Abstract

The aim of this thesis is to separate a melodic instrument from a mix through a Blind Source Separation (BSS) approach using neural networks and builds on another thesis from 2019 called *Vocal Extraction from Music using Deep Learning* by Flury, Kaufmann and Müller. In contrast to the previous thesis, which focused on vocal track separation, this work explores the application of this approach to the problem of separating melodic instruments from a mix.

This thesis focuses on the piano and the guitar. It was found that the approach examined in this thesis seems applicable, but it can not be used on both instruments without adjustments. Due to limitations in publicly available datasets, a new private dataset of guitar and piano tracks had to be created. Audio stems had to be gathered, labelled and mixed into two tracks, one containing the instrument to be separated and one containing all other instruments. This had to be done for both instruments. The new dataset is built from MedleyDB, MedleyDB 2.0, tracks from the rhythm games Rockband and Guitar Hero as well as the music mixing website Cambridge Music Technology (CMT).

A network structure called Stacked Hourglass was used. The main difference between the trainings is the amount of pre-processing done on the training datasets, the size of the datasets and the ratio used when mixing datasets. The evaluation of the trainings was done using *mir_eval* and the three commonly used measures Source-to-Distortion Ratio (SDR), Source-to-Interference Ratio (SIR) and Source-to-Artifact Ratio (SAR).

The best achieved SDR score was 0.73 for guitar by using as big a training dataset as possible. For piano an SDR score of -7.28 was achieved by enhancing a much smaller dataset with silence adjustments and adding solo piano playing to the dataset. Comparing trainings on different instruments with similar datasets shows that with all other parameters being the same, results still differ greatly. In general though, a larger dataset will achieve better results than other attempts to enhance training sets.

A simple web demonstrator was created to allow easy access to the trained models. All source code created for this thesis has been publicly released in lines with Open-Source Software (OSS) principles on GitHub.

Acknowledgments

We would like to thank Dr. Martin Loeser and Prof. Dr. Matthias Rosenthal for the advice and feedback they gave us, as well as the time invested into our weekly meetings.

We want to also show our appreciation to the Institute of Embedded Systems (InES) at ZHAW for providing the servers to run our training network on.

Contents

1 Introduction

This thesis aims to separate an instrument audio signal from a mix without any further information about the composition using neural networks. Our work builds on a previous thesis [1] that separated vocals from a mix using the same approach. Our thesis provides the necessary theoretical foundations to readers without prior experience in audio processing or machine learning to understand the core ideas expressed in it.

1.1 Motivation

The motivation for this work was to find out if separating an instrument audio signal from a complete mix can be done using a deep learning network previously designed for vocal extraction. This includes the exploration of the necessary changes to the network structure for it to generate good results as compared to other approaches.

Another motivation is investigating the difference between instruments when running them through the same approach. Can similar results be achieved with enough training or does the network structure need to be adjusted for each instrument?

The application of such a separation approach could have benefits for musicians, since practicing or analyzing a piece of music can be made easier by excluding or isolating some instruments.

1.2 Goals

The first goal of this work is to examine the work of the previous thesis [1] and use it to separate a melodic instrument from a mix. To achieve this, it is necessary to gather significant amounts of tracks and organize them in a suitable structure for the neural network. This includes checking the overall quality of each track and its fitness for this task.

The second goal is to evaluate the results of the neural network through automatically calculatable metrics and define changes to the datasets, the neural network structure or configuration to improve the results if needed. Another part of this step is to check if such a neural network can handle the different instruments or if there are inhererent problems with one of them.

Additionally, a web application that makes the results available in a tangible and easy to use way will be developed. The web application is intended to take a mix as an input and seperate a defined instrument from the rest of the mix.

Finally, all source code used for creating datasets, training and the web application will be made publicly available in line with OSS principles.

2 Theoretical Foundations

This section is intended for readers without in-depth knowledge about signal processing or neural networks and covers the fundamentals needed for this thesis. Readers who are familiar with these topics can skip to section 3.

2.1 Audio and its Digital Representation

A good audio separation approach needs to be able to represent audio in a way that allows for easy separation. This means a proper understanding of sound and its digital representation is required. The sounds we as human beings hear are vibrations in some medium, usually air, that are perceived by our ears and then interpreted by our brains. These vibrations can be described as a combination of sinusoidal waveforms, with their amplitude corresponding to the loudness of the sound and their frequency corresponding to the pitch. Since these waveforms are continous functions, they can not be fully and accurately stored in a discrete digital form.

2.1.1 Digital Waveforms

For an acoustic signal to be represented digitally, the air pressure changes are recorded with a microphone and converted to an electrical signal. The voltage of this signal is then sampled and quantized. The amount of these samples per second is called the sample rate of a digital audio signal and is measured in hertz (Hz). These raw audio samples can then be further transformed into more usable representations.

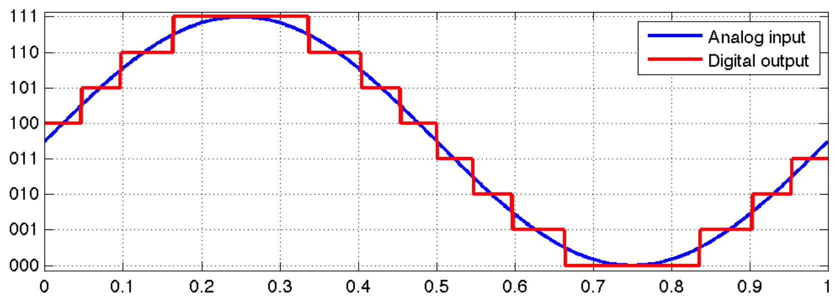


Figure 1: Converting an analog input to a digital output [2]

