverse Fourier Transform* (iFT). Here we see the importance of the FT being a linear transformation, because this allows us to remove the noise in the FT of a signal, and then transform it back into an audio signal using the iFT, to get the source without the noise. Mathematically the iFT is given as

$$y(t) = \int_{-\infty}^{\infty} \text{FT}(\xi) e^{2\pi i t \xi} d\xi \quad (2.3)$$

Before we take a look at an example, it is important to know how we look at the FT. Taking the FT of a signal gives a sequence of complex values as result. In order to get the magnitude at which the frequencies occur, we take the absolute values of the FT. When we visualize the FT we always show the magnitude. We can also recover the phase from the FT by taking the argument of the complex values.

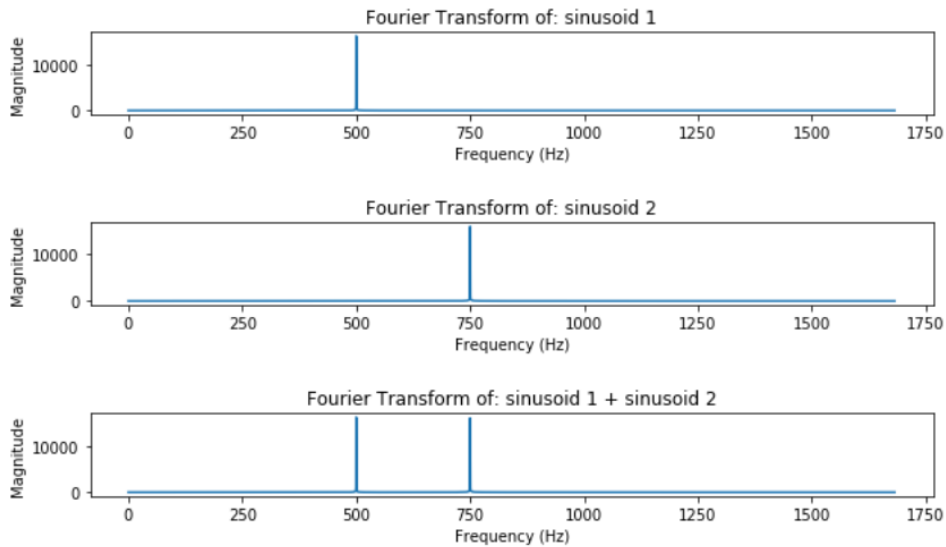


Figure 2.2: Fourier Transform of simple and complex sinusoids.

In Figure 2.2 we see the FT of the sinusoids from Figure 2.1. As expected, we can see a clear peak at 500 Hz for sinusoid 1, and a peak at 750 Hz for sinusoid 2. Looking at the FT of the superposition of the two sinusoids we can see the two frequencies clearly. We also observe the linear transformation property, shown by the fact that the FT of the superposition of the sinusoids is the same as the sum of the FT of sinusoid 1 and sinusoid 2.

2.2 Short-Time Fourier Transform

The Short-Time Fourier Transform (STFT) takes the FT one step further, with the goal to keep the time domain that is otherwise lost with the FT. The STFT performs the FT multiple times on different parts of the same signal. For the STFT we require a *window size* and *hop size*, which define for us the number of samples we use for the FT and how many samples we should step across time for the next window of samples to perform the FT on. Doing so for many windows of time across a signal gives us a visualization of how the intensity of certain frequencies changes throughout that signal. Intuitively, to go from multiple FTs to the visualisation of the STFT, we stack the FTs vertically next to each other in chronological order, and we represent the intensity that a frequency occurs at using a color scale. let us take a look at the STFT of the complex signal from Figure 2.1.

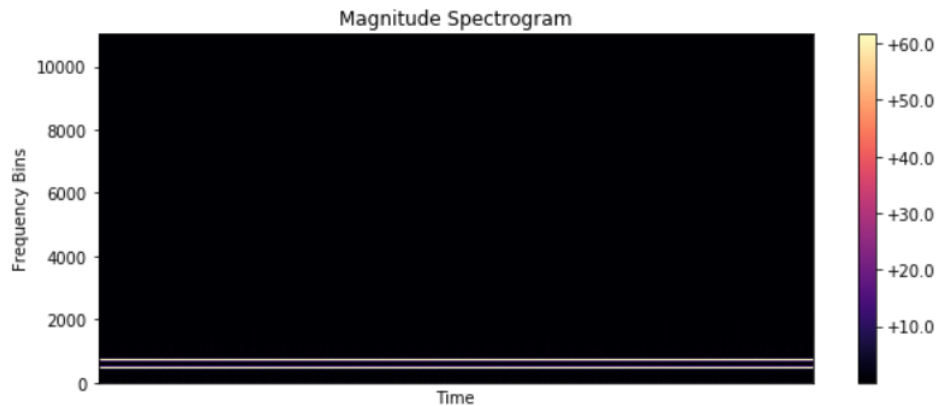


Figure 2.3: Short-Time Fourier Transform of a composed sinusoids from Figure 2.1

As we can see in Figure 2.3, both the frequencies of the pure sinusoids, that make up the superposed signal, persist over the entire duration at a constant magnitude. This transformation is used throughout this paper to create the model input. It is important to note that a window function is applied to each window before the FT is performed on that window. This window function has to be COLA compliant (2.3) in order for the STFT to be near perfectly invertable.

2.3 Constant Overlap Add (COLA)

The STFT is a lossy transformation if the window function used is not COLA compliant. It is common to take a hop size that is half the window size or less to refine the detail of the STFT. This causes the windows to overlap, and during the inverse-STFT, when the signal is reconstructed, these overlapping windows don't overlap smoothly. To solve this issue, a window function is applied to each window of samples that the FT is performed on. The window function usually limits the intensity towards the edges of a window, such that when they overlap with another window they yield the full intensity again. To achieve this near perfect overlap, the window function has to be COLA compliant. We use Figure 2.4 to illustrate what this means.

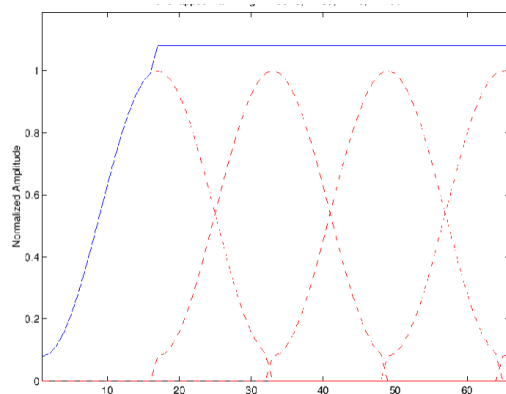


Figure 2.4: Shows multiple instances of the Hamming window function overlapping and their sum. [5]

In Figure 2.4 we see the Hamming window function in red, overlapping with multiple instances of itself. In blue we see the sum of the overlapping

windows. The fact that the Hamming window is COLA compliant at this amount of overlap can be seen by the blue line being perfectly horizontal.

2.4 Depth-wise Separable Convolutions

In Section 3.2.6 we will compare depth-wise separable convolutions and standard convolutions, analyzing the trade-off between the number of parameters and the performance. In this section we look at depth-wise separable convolutions and how they work differently than standard 2D convolution. Depth-wise separable convolutions are used as an alternative to standard convolutions with the purpose of having less parameters with comparable results. The ability to achieve comparable accuracy with less parameters is becoming increasingly valuable as machine learning becomes more and more popular in applications running on edge devices.

In order to explain depth-wise separable convolutions we are going to define a 3D network input F as having a width D_r , height D_r and depth M , being the number of channels. Instead of applying a single convolution layer, depth-wise separable convolutions are done using two steps. We first perform depth-wise convolution on the input F to get intermediate output $G1$, followed by point-wise convolution on $G1$ to get the final output $G2$.