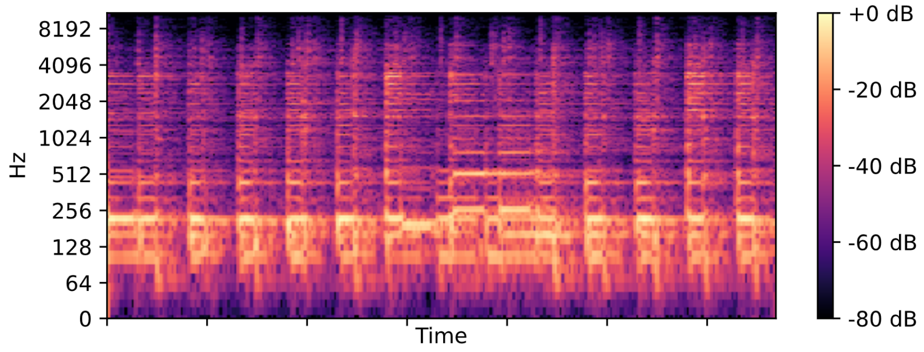


Figure 2: An example spectrogram

2.1.2 Spectrogram

The most common type of representation are time-frequency representations like a Short-Time Fourier Transform (STFT). A STFT is calculated from a waveform representation by computing a Discrete Fourier Transform (DFT) of a small time window sliding across the signal. Different window sizes influence the amount of frequency bins in the end result. The amount of frequency bins is equal to $\frac{window\ size}{2} + 1$. The result is represented as a heatmap with time along the horizontal axis and frequency along the vertical axis. Each position has a color which shows the amplitude of the signal at that particular time and frequency, representing loudness as measured in decibel (dB).

The phase of a signal involves the relationship between the position of the amplitude crests and troughs of two waveforms. It is difficult to model the phase. Therefore most audio separation approaches only operate on some variant of a spectrogram that does not represent the phase in each time-frequency position.

One type of visualization is the log spectrogram. Human hearing or rather the decibel scale is logartithmic base 10 scale. [3] The log spectrogram is calculated by taking the log of the absolute value of each element in the STFT.

2.1.3 Blind Source Separation

Blind Source Separation (BSS), also known as Blind Signal Separation, describes the task of separating a set of mixed signals into a set of source signals, without the use of information about the mixing process or the source signals themselves. The objective is to recover the original source signals from a mix of signals. Methods for solving BSS usually try to reduce the set of potential solutions as much as possible without excluding the actual expected result. Generally, this is done using some form of machine learning.

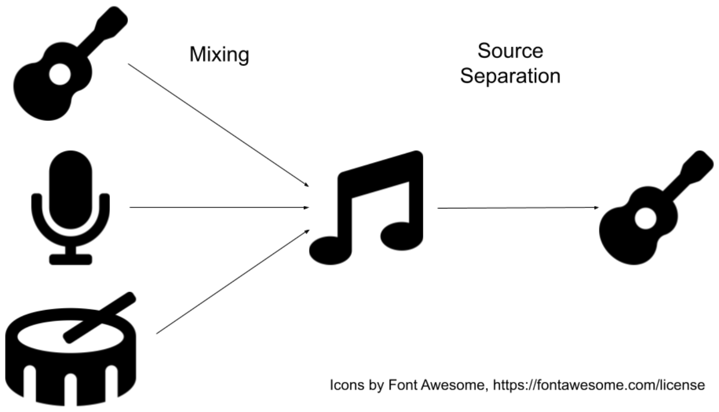


Figure 3: The process of mixing and BSS

To extract a source from a mix of signals, a so called mask is generated. In general, a mask defines which parts of a signal should be used and which parts should be thrown away. In terms of audio processing, this usually means a mask is applied to its frequencies to extract the instrument producing sound in that frequency range. When inverted, such a mask will remove that instrument from the resulting mix. This process is not foolproof though, since multiple instruments can share the same frequency ranges or overlap at the edges.