1. Depthwise Convolution: Filtering Stage

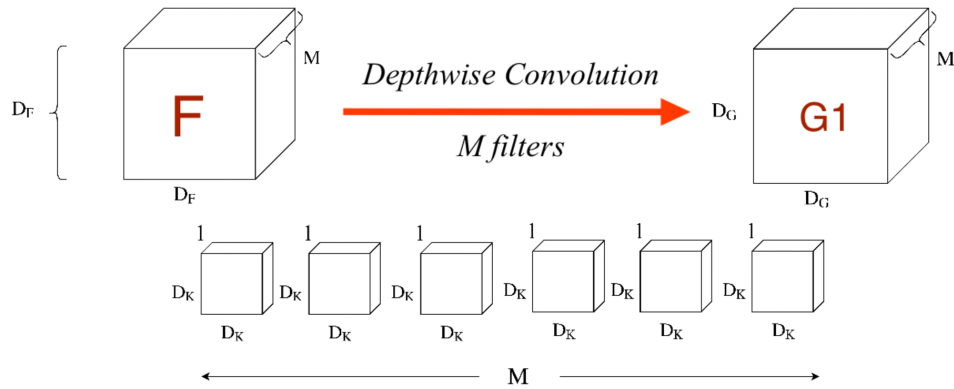


Figure 2.5: Illustration of depth-wise convolution [3]

Depth-wise convolutions are done by performing separate convolutions on each of the M input channels. For each of the channels in the input we

have one kernel, this results in an intermediate output $G1$ with the same dimensions as input F .

2. Pointwise Convolution: Filtering Stage

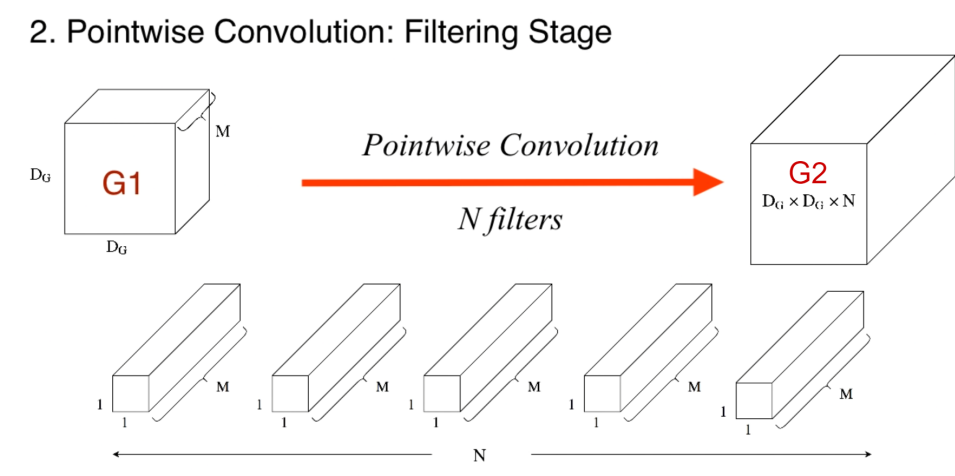


Figure 2.6: Illustration of point-wise convolution [3]

Next, we perform point-wise convolutions on $G1$. Point-wise convolutions are identical to standard convolutions but the kernel size is limited to $1 \times 1 \times M$. The number of filters chosen for the point-wise convolution dictates the number channels N in the output $G2$.

For a better understanding of depth-wise separable convolutions we can take a look at Figures 2.5 and Figure 2.6, which illustrate depth-wise convolutions and point-wise convolutions respectively.

Chapter 3

Research

This chapter contains the details on the research we have done. We start with Section 3.1 where we go over the setup of our research. In Section 3.2 we discuss the research we have done and analyze our measurement.

3.1 The Setup

The section starts with Section 3.1.1 which describes the dataset used to train and evaluate our models. In Section 3.1.2 we cover the processing steps that we apply to the data in the dataset before we use it for our research. We then explain how we perform audio source separation using the spectrograms from preprocessing, in Section 3.1.3. Section 3.1.4 explains how we evaluate and compare our models. Lastly, in Section 3.1.5 we go over the architecture and parameters used for training and inference during this research.

3.1.1 The Dataset

In order to answer our research questions we are going to train our models on the standard non-HQ variant of the MUSDB18 dataset[31]. The dataset contains a total of 150 full length songs with a variety of different genres, encoded in the Native Instruments stems format (.mp4)[16]. Each file is a multitrack format, composed of 5 stereo streams[29]. The tracks provided for each song are the *mixture*, *vocals*, *drums*, *bass* and *other accompaniment*. In total the dataset contains ~ 10 hours of audio, with the length of an

average song being 4 minutes. All the audio is available at a sample rate of 44100 Hz. The dataset is divided into a training set and a test set. The set used for training contains 100 songs, and the test set the remaining 50. We further discuss how this data is prepared for the purpose of our research in Section 3.1.2.

3.1.2 Data Preparation

The audio from the MUSDB18 dataset is first down sampled from the original sample rate of 44100Hz, to a sample rate of 22050Hz. Other work on spectral based audio source separation often down samples the audio to a sample rate of 16000Hz, however we find that this results in a significant audible loss of quality of the audio. For this reason we down sample to 22050Hz, which reduces the number of samples that need to be processed for a specific duration of the audio, without a noticeable loss in quality to the listener.

To prepare the audio for the STFT, we segment the full length songs and their sources. We have chosen for each segment to have a length of 65536 samples, resulting in each segment having a duration of 2.97 seconds. To go from one segment to the next, we move forward over the audio track by 32768 samples. This results in the segments overlapping by $\frac{1}{2}$, such that the overlap is 1.49 seconds in duration. When we segment the audio in this way, the average song with a duration of 4 minutes is divided into 159 segments. Each song is segmented and the segments are stored in separate files for training.

During training we compute the STFT of each segment. When we compute the STFT of a segment, we use a window size (WS) of 512 samples, a hop size (HS) of 128 samples, and we use the Hamming window (HW) function. When using this window size and hop size we have an overlap of $\frac{3}{4}$, which in combination with the Hamming windowing function is COLA compliant 2.3. Using the given parameters and the segment length (SL) we are able to calculate the format of our spectrograms.

$$\begin{aligned}
\text{FR} &= \frac{\text{SL} - \text{WS}}{\text{HS}} + 1 \\
&= \frac{65536 - 512}{128} + 1 \\
&= 509
\end{aligned} \tag{3.1}$$

$$\begin{aligned}
\text{FB} &= \frac{\text{WS}}{2} + 1 \\
&= \frac{512}{2} + 1 \\
&= 257
\end{aligned} \tag{3.2}$$

In Eq. 3.1 & Eq. 3.2 we see that computing the STFT of a segment (S) of length 65536, gives us a spectrogram with 509 frames (FR) on the time axis, and 257 frequency bins (FB) on the frequency axis. After transforming our audio segments to complex valued spectrograms, we want to compute the magnitude for each of those values. We compute the magnitude spectrogram by taking the absolute values of the complex spectrogram. We now want to convert the magnitude spectrogram to the decibel scale and then normalize the decibel values to be between 0 and 1. The current magnitude spectrograms contain values below 1, so before the decibel conversion we first increment all magnitude values by 1. After that, we move to the decibel scale by taking the \log_{10} of each value. Once in the decibel scale, our spectrogram values range between 0 and 2.10. To normalize the spectrograms we divide all values by 2.10 in order to get a range of values between 0 and 1. Eq. 3.3 shows the entire data preparation process as a single formula. Once all these steps have been applied, the data is ready for training.

$$\text{I} = \frac{1}{2.10} \cdot \log_{10} (|\mathcal{F}(S, \text{WS}, \text{HS}, \text{HW})| + 1) \tag{3.3}$$

3.1.3 Audio Source Separation using Spectrograms

In this Section we describe how we go about the task of audio source separation throughout our research. We start by processing the audio from the dataset, transforming all to audio segments to normalized magnitude spectrograms, as described in Section 3.1.2. We use the prepared spectrogram of the mixture of sources as input to our network architecture, training it to produce the normalized magnitude spectrogram for a single source, vocals in our case, as shown in Figure 3.1. We also include an example spectrogram for a mixture and its vocals in Sections A.2 and A.3.