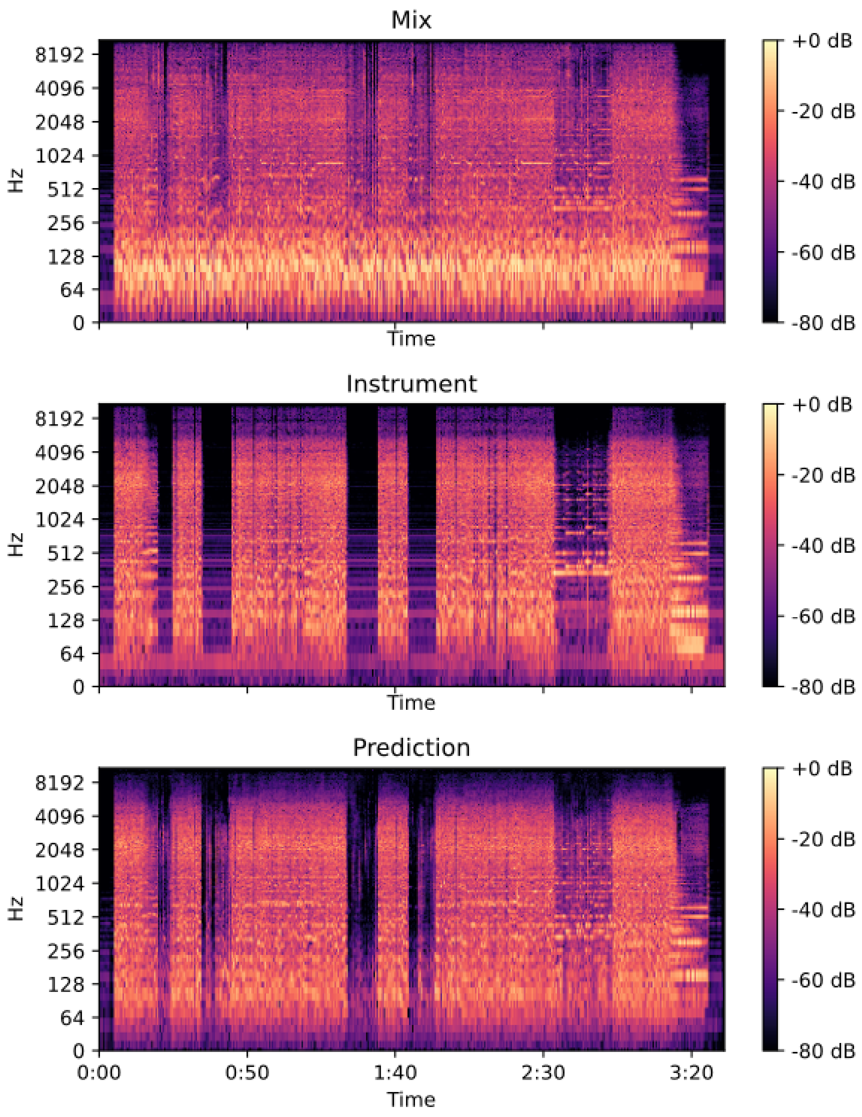


Figure 4: An example of an instrument spectrogram in relation to the full mix and the predicted instrument spectrogram

2.2 Machine Learning

2.2.1 Deep Learning

Deep learning is a method of machine learning based on artificial neural networks. Deep learning architectures such as deep neural networks and Convolutional Neural Network (CNN)s have been applied to many different topics, like computer vision, speech recognition as well as music recognition.

2.2.2 Neural Network

At the base of neural networks lies the concept of the neuron as seen in figure 5. Inspired by biological neurons, these artificial neurons accept multiple inputs, each influenced by a weight, which are then summed up and passed through an activation function, with the result being the output of the neuron. A neural network in general consists of multiple layers and each layer consists of multiple neurons, as seen in figure 6.

The activation function of a neuron defines the output of that neuron. There are an incredible amount of activation functions. We will only discuss those two activation functions used in the network we use in our thesis. The simplest function is the linear activation $f(x) = x$. It just takes the input and returns it as the output. Some non-linear activation functions take the input and if it is larger than their internal limit, the output is set to that limit. This has the issue that it could saturate the network at those limits. Thus a hybrid function $f(x) = max(x, 0)$ called Rectified Linear Unit (ReLU) is often used. ReLU returns the input if it is larger than zero. If it is not, then zero is returned.

When running a calculation in a network, an input vector containing the source data is passed to the input layer and then fed forward through the network onto the last layer, also called the output layer, where the result can be

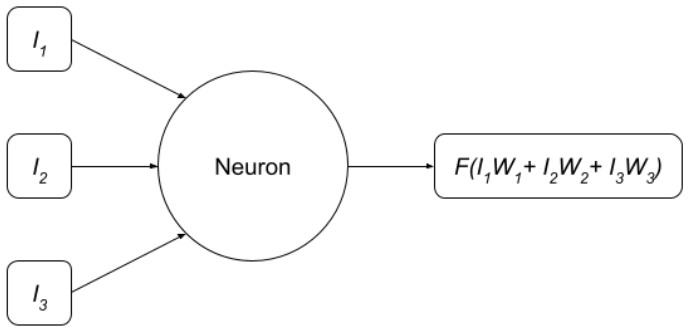


Figure 5: A basic neuron with inputs I, weights W and activation F

interpreted. A lot of network structures, different activation functions, specialized neurons and layers exist to enable neural networks to solve many varied problems. Some of those relevant to this thesis are explained in section 2.2.3.

The trick in using neural networks is that its weights can be trained automatically. Using a dataset where the expected output to a given input is known, the result of the network can be analyzed using a loss function. Depending on how well the network fits, the weights in the network can be adjusted through time. Using figure 4 as an example, the full mix spectrogram would be the input and the instrument spectrogram would be the expected output. The prediction spectrogram would then be the result of the network after training.

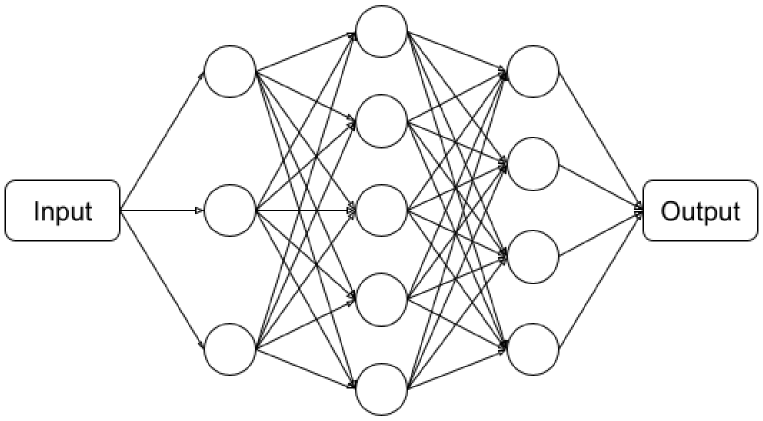


Figure 6: A simple example neural network

2.2.3 Convolutional Neural Network

The name Convolutional Neural Network indicates that the network employs a mathematical operation called convolution. It involves the multiplication of a set of weights with the input. The multiplication is performed between an array of input data and an n-dimensional array of weights, called a filter. [4]

A filter is an array of weights, smaller than the input data. The multiplication is made with a slice of the input data with the same size as the filter and the filter itself. The calculation itself is a scalar product. This filter-sized window moves with a specified stride size over the data and the calculation is applied with each step. The result from multiplying the filter with the filter-sized input one time is a single value. As the window steps over the input data and the filter is applied each time, the result is an array of values that is called a feature map. See figure 7, for an example application of a filter.

An example of a filter in image processing would be a edge detection filter as shown in figure 8. An edge in an image can be described as a sudden change in luminosity. When looking at the filter in the middle, we can see that the resulting value of applying it will be the sum of values to the right of its center subtracted from the sum of values to the left of its center. If the filter is applied to an area of solid color, its resulting value will be 0. With increasing difference between the colors to its left and right, the resulting value will move away from 0. Thus, when the filter is applied to the entire image with a stride of 1, the result is a feature map showing all the vertical edges.

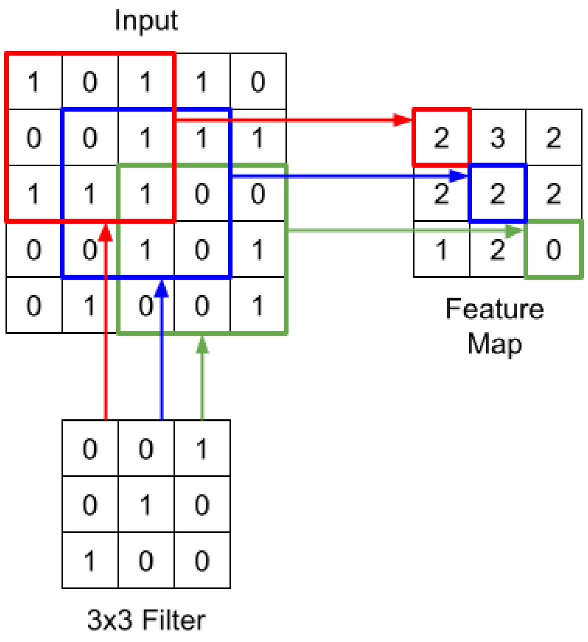


Figure 7: A single filter applied to a 5x5 input with stride 1

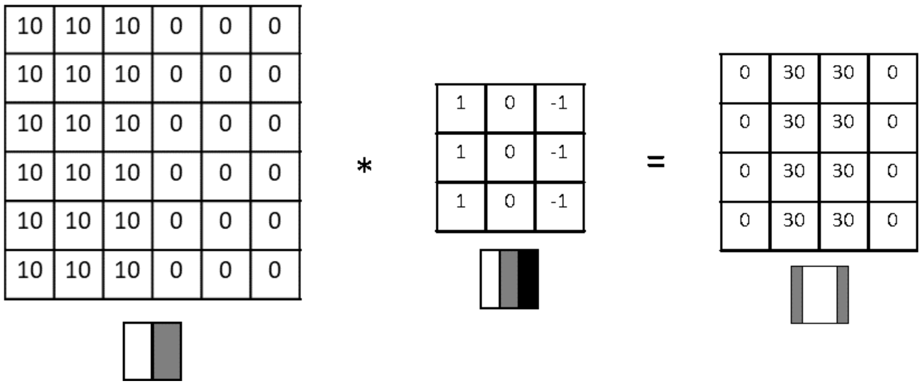


Figure 8: An example of a filter for vertical edge detection. [5]

A limitation of feature maps is that the result of them can vastly change if the input is slightly different. A common approach to address this problem is called downsampling and originates from signal processing. [6, pp. 285–289] Downsampling is the use of a lower resolution version of an input signal that still contains its important structural elements, without the finer details. Downsampling in CNNs can be achieved by using a pooling layer. [7]

A pooling layer is usually added after a convolutional layer and uses a similar operation where a window is moved around the input and a value is calculated. There are two common functions used as pooling operations, Maximum Pooling and Average Pooling. Maximum Pooling calculates the maximum for each patch of the feature map, while Average Pooling calculates the average value for each patch. The pooling layer always reduces the size of the feature map. For example, if the pooling operation is the size of 2x2 pixels and the stride is 2 pixels, the dimensions of the feature map will be halved. For instance, a 6x6 (36 pixels) feature map would be reduced to a 3x3 (9 pixels) feature map. The result of these operations can be described as a summarized version of the features detected in the input. This helps with the problem of small changes in the input data resulting in big changes in outputs by having a pooled feature map that can still detect a feature even if it has moved.

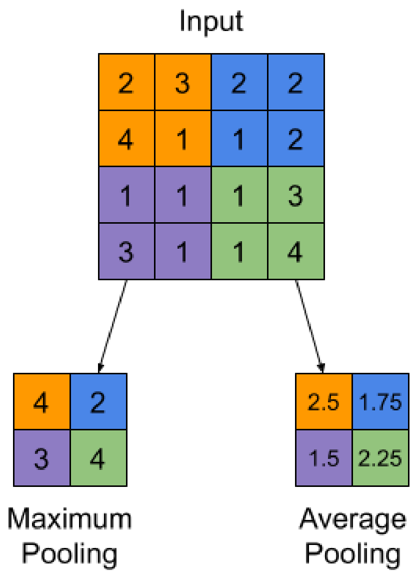


Figure 9: An example of 2x2 pooling

Large neural networks trained on small datasets can overfit the training data, meaning that they learn to work on the provided training data only, which leads to poor results when new data is introduced. To mitigate overfitting, a regularization method called dropout is often used. In a dropout layer some nodes will randomly be ignored or "dropped out" during training by ignoring all incoming and outgoing connections. Figure 10 shows an example of a neural network with dropout added to the middle layer. This has the effect of the next layer being treated like it has a different number of nodes and connectivity to the prior layer. Since the nodes that are affected are chosen at random to fit the dropout rate, this has the effect of making the training process non-deterministic, forcing nodes within a layer to randomly take on more or less responsibility in handling data passing through it. Dropout therefore helps the network learn to generalize on training data. [8]