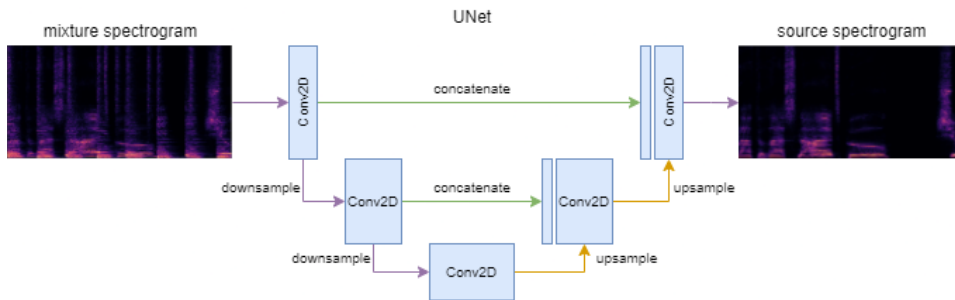


Figure 3.1: A UNet architecture producing a source spectrogram from a mixture spectrogram.

To reconstruct the complex valued spectrogram of the source using the predicted source magnitude spectrogram, we copy over the phase from the mixture’s spectrogram. In Eq. 3.4 & 3.5 we show how we obtain real component (RC) and imaginary component (IC) of the complex valued source spectrogram, using the mixture’s complex spectrogram (MCS) and the predicted source magnitude spectrogram (SMS). Once the complex valued spectrogram is constructed, we use the inverse Short-Time Fourier Transform on it, to calculate the source’s audio signal.

$$\text{RC} = \text{SMS} \times \cos(\text{phase}(\text{MCS})) \quad (3.4)$$

$$\text{IC} = \text{SMS} \times \sin(\text{phase}(\text{MCS})) \quad (3.5)$$

3.1.4 Model Evaluation

Signal to Distortion Ratio (SDR) is by far the most compared metric across various audio source separation papers. Even though SDR is an indication on the quality of the separation, it is not the definitive measure of quality. The developers of SDR mention this in their evaluation tutorial [9]. The authors show that two different predicted audio samples, with a distinct audible quality difference, achieve the same SDR. They emphasise that ‘the metrics are a good estimate of the quality, but human evaluation is the gold standard of measuring the quality of a system’ [9].

In our research, we do not need to compare our work with state-of-the-art work that reports SDR, instead we need to assess the performance difference between two of our models, to conclude how useful a change to a model is. To quantify the value of the quality at which people perceive the source separation to be, is very difficult. In order to compare our models, we need a metric that is sensitive to small differences in quality between the original source audio and the predicted source audio. We have chosen to use the Mean Squared Error (MSE) as the metric by which we compare our models. We wanted to use the full 100 training songs for training, so we used our test set, of 50 songs, to calculate the MSE validation loss during training, in order to compare our models. MSE is an error metric, this means that the lower the MSE the closer the predicted output is to the desired output. This is why we often report the Minimum Validation Loss (MVL) as the metric by which we compare our models in our research.

3.1.5 Network and Parameters

We implemented our own UNet architecture from scratch. We provide a detailed visualization of our UNet architecture in Section A.1. When we remove the skip connections and concatenations from a UNet architecture we are left with an architecture called an Auto Encoder. The Auto Encoder (AE) used in Section 3.2.3, is our UNet architecture without the skip connections and concatenation layers. To build and train these networks we used Tensorflow 2.0 [1]. In Table 3.1 we show the parameters used during the experiments.

Table 3.1: Contains the parameters used for the network and its training, in order to recreate the experiments. The *section* column indicates if a parameter is tweaked in a specific section, for the purpose of the experiment.

parameter	value	section
optimizer	Adam	3.2.2
learning rate	0.00001	-
batch size	8	-
layer type	conv2d	3.2.6
activation function	Leaky ReLU	-
output activation function	Leaky ReLU	3.2.4
loss function	Mean Squared Error	-
spacial dropout	10%	-
network depth	3	3.2.3
target source	vocals	-
output format	spectrogram	3.2.4

3.2 The Research

We begin with Section 3.2.1, where we go over the preparations we did before starting performing our main research. Our main research starts in Section 3.2.2 where we compare various relevant optimizers. In Section 3.2.3 we compare AEs and UNets at various depths, to see the effects of skip connections. Then we discuss the difference in performance when training a model to predict a soft mask instead of the source spectrogram, in Section 3.2.4. Section 3.2.5 details our experiments on using dynamic mixing as a data augmentation method. Finally, in Section 3.2.6 we compare standard convolutions to depth-wise separable convolutions to view the trade-off between performance and number of parameters.

3.2.1 Research Preparation

Before we started running our experiments we performed a couple of smaller experiments as preparation to ensure the following: avoid erratic validation loss curves, avoid over fitting and avoid dead weights for as many epochs as possible. We tested: spacial dropout, number of layers per depth level, activation functions, loss functions and initial learning rate. The outcome of the tests led us to use the following: a spacial dropout of 10%, 1 layer per depth level, the *Leaky ReLU* activation function, the Mean Squared Error loss function and an initial learning rate of 0.00001.

3.2.2 Optimizer Comparison

In this section we take a look at various relevant optimizers, and how they perform at the task of Audio Source Separation compared to the popular *Adam* optimizer. All the optimizers are used in combination with a UNet model that directly predicts the spectrogram of the target source audio. We compare each optimizer based on the validation loss over 100 epochs. To calculate the validation loss between the predicted source spectrogram and the target source spectrogram, we use the *Mean Squared Error* (MSE) loss function. The optimizers that we compared are: *Adadelta*[40], *Adagrad*[6], *Adamax*[17, 12], *Nadam*[4], *RMSprop*[15], *SGD*[34] and *Adam*[18].

To compare the optimizers we performed a randomly initialized training, three times, for each optimizer, totaling 21 experiments. In Figure 3.2 we plot the average validation loss of the three runs for each of the optimizers.

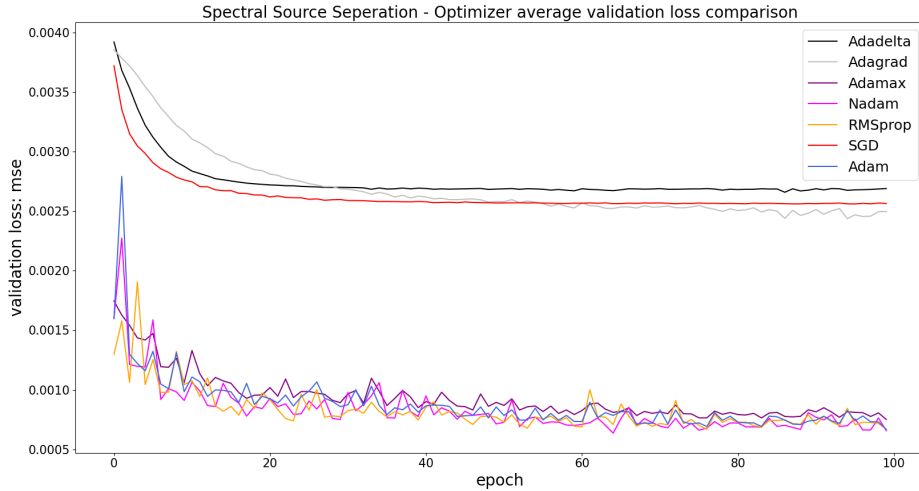


Figure 3.2: The validation loss (MSE) of models trained with different optimizers.

In Figure 3.2 we observe that *SGD*, *Adadelta* and *Adagrad* all perform more than twice as bad as the rest of the optimizers, based on average validation loss. They do show improvement in the first ~ 20 epochs, after which they level out. That the validation loss for these optimizers plateaus, may indicate overfitting, or that the optimizer is not able to help the network to grasp the underlying concepts/patters needed to separate the source audio from the mixture.

Adam, *RMSprop*, *Nadam* and *Adamax* all perform well, and have validation loss values in similar ranges. In order to compare these four optimizers in more detail we are going to compare the measurements in Table 3.2.

Table 3.2: Shows the minimum validation loss (MVL), average epoch runtime in seconds (AERT) and *Improvement Over Baseline* (IOB) for each of the four optimizers.

Optimizer	MVL	AERT (s)	IOB
Adam	0.000653	344.63	0.00%
RMSprop	0.000608	376.39	7.32%
Nadam	0.000625	404.09	4.40%
Adamax	0.000711	359.21	-8.22%

Due to the popularity of the Adam optimizer we have used it as a baseline to compare the other measurements with. The *Improvement Over Baseline* (IOB) shows us how much an optimizer’s Minimum Validation Loss (MVL) differs, percentage wise, from the MVL of the Adam optimizer. Table 3.2 also shows the *Average Epoch Runtime* (AERT) in seconds in order to compare the optimizers based on how long they take to achieve their MVL.

We compare the IOB to see that *Adamax* performs the worst out of all the optimizers in Table 3.2, performing $\sim 8\%$ worse than the *Adam* optimizer. In addition, we also observe that the AERT of *Adamax* is longer than *Adam*’s AERT. This tells us that *Adamax* both performs worse and takes longer to do so than *Adam*, leading us to conclude that *Adamax* is not an optimizer we recommend for spectrogram based audio source separation.

Looking at the other entries in Table 3.2 we see that *RMSprop* and *Nadam* both outperform *Adam* based on MVL. *RMSprop* achieves a larger IOB than *Nadam*, and it is able to do so in less time than *Nadam*. This tells us that *RMSprop* is the best alternative to the *Adam* optimizer at this task, with an IOB of $\sim 7\%$. On average *RMSprop* takes $\sim 9\%$ longer than *Adam* to achieve its MVL.

In conclusion, we can recommend the use of the *RMSprop* optimizer over the *Adam* optimizer when training to achieve the lowest MVL for spectrogram based audio source separation. However, during testing, or when time is limited and training times are long, we can still recommend the *Adam* optimizer to achieve adequate performance in a shorter time.

$$\text{IOB} = \frac{MVL_{Adam}}{MVL_{Optimizer}} - 1 \quad (3.6)$$

3.2.3 Depth of Auto-encoders and UNets

This section has two main focuses. First, we look at the effect of skip connections in UNets, in a comparison between AEs and UNets as we increase their depth. Second, we observe the effectiveness of using deeper UNets, when trying to achieve the minimum validation loss.

Depth variations in Auto Encoders and UNets

In order to compare Auto Encoders (AEs) and UNets we performed three runs for each of the architectures at five different depths, totalling 30 runs. In Figure 3.3 we plot the average validation loss for each of the three runs.

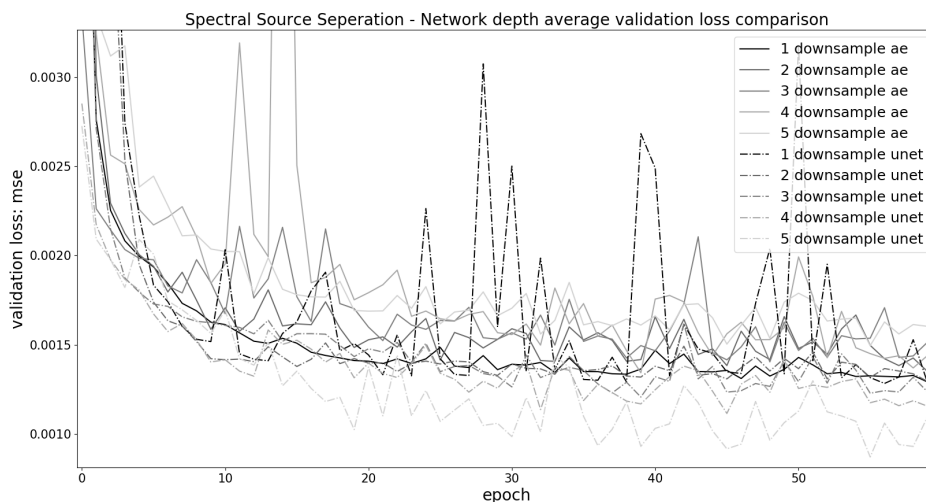


Figure 3.3: The average validation loss (MSE) of unet models trained at various depths.

Figure 3.3 shows us the opposing effect of deepening AEs versus UNets. The lines are colored such that the lighter the color, the deeper the network. Looking at validation loss lines for the AEs (solid) we see that as the network gets deeper, the average validation loss goes up. Similarly, but opposite, the validation loss lines for the UNets (dotdash) show that as the network becomes deeper, the average validation loss goes down. In conclusion, this indicates that as AEs get deeper and the dimension of the latent space decreases, it becomes harder to recover the larger dimension's higher detail, required for the output. The use of skip connections in UNets allow the detail of the higher dimension input to bypass the down sampling. This allows the network to utilize both the original high dimension input details as well as the extracted spectral characteristics from the latent space to be used to synthesise the output.

The Effectiveness of Deepening UNets

Table 3.3: Shows the minimum validation loss (MVL), average epoch runtime in seconds (AERT), number of parameters (#PARAM) and *Improvement Over Previous* (IOP) for unets at different depths.

Depth	MVL	AERT (s)	#PARAM	IOP
1	0.001249	563.22	5759	0.00%
2	0.001179	567.70	27855	5.61%
3	0.001167	570.55	115567	1.03%
4	0.001017	574.38	465071	12.86%
5	0.000825	585.43	1860399	18.88%

Table 3.3 contains the IOP metric, which tells us the percentage improvement in MVL compared to the previous depth. The IOP is a positive percentage for all non trivial depths, indicating that in our experiments, a depth increase always led to lower MVL. We can see significant increases in prediction quality when we use a depth of 4 or 5.

Increasing the depth even more would most likely lower the MVL even further. However, at some point the depth is limited by the resolution of the input which can no longer be down sampled.

$$\text{IOP} = \left(\frac{MVL_{n-1}}{MVL_n} - 1 \right) \quad (3.7)$$

3.2.4 Spectral Synthesis and Mask Synthesis

In this section we compare direct synthesis to mask synthesis. Doing direct synthesis means the task of the network is to produce a source spectrogram that matches the target source spectrogram. The goal of mask synthesis is to produce an output of the same dimensions as the target source spectrogram, which is then multiplied with the mixture spectrogram, to get the target source spectrogram. In Figure 3.4 we show how the network is trained to predict the source spectrogram as output. In Figure 3.5 we show how the network is trained to produce a soft mask as output.

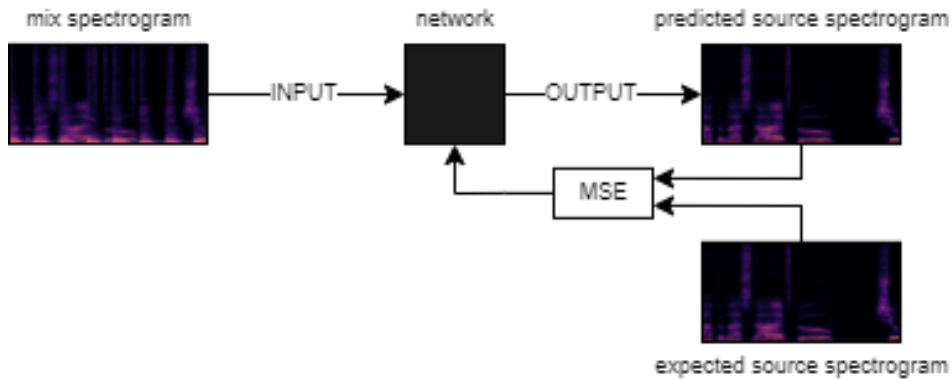


Figure 3.4: Illustrates how the network trains to produce a spectrogram.