Another method that helps with the optimization of a Convolutional Neural Network (CNN) is batch normalization. While the exact reason for its results are not yet known [9], it provides an elegant way of re-centering and re-scaling data in neural networks. If the CNN does not have to keep coordinating updates with ever changing parameters from the previous layer, it significantly improves the performance of the CNN. It does this by normalizing layer inputs by scaling them for each mini-batch. A mini-batch is simply a further split of a batch of input data into smaller parts. This all leads to the assumptions about the spread and the distribution being made by the subsequent layers to not change dramatically. This has the effect of stabilizing and improving the training process. [10]

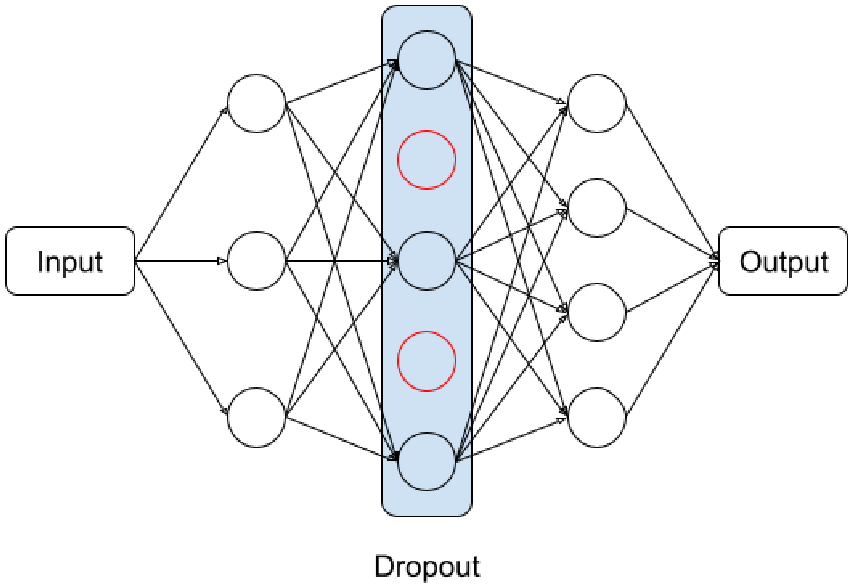


Figure 10: Figure 6 with an added dropout layer with dropout ratio of 0.4

3 Concept

3.1 State of the Art

When looking at state of the art solutions [11] to in the field of BSS we can see that many different approaches exist, with the best one changing several times throughout the years. The field is highly active and improving constantly, as can be seen by looking at the submission years of different solutions. At time of this thesis, the top three solutions all were submitted in the previous year of 2020. The majority of the other solutions were released in 2019 and none are from before 2018. The approaches differ in many aspects, but generally we can group them by the way they represent audio and their network structure. In table 1 we list the currently highest ranking models and present some of their properties for a quick overview. For each approach we note if they work on a STFT spectrogram or on the audio waveform directly, and if they use CNNs or Recurrent Neural Networks (RNNs)[1].

	D3Net [12]	**LaSAFT** [13]	**MMDenseLSTM** [14]	**Demucs** [15]	**U-Net / Spleeter** [16] [2]
STFT	X	X	X	-	X
Waveform	-	-	-	X	-
CNN	X	X	-	X	X
RNN	-	-	X	X	-
	Conv-TasNet [17]	**Meta-TasNet** [18]	**UMX** [19]	**Wavenet** [20]	**Wave-U-Net** [21]
STFT	-	-	X	-	-
Waveform	X	X	-	X	X
CNN	X	X	-	X	X
RNN	-	-	X	-	-

Table 1: A comparison of BSS approaches

In contrast to the state of the art solutions, our thesis is interested in the separation of different instruments less commonly used and the performance of an existing approach to these instruments. This requires a great focus on creating a fitting dataset, as well as adjusting and configuring an existing solution to this dataset.

3.2 Neural Network

Since this work is based on a previous thesis that seperated vocals from a mix also using neural networks, taking a similar approach made sense. Their work had the best results with a CNN structure called Stacked Hourglass. Using the structure and configurations as described in *Vocal Extraction from Music using*

[1]Another type of neural networks not further explained in this thesis
[2]Spleeter is a collection of pre-trained U-Net models

Deep Learning [1, p. 26] as a jumping off point, we plan to discover the necessary changes to achieve good results for separating melodic instruments from a mix through a BSS approach.

3.3 Data

In order to be able to properly train a neural network we need to collect music tracks to build training datasets. We are planning to separate piano and guitar tracks from a mix, so our training data needs to contain those instruments. As there are only very few datasets already publicly available with those instruments separated, a new dataset for our purposes will need to be created. The data needs to have a diversity in their genres and playing styles and be available as seperated stems. Once all the data is collected it will need to be mixed into a fitting format for the neural network, that being one stem containing only audio from the instrument we plan to separate and another stem containing all other instruments mixed together. To evaluate the results of the neural network, a separate dataset of tracks used for validation needs to be created.

To ensure that the dataset we will build is fitting, a tool that supports checking the quality of tracks will need to be created. With this tool it should be possible to easily listen through a list of tracks and determine if the track meets the quality requirements. After the tracks have been checked, they are ready to be used for trainings with the neural network.

4 Training data

4.1 Data Collection

There exist several datasets commonly used in Music Information Retrieval (MIR) and BSS, like musdb18 [22] or dsd100 [23]. While these datasets contain well labeled data, the stems contained in them are only separated into vocals, drums and bass, with all other instruments being grouped together into a singular stem. Without access to stems of the melodic instruments we intend to separate from their respective mixes, these datasets proved insufficent for the purposes of this thesis. Optimally, a dataset would provide a single stem for each unique instrument in a mix as a WAVE file, although different file formats are easily transformed into WAVE files.

The only datasets we found that fulfill these requirements exactly are MedleyDB [24] and MedleyDB 2.0 [25]. Both of these dataset are organized in the same way, with a folder for each track containing all the stems as WAVE files, and a separate folder containing all metadata for the tracks in the form of YAML documents. The metadata provides information on which files contain what instrument. The combination of both datasets contains a total of 196 tracks.