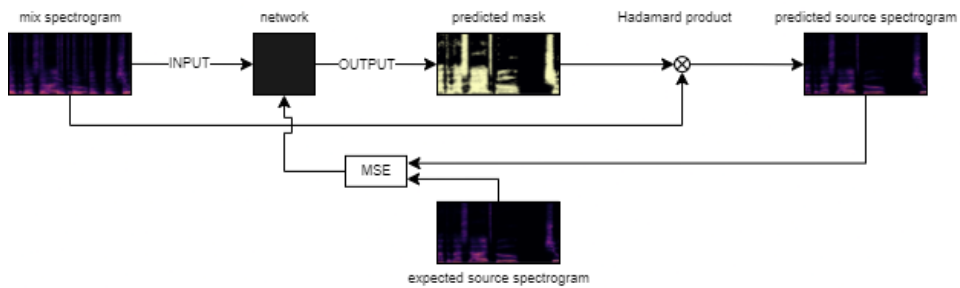


Figure 3.5: Illustrates how the network trains to produce a soft mask for source separation.

In order to compare direct and mask synthesis we performed three runs for each output type, spanning 100 epochs. We tested direct synthesis with Leaky ReLU as the output layer activation function. The two masks we tested were, with Leaky ReLU as the output layer activation function, and with Swish as the output layer activation function. In Figure 3.6 we can see the minimum validation loss (MVL) up until each epoch.

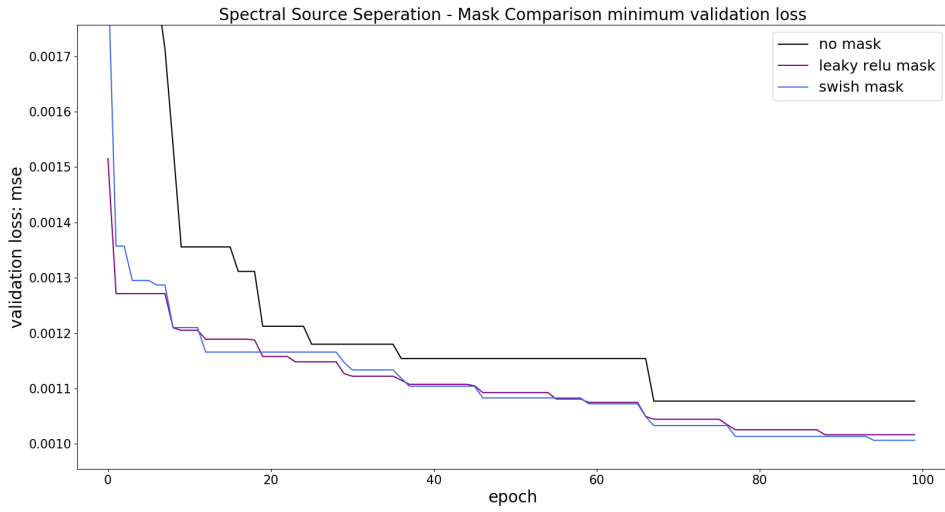


Figure 3.6: The minimum achieved validation loss (MSE) up until each epoch for unet performing direct synthesis, leaky relu masking, and swish masking.

Figure 3.6 shows us that after 100 epochs both the masking methods have achieved a better validation loss. We also observe that in direct synthesis the rate at which a lower validation loss is achieved decreases as more epochs elapse. On the contrary, both mask synthesis methods show a constant rate of improvement, but not every improvement is as significant.

We look at Figure 3.7 to see the effect that the output method has on training stability.

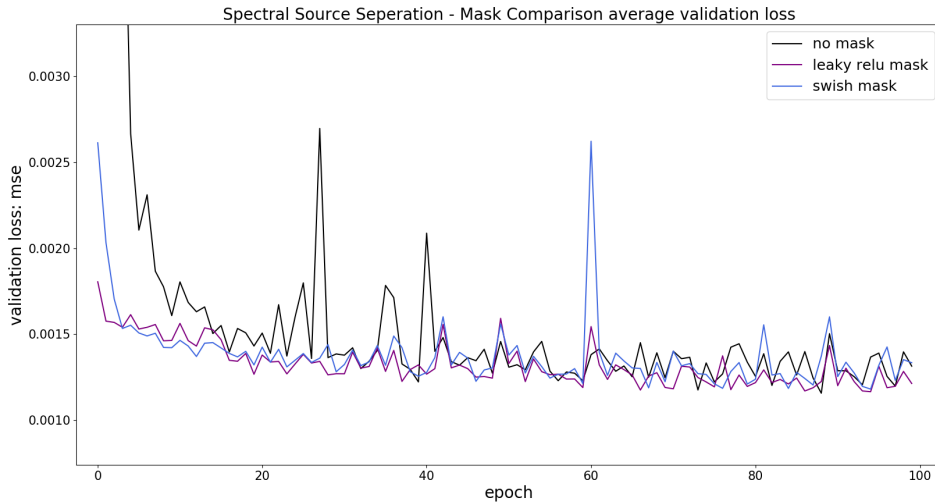


Figure 3.7: Average validation loss (MSE) for each output methods

In Figure 3.7 we see that both direct synthesis and Swish mask synthesis exhibit large peaks in validation loss. Direct synthesis shows training instability at the first half of training, and Swish mask synthesis has a rare occurrence of one peak. The results show that Leaky ReLU mask synthesis is more stable during training then the other two methods, when looking at average validation loss.

Table 3.4: Shows the minimum validation loss (MLV), average epoch runtime in seconds (AERT) and *Improvement over no mask* (IONM) for each synthesis method.

method	MSE	AERT (s)	IONM
direct	0.001077	537.19	0.00%
leaky relu mask	0.001016	545.92	5.97%
swish mask	0.001006	574.53	7.04%

When looking at the IONM in Table 3.4, we see that both the mask methods outperform the direct synthesis method. When we compare the AERT of direct synthesis and leaky relu masking we see that masking takes a little more time on average then the direct synthesis method. Swish masking however takes over 37 seconds longer per epoch. When we take both AERT and IONM into account, we observe that the swish mask advantage over

the leaky relu mask is less important. Showing that if training time is of concern, leaky relu masks are the optimal way to get the lowest validation loss in a shorter time period then the rest.

$$\text{IONM} = \frac{MVL_{direct}}{MVL_{mask}} - 1 \quad (3.8)$$

3.2.5 Data Augmentation using Dynamic Mixing

In this section we discuss dynamic mixing as a data augmentation method. This means we created new track segments by taking different source segments for different tracks, and combined them together to give a unique combination of sources. These new mixed audio segments are then used as additional training data.

For each of the experiments we used the same network with the same weight initialization, only changing the training data. For the experiments that used data augmentation we created 5000 additional audio segments from the 100 training tracks. Let us take a look at Figure 3.8, showing the validation loss over 100 epochs, for half the dataset, half the dataset plus the augmented segments, the full dataset, and the full dataset plus the augmented segments.

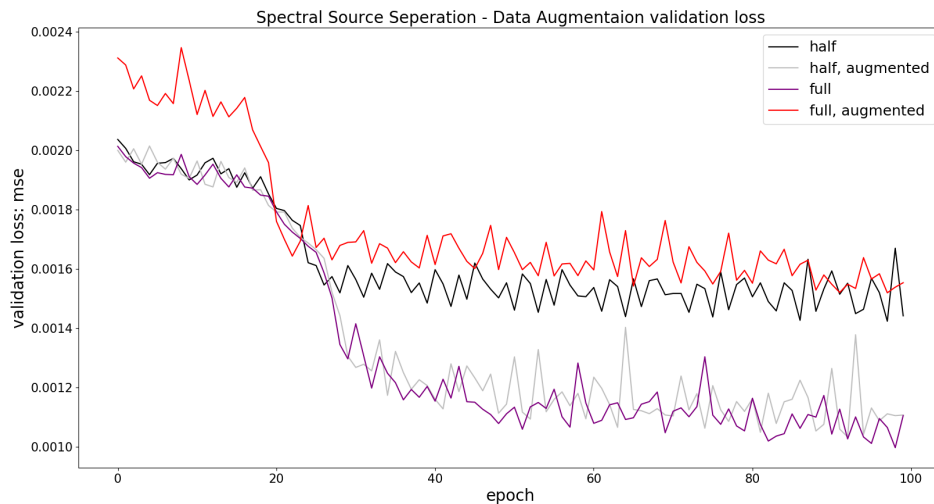


Figure 3.8: Validation loss (MSE) for networks trained on a different (part of the) dataset.

What we observe in Figure 3.8 is that our data augmentation did not benefit training using the full dataset. We see that training using half the dataset yields roughly the same results as training using the full dataset with augmented data. Additionally we also see that training using just the full dataset performs similarly to training using half the dataset with the augmented data. This is an interesting observation, showing us that adding our augmented data to the full dataset negatively impacts the performance of the model. We believe this is due to overfitting. The dynamically mixed audio segments don't provide additional spectral characteristics of the vocals in order to aid the separation. When using half of the dataset, the augmented data did improve training accuracy, because the augmented data was created using the entire dataset, thus providing unique (unseen) spectral characteristics of the target source to separate.

To conclude, the use of data augmentation in our experiments did not improve the validation loss when already using the full dataset. In further research we recommend mixing multiple target sources to encourage the learning of new spectral characteristics. Using less augmented data during training may also prevent overfitting, and the dynamic mixing may still benefit the model performance.

3.2.6 Depth-wise Separable Convolutions

In this section we look at depth-wise separable convolutions and how they compare to the use of standard convolutions. For both types of convolution we ran experiments using UNets at three different depths, running each experiment three times. Let us look at Table 3.5 showing the measurements done during the experiments, over the the extent of 60 epochs.

Table 3.5: Shows the minimum validation loss (MVL), average epoch runtime in seconds (AERT), *Parameter difference compared to normal convolution* (PDCNC) and *Percentage improvement over standard convolutions* (PIOSC) for each synthesis method.

depth, layer type	MVL	AERT	#PARAM	PDCNC	PIOPD
d3, conv2d	0.001059	370.86	156847	0.00%	0.00%
d3, depthsep	0.001212	628.09	51280	67.31%	-12.64%
d4, conv2d	0.001046	367.42	465071	0.00%	0.00%
d4, depthsep	0.001077	642.16	165776	64.35%	-2.86%
d5, conv2d	0.000890	365.90	1860399	0.00%	0.00%
d5, depthsep	0.000994	643.37	649360	65.10%	-10.44%

We can look at the #PARAM and PDCNC in Table 3.5 to see what the impact of using depth-wise separable convolution layers is on the number of parameters. We see that using depth-wise separable convolution to replace all the convolution layers in a network, reduces the number of parameters (#PARAM) by $\sim 65\%$.

When using depth-wise separable convolutions we expect a trade off between the number of parameters and the performance of the model. Analyzing the PIOSC shows that for a depths three and five, we have an $\sim 11\%$ decrease in the accuracy in the model. We also observe that with a depth of four, the model performance only decreased by $\sim 3\%$. We hypothesise that this large difference in performance impact is due to the random weight initialization of each experiment. This makes it difficult to draw an decisive conclusion and these results are influenced by many different factors. We

note two important reasons for why this might be. Either, weight initialization was unlucky and the models with standard convolution layers at depth four performed sub par, or the weight initialization for the models using depth-wise separable convolution layers show the possibility for depth-wise separable convolutions to perform almost as good as standard convolution.

$$\text{PIOSC} = \frac{MVL_{dn,conv2d}}{MVL_{dn,depthsep}} - 1 \quad (3.9)$$

Chapter 4

Related Work

4.1 TRU-Net

Choi et al. published their Tiny Recurrent UNet for the purpose of demodulating and speech enhancement [2]. They managed to achieve current state-of-the-art performance with only ~ 380000 parameters. After their model had been quantized, the total model size came down to 362 kilobytes, small enough to deploy on edge devices. In addition, they combined their model with a method called phase-aware β -sigmoid mask.

4.2 Multi Modal Audio Source Separation

The work of Lluís et al. explores a multi modal approach to audio source separation [25]. Their method takes both a spectrogram and a 3D point cloud as input, to predict a mask used for the source separation. They use two separate models for processing the different inputs, as well as a third model to combine the extracted features from both modalities.

4.3 MMDenseNet

MMDenseNet was published in a paper by Takahashi et al. in 2017. Their model is largely inspired by DenseNet [37], which achieved excellent results at the task of image classification. In their work they split the spectrogram input into N frequency bands, which they each process separately using an MDenseNet. By separating these bands they hope to better extract spectral

characteristics unique to the range of frequencies in each band. They later combine the output of processing each band and pass that through a final dense block to produce their source spectrogram as output.

4.4 Spleeter

Spleeter was created by Hennequin et al. as part of the popular music streaming service Deezer [14]. The model used for Spleeter is said to be constructed out of multiple UNets, trained to produce a soft mask, and using the L_1 -norm as loss function. Spleeter is one of the most popular tools for audio source separation for those that don't want to train their own models.

Chapter 5

Conclusions

During our research we have investigated the following topics in relation to the task of spectrogram based audio source separation: How do popular optimizers compare at the task? How do skip connections benefit task results when comparing AEs and UNets at different depths? How do direct synthesis and mask synthesis compare at the task? How can we use dynamic mixing as a data augmentation technique for this task? Finally, how do depth-wise separable convolutions compare to normal convolutions when performing the task?

We have examined a multitude of popular optimizers, and our results show that the RMSprop optimizer achieves the best minimum validation loss (MVL). When time is of the essence, using the Adam optimizer still achieves acceptable results.

Our experiments have shown that increasing the depth of an AE reduced network performance, but adding skip connections to make them UNets allows the model to leverage the better feature extraction of the deeper models. Thus, increasing the depth of UNets improved the MVL further and further.

When comparing spectrogram synthesis to mask synthesis we found that our measurements show that using a mask is always a better option. Using a Leaky ReLU or Swish mask improved training stability and decrease MVL by up to $\sim 7\%$.

The results for using dynamic mixing as a data augmentation method led us to the conclusion that we added too many augmented samples to our training data. Adding the augmented samples to the full dataset negatively impacted the MVL and likely caused the model to overfit sooner. We suggest using less augmented samples, and to experiment with mixing multiple target sources to generate more relevant spectral characteristics of the target source.

Finally, the experiments looking into depth-wise separable convolutions led us to the conclusion that using this type of convolutions reduced the number of parameters by $\sim 65\%$. The parameter reduction comes with a trade-off, increasing MVL by $\sim 3 - 11\%$.

In conclusion, our results suggest that to achieve the best results at spectrogram based audio source separation network architectures should adhere to the following: Use the RMSprop or Adam optimizer for training, deeper models perform better as long as skip connections are used correctly, synthesising a mask always better than directly predicting the target source spectrogram, dynamic mixing may be useful but don't add too many samples to the training data and using depth-wise separable is a useful technique to reduce model size for use on edge devices.