Slakh2100-redux [26] is another dataset that does provide well defined stems for each single instrument, but the stems are provided as FLAC files instead of WAVE. The speciality of this dataset is that the stems are not real instrument recordings, but rendered Musical Instrument Digital Interface (MIDI) files. While this provides a wealth of tracks that are perfectly separated, the quality of stems vary greatly depending on the current capability of synthesizing any given instrument. At the time of writing, piano for instance can be synthezised quite well, while guitars are severly lacking. All stems of a track are grouped together in a single folder. Each folder additionally contains metadata about the track in a YAML document. The metadata provides information on the MIDI patches used to render the different stems, as well as data like the integrated loudness of a stem or the overall gain of a track. The dataset contains a total of 1'710 tracks.

Another class of datasets are tracks that contain stems for single instruments, but no metadata. This includes tracks from the rhythm games Rockband and Guitar Hero [27], as well as tracks from the music mixing website Cambridge Music Technology (CMT) [28]. Many Rockband and Guitar Hero stems only contain separate stems for guitar and bass, especially those from earlier iterations of the series. The stems from CMT on the other hand could be considered too detailed for our purposes, since they are recordings used for mixing practice. This means there usually exist multiple stems for the different parts of the drums, different guitar or vocal tracks as well as all kinds of other audio stems. The difficulty with these datasets is the additional work needed to classify the different instrument stems and mixing them together. The dataset built from CMT contains 141 tracks, and the dataset built from Rockband and Guitar Hero tracks contains 862 tracks. This results in a dataset with a total of 1'003 tracks.

The last class of datasets we used consists of instruments playing alone. These can be used to train the network to recognize an instrument with no accompaniment. There exists a wealth of these on platforms like YouTube, often compiled into long collections of several tracks one after the other. We downloaded these and split them into smaller pieces of music, usually three minutes long, that can be more easily worked with. We did not follow any pauses or track lengths when splitting the longer collection, since that information is not relevant for us.

We created a set of pure piano tracks from two YouTube videos. A video of piano covers of Disney songs [29], with a length of about 3 hours, and a video of background music consisting only of piano [30], with a length of about 1 hour. The songs were split into 3 minute long segments, resulting in 80 pieces of music to be used.

We also created a set of pure guitar tracks from five YouTube videos. One video with a mix of relaxing acoustic guitar sounds [31], with a length of about 70 minutes, a video with about one hour of electric guitar solos [32], and three videos of Foo Fighters[3] frontman Dave Grohl playing various styles of electric guitar [33] [34] [35]. These videos result in 68 pieces of audio at 3 minutes each to be used for training.

4.2 Data Organization

We store multiple iterations of our datasets to ensure we always have the possibility to use them in a different way. The first version are the untouched datasets, with each contained in a single folder bearing its name.

From there, the second iteration is created by using metadata and manual classification to group all stems into instrument classes, specifically *guitar*, *bass*, *drums*, *percussion*, *piano*, *vocals* and *other*. Two special classes, called *mix* and *excluded*, also exist. *mix* is used for pre-mixed tracks provided by some datasets as an example. *excluded* is used for stems that should not be used further, like metronomes, reference tracks or room recordings. In this iteration, all tracks still have their own folder, but they are no longer grouped by their datasets. A track contains a folder each for the different instrument classes for which it has stems.

[3]a contemporary rock band

The final iteration is used to train the network in the end, so a more involved process is necessary to create it. The aim of this iteration is to have pairs of stems, one containing the audio of the instrument that should be separated, the other containing all other audio. Since training data is used for a single instrument, multiple versions of this iteration may exist. To reduce filesize, and in turn speed up processing times, these datasets can be generated with multiple sample rates. The top most level contains a folder for each of the generated sample rates. One level lower, a folder exists for each track from the previous iteration, containing exactly the two WAVE files making up the stem pair. The file containing the target instrument has the prefix *instrument_*, while the other has the prefix *rest_*, each followed by the track name.

4.3 Data Preparation

4.3.1 Quality Assurance

To ensure stems used in training are of appropriate quality, a tool to quickly skim through different parts of an audio file was created. For each track folder, all stems of the chosen instrument are split into parts. By default, we used 5 splits. All splits are played one after the other, with the possibilty to skip to the next one. Different labels can be set once a problem is discovered with a stem.

The following labels are used in our tool:

- *missing*
  - Used when there seems to be no data at all in the stem
- *bleed*
  - Used when other sounds can be heard next to the main sound source
- *bad_quality*
  - Used when the stem is of unusable quality, be that from a bad recording or file quality
- *inaudible*
  - Used when the content of the stem can not be heard without further enhancements
- *mislabeled*
  - Used when the main sound source of the stem is not what it was labeled as
- *detailed*

    - Used when the stem needs a more detailed analysis, either by checking more splits or using some other specialized tool

- *checked*
    - Used as a general marker that the quality tool was ran on a track
    - Set automatically by the quality tool

Each mark is created as an empty file with a name in the form of *instrument_mark*. For instance, the existence of the two files *guitar_bleed* and *guitar_checked* would indicate the guitar stems of a mix were checked for their quality and found to contain bleed from some other instruments.

4.3.2 Mixing

To create training data from loose stems, these stems need to be mixed into two audio files, one containing only audio from an instrument we wish to separate from a complete music track, and the other containing everything but the chosen instrument. For this purpose, stems are grouped into either *instrument* stems or *rest* stems, with all stems of the same group getting mixed together in equal proportion. The exception to this are stems labeled as *mix* or *excluded*. None of those stems are mixed into any of the two groups to keep the final stems clean and well separated.