Bibliography

- [1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- [2] Hyeon-Seok Choi, Sungjin Park, Jie Hwan Lee, Hoon Heo, Dongsuk Jeon, and Kyogu Lee. Real-time denoising and dereverberation with tiny recurrent u-net, 2021.
- [3] CodeEmporium. Depthwise separable convolution - a faster convolution!, March 2018. <https://www.youtube.com/watch?v=T7o3xvJLuHk>.
- [4] Timothy Dozat. Incorporating nesterov momentum into adam, 2015. http://cs229.stanford.edu/proj2015/054_report.pdf.
- [5] DSPrelated. Spectral audio signal processing. https://www.dsprelated.com/freebooks/sasp/Constant_Overlap_Add_COLA_Cases.html.

- [6] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization, 2011. <https://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf>.
- [7] Ngoc Q. K. Duong, Emmanuel Vincent, and Rémi Gribonval. Under-determined reverberant audio source separation using a full-rank spatial covariance model. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(7):1830–1840, 2010.
- [8] Manilow Ethan, Seetharaman Prem, and Salamon Justin. What is source separation?, 2020. https://source-separation.github.io/tutorial/intro/src_sep_101.html.
- [9] Justin Salamon Ethan Manilow, Prem Seetharaman. Evaluation, 2020. <https://source-separation.github.io/tutorial/basics/evaluation.html>.
- [10] Derry FitzGerald, Antoine Liutkus, and Roland Badeau. Projection-based demixing of spatial audio. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24(9):1560–1572, 2016.
- [11] Derry FitzGerald, Antoine Liutkus, and Roland Badeau. Projet — spatial audio separation using projections. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 36–40, 2016.
- [12] Google. Tensorflow page explaining adamax, 2021. https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adamax.
- [13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.
- [14] Romain Hennequin, Anis Khlif, Felix Voituret, and Manuel Moussallam. Spleeter: A fast and state-of-the art music source separation tool with pre-trained models. Late-Breaking/Demo ISMIR 2019, November 2019. Deezer Research.

- [15] Geoffrey Hinton, Nitsh Srivastava, and Kevin Swersky. Neural networks for machine learning. http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.
- [16] Native Instruments. The native instruments stems page, providing the audio format, oct 2021. <https://www.stems-music.com/>.
- [17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.
- [18] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.
- [19] Filip Korzeniowski and Gerhard Widmer. A fully convolutional deep auditory model for musical chord recognition. *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP)*, Sep 2016.
- [20] Jonathan Le Roux, John R. Hershey, and Felix Weninger. Deep nmf for speech separation. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 66–70, 2015.
- [21] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [22] Antoine Liutkus, Derry Fitzgerald, and Roland Badeau. Cauchy non-negative matrix factorization. In *2015 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 1–5, 2015.
- [23] Antoine Liutkus, Derry Fitzgerald, and Zafar Rafii. Scalable audio separation with light kernel additive modelling. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 76–80, 2015.
- [24] Antoine Liutkus, Derry Fitzgerald, Zafar Rafii, Bryan Pardo, and Laurent Daudet. Kernel additive models for source separation. *IEEE Transactions on Signal Processing*, 62(16):4298–4310, 2014.

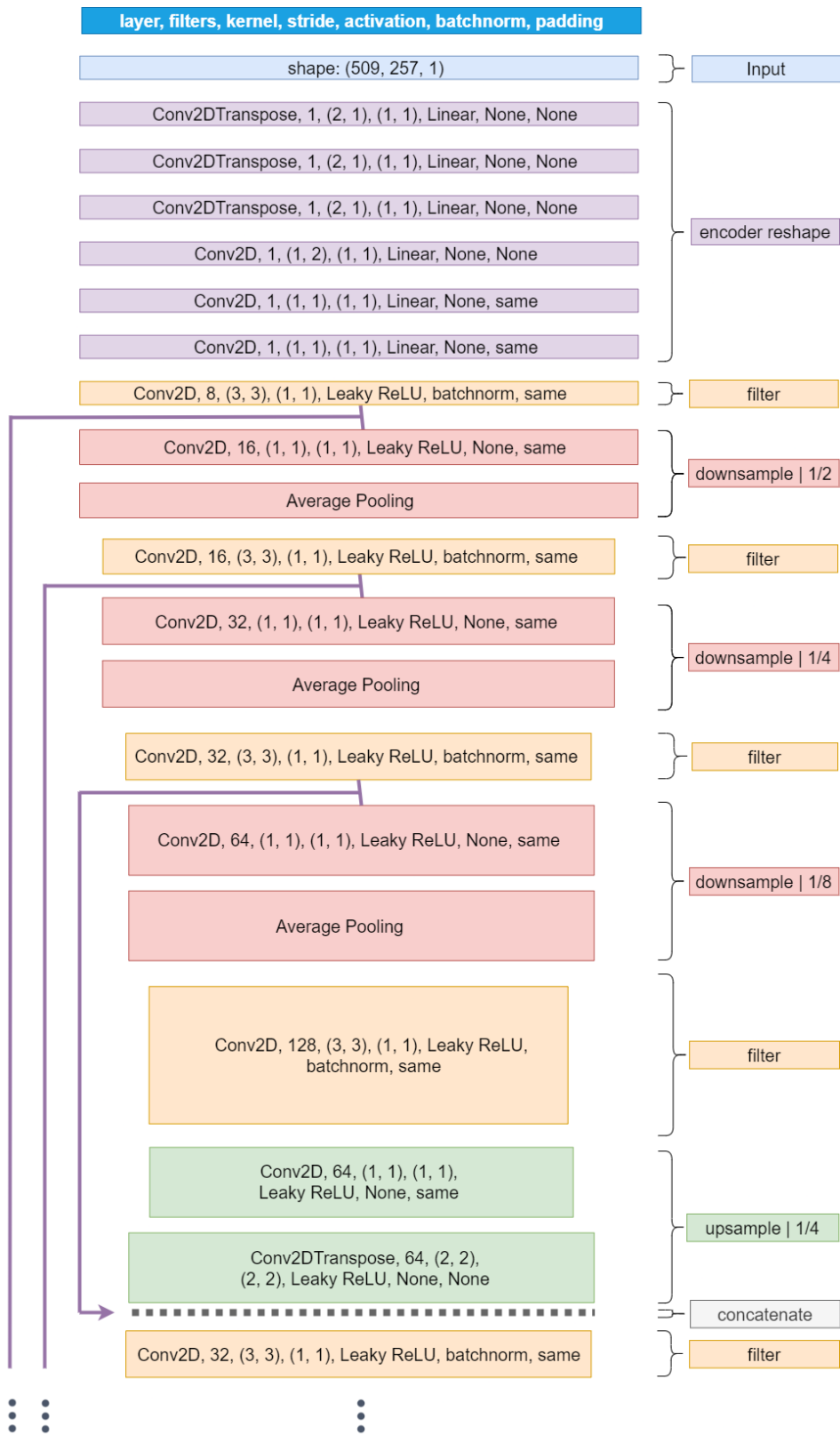
- [25] Francesc Lluís, Vasileios Chatziioannou, and Alex Hofmann. Music source separation conditioned on 3d point clouds, 2021.
- [26] Yuki Mitsufuji, Shoichi Koyama, and Hiroshi Saruwatari. Multichannel blind source separation based on non-negative tensor factorization in wavenumber domain. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 56–60, 2016.
- [27] Aditya Arie Nugraha, Antoine Liutkus, and Emmanuel Vincent. Multichannel audio source separation with deep neural networks. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24(9):1652–1664, 2016.
- [28] Alexey Ozerov and Cédric Févotte. Multichannel nonnegative matrix factorization in convolutive mixtures for audio source separation. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(3):550–563, 2010.
- [29] Zafar Rafii, Antoine Liutkus, Fabian-Robert Stöter, Stylianos Ioannis Mimilakis, and Rachel Bittner. The dataset MUSDB18 for music separation discription page, dec 2017.
- [30] Zafar Rafii, Antoine Liutkus, Fabian-Robert Stöter, Stylianos Ioannis Mimilakis, and Rachel Bittner. The MUSDB18 corpus for music separation, December 2017. <https://doi.org/10.5281/zenodo.1117372>.
- [31] Zafar Rafii, Antoine Liutkus, Fabian-Robert Stöter, Stylianos Ioannis Mimilakis, and Rachel Bittner. The MUSDB18 corpus for music separation, dec 2017.
- [32] Tom Sercu, Christian Puhersch, Brian Kingsbury, and Yann LeCun. Very deep multilingual convolutional neural networks for lvcsr. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4955–4959, 2016.
- [33] Daniel Stoller, Sebastian Ewert, and Simon Dixon. Wave-u-net: A multi-scale neural network for end-to-end audio source separation, 2018.