The two mixes are then rendered to WAVE files with a sample rate of 44'100 Hz. Additional versions of the mixes with differing sample rates can be rendered to create smaller files that can more quickly be transferred and read. For this, a version with a sample rate of 22'050 Hz is normally created.

4.3.3 Silence

Especially with piano tracks it was discovered that often times the instrument mix would be mostly silent, with only intros, outros or bridges containing any actual piano playing.

To ensure that all training data has a good amount of audio in it, a tool was written to detect contigous silence and remove it to increase the ratio of audio to silence up to some defined threshold. For the purpose of chunks of tracks, silence has been defined as any stretch of audio longer than one second with -50 dB relative to full scale (dBFS). Shorter lengths of silence were not considered because we did not want to remove pauses in playing that might be musically relevant. Silences are cut in order of their length so that longer silent parts are cut first, since they are probably not relevant to the instrument in question.

This tool is ran only on the two final audio files instead of all stems to save on time and performance necessary for re-rendering and re-mixing. Any track that did not reach the intended silence ratio after adjustement, for instance because it contains many short silences, is still rendered unless it falls below an optional cutoff ratio where tracks are considered unusable. See section 4.4 for exact values used. See appendix B.1 for experimental trainings with different silence ratios.

4.4 Training Datasets

Several different datasets were created using the approach outlined in section 4.2. These datasets differ in the source datasets they pull from, the targeted instruments as well as the processing applied to them. Where not noted otherwise, all data sources were checked for their quality by hand as described in section 4.3.1. The difference in track counts between deriving datasets are due to not having enough audio data in them after processing.

1. Recorded guitar
   - Created from MedleyDB, Rockband and Guitar Hero datasets
   - 44'100 Hz and 22'050 Hz
   - 709 tracks in total
   - 116 tracks split off for validation set
   - 593 tracks used for training
2. Recorded guitar (silence adjusted)
   - Created from recorded guitar training dataset
   - 22'050 Hz
   - Tracks with audio to silence ratio below 70% adjusted to 70% or above
   - Tracks with audio to silence ratio below 50% after adjustement were discarded
   - 592 tracks used for training
3. Recorded piano
   - Created from MedleyDB and CMT datasets
   - 44'100 Hz and 22'050 Hz
   - 153 tracks in total
   - 22 tracks split off for validation set
   - 131 tracks used for training

4. Recorded piano (silence adjusted)
    - Created from recorded piano training dataset
    - 22’050 Hz
    - Tracks with audio to silence ratio below 70% adjusted to 70% or above
    - Tracks with audio to silence ratio below 50% after adjustement were discarded
    - 123 tracks used for training
5. Synthesized piano
    - Created from Slakh2100-redux dataset
    - 44’100 Hz and 22’050 Hz
    - Limited to 300 tracks
    - Not quality checked
6. Pure guitar
    - Created from pure guitar dataset
    - 22’050 Hz
    - 68 tracks use for training
    - Not quality checked
7. Pure piano
    - Created from pure piano dataset
    - 22’050 Hz
    - 80 tracks used for training
    - Not quality checked

For exact statistics of datasets before and after silence adjustements, see appendix A.

4.5 Validation Datasets

To evaluate trainings results, validation datasets were created. These are tracks that the network never gets to train on and are only used to evaluate how well a trained network performs. The tracks in these datasets were chosen by hand from the training datasets with a focus on having a broad spectrum of genres and playing styles.

- Guitar
  - Created from training data set 1
  - 116 tracks
- Piano
  - Created from training data set 3
  - 22 tracks

5 Network

5.1 Structure

We use a network structure known as a Stacked Hourglass as described in "Music Source Separation Using Stacked Hourglass Networks" [36] and as implemented and optimized in the preceding thesis [1, p. 24]. This network structure can be roughly split into three parts, which are the initial convolutions, the hourglass modules and the final convolutions. In the original paper, multiple hourglass modules are used, but as noted in *Vocal Extraction from Music using Deep Learning* a single module is sufficent when separating a single source.

An array of spectrogram chunks with a size of $H \times W \times C$ is used as the input to the network. H is equal to the the amount of frequency bins in the STFT, which is calculated as $\frac{window\ size}{2} + 1$. W is equal to the width of one spectrogram chunk. C is equal to the amount of tracks to separate.

Five initial convolutions are used, with 64 7x7 filters for the first layer, 128 3x3 filters for the middle three layers and 256 3x3 filters for the last layer. Each convolutional layer is also followed by a batch normalization layer and a dropout layer.

An hourglass module consists of multiple so-called stacks, with each stack consisting of a downsampling step and a corresponding upsampling step. The number of these stacks can be adjusted. The downsampling step consists of a 2x2 max pooling layer, a convolutional layer with 256 3x3 filters, a batch normalization layer and a dropout layer. The upsampling step consists of a convolutional layer with 256 3x3 filters, a batch normalization layer, a dropout layer and a 2x2 upsampling layer. The downsampling steps are connected to the next lower downsampling step directly. Additionally, they are connected to their upsampling step by a skip connection consisting of a convolutional layer, a batch normalization layer and a dropout layer. In each stack the size of the feature maps first gets reduced by half due to the pooling layers and afterwards gets

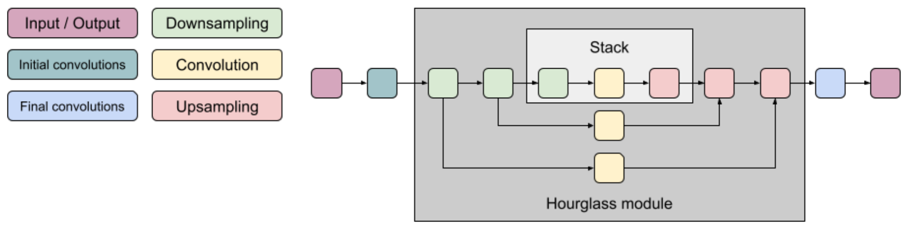


Figure 11: Diagram of the Stacked Hourglass structure with one hourglass module containing 3 stacks

doubled again in the upsampling layers. The multiple down- and up-sampling steps connected by skip connections allows the network to work on multiple resolutions of the signal, greatly improving results.