- [34] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. <http://proceedings.mlr.press/v28/sutskever13.pdf>.
- [35] Naoya Takahashi, Michael Gygli, Beat Pfister, and Luc Van Gool. Deep convolutional neural networks and data augmentation for acoustic event detection. 04 2016.
- [36] Naoya Takahashi, Michael Gygli, and Luc Van Gool. Aenet: Learning deep audio features for video analysis. *IEEE Transactions on Multimedia*, 20(3):513–524, 2018.
- [37] Naoya Takahashi and Yuki Mitsufuji. Multi-scale multi-band densenets for audio source separation, 2017.
- [38] Stefan Uhlich, Franck Giron, and Yuki Mitsufuji. Deep neural network based instrument extraction from music. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2135–2139, 2015.
- [39] Stefan Uhlich, Marcello Porcu, Franck Giron, Michael Enenkl, Thomas Kemp, Naoya Takahashi, and Yuki Mitsufuji. Improving music source separation based on deep neural networks through data augmentation and network blending. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 261–265, 2017.
- [40] Matthew D. Zeiler. Adadelata: An adaptive learning rate method, Dec 2012.

Appendix A

Appendix

In Figure A.1 we show the details of the network we used. It is known as a UNet, because when shown from left to right with horizontal skip connections, it takes on the shape of a U. When the skip connections are left out, we have an architecture known as an Auto-Encoder.



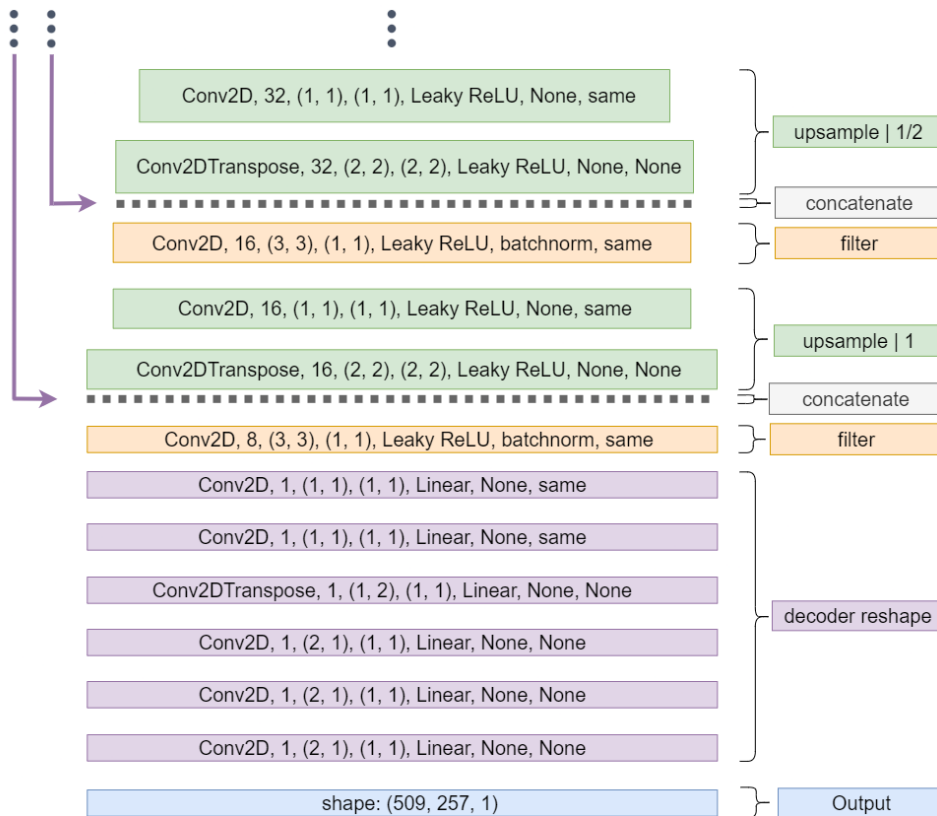


Figure A.1: The UNet architecture that we use during our research.

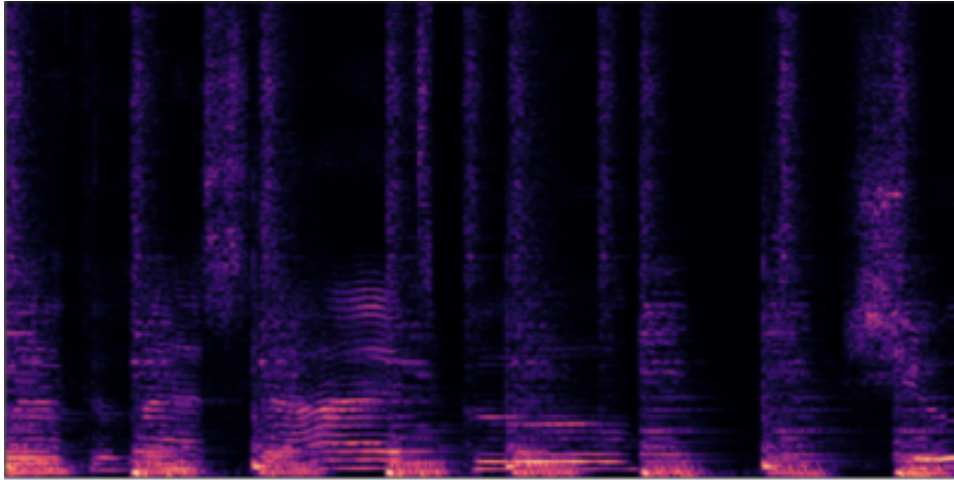


Figure A.2: Spectrogram of a segment of mixed audio sources.

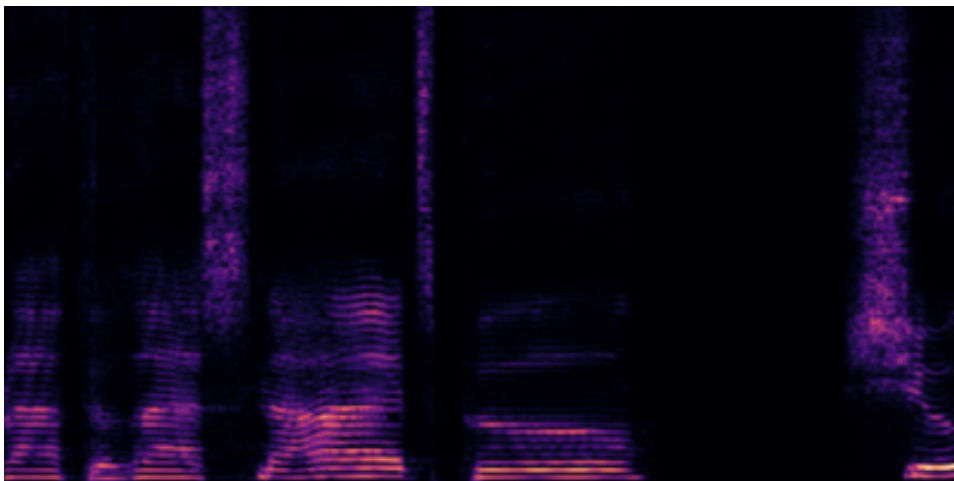


Figure A.3: Spectrogram of a segment from vocal audio.