At the end of the network three further convolutional layers are added, with 256 3x3 filters for the first layer, 256 1x1 filters for the second layer and a single 1x1 filter for the last one, since we only work with mono audio at this point. Dropout layers are applied to the first two layers and a batch normalization layer is applied to the first layer only.

All dropout layers in the network have a dropout rate of 0.2. All convolutional layers use a ReLU activation, except for the last one in the hourglass module, which uses a linear activiation.

The output of the network is a mask as described in section 2.1.3. This mask can then be applied to the input to the network, once as is and once inverted. This results in two tracks, one containing only the predicted separated instrument and the other containing the rest of the input.

5.2 Optimizations

The implementation was previously used for vocal extraction, so we experimented with some parameters to evaluate their influence on the results when used to extract instruments from mixes. While most optimization was done in the training data, we did spend some time on deciding on the best Fast Fourier Transform (FFT) window length and amounts of filter to use in the hourglass module.

FFT windows of 512, 1'024, 1'536 and 2'048 samples were tested, with 1'536 being the length used in the preceding thesis. [1, p. 29] While the SDR of the 1536 length trainings is actually the lowest, due to the minor difference to the scores of the other lengths and the better subjective quality of example separations, we decided to keep using 1'536 samples as our FFT window lengths.

To speed up training, different amounts of filters in the hourglass module were tested. The original implementation used 256 filters, resulting in a network with 12'546'689 trainable parameters. We found that 192 filters actually produced slightly better results and reduced training time by about a sixth. The network then only had 7'713'857 trainable parameters.

The trainings and results used in experimentation and evaluation of the most fitting network structure and settings are documented in appendix B.

5.3 Final Structure and Settings

The network we decided to use is a CNN using a Stacked Hourglass structure as described in section 5, with a single hourglass module with 6 stacks and 192 filters for the convolutional layers. L1 is used as a loss function, only slightly adjusted to work with the predicted mask instead of directly comparing the output of the network to the expected value. A validation rate of 0.2 was used during training. Adam optimizer with a learning rate of 0.0001 was used.

To create the spectrogram an STFT window of 1'536 samples is used. A window of length 64 slid over a spectrogram is used to get smaller batches of training data. A batch size of 4 was used during training. We treated all tracks as mono, removing panning where necessary, meaning only a single track was fed into the network. This results in an input to the network with the size of $(769, 64, 1)$.

6 Evaluation Methodology

To evaluate the results of a training, a Python library called *mir_eval* [37] was used. It offers functionality to calculate three measures commonly used in BSS, called Source-to-Distortion Ratio (SDR), Source-to-Interference Ratio (SIR) and Source-to-Artifact Ratio (SAR). All of these are measured in dB with higher numbers being better. Usually when only one number is given in music separation solutions it is SDR since it can be seen as a general measure on how good a source sounds. SIR can be described as the amount of other sources audible in a signal, which is usually described as bleed in audio production. SAR can be interpreted as the amount of unwanted artifacts a signal has.

These measures are calculated from the true source s_T, interference i, noise n and artifacts a, where the estimated source s_E is defined as $s_E = s_T + i + n + a$.

$$\begin{aligned} SDR &:= 10\ log_{10}\left(\frac{||s_T||^2}{||i+n+a||^2}\right) \\ SIR &:= 10\ log_{10}\left(\frac{||s_T||^2}{||i||^2}\right) \\ SAR &:= 10\ log_{10}\left(\frac{||s_T+i+n||^2}{||a||^2}\right) \end{aligned}$$

To give some examples of the ranges these values can reach, at the time of writing the five best performing systems on `https://paperswithcode.com` [11] reported an SDR of 7.1 to 7.8 dB when separating vocals and an SDR of 4.5 to 5.4 dB when separating *other* instruments, which is the category this thesis would presumably fit into. It is to be noted that any results from this thesis can not be directly compared with the above mentioned results, since different tracks were used for validation. While we would like to use the same validation set, the solutions on `https://paperswithcode.com` use musdb18 for validation, which does not have separated piano or guitar tracks.[4]

When evaluating a training, one of the datasets described in 4.5 is chosen. For each track in the validation set, the instrument and the rest sources are mixed together and ran through the trained network. This results in a newly separated instrument track, as well as a track containing everything not in the separated track. These tracks have differing amounts of differences to the true sources that were used as inputs to the network. At this point, these four sources are provided to the *mir_eval* function *bss_eval_sources*, which returns an array containing the SDR, SIR and SAR. These are saved for each track and later used to calculate a single value for each measure for the training by calculating the median of all saved measures.

[4]As noted in section 4.1

Additionally, the subjective listening experience of separated tracks need to be observed. While automatically computed metrics from systems like *mir_eval* can give a rough idea of quality, they do not map fully to human hearing and preferences.

7 Results

To evaluate the performance of the network, multiple trainings were ran. Unless noted otherwise, all trainings were performed with the following settings and restrictions to allow for proper comparison between then:

- 131 tracks used [5]
- 22050 Hz tracks used
- 36 epochs with early stopping if validation loss is stagnant for 10 epochs
- no limit on items from a single track
- datasets are not shuffled
- network structure from section 5.3 used
- trainings ran on the Graphics Processing Unit (GPU) of a server from InES [6]

We trained networks for two different instruments, piano and guitar, in four different variants. These variants are:

- the dataset without changes
- the dataset with reduced silences
- the dataset with pure instrument data added
- the dataset with reduced silences and pure instrument data added

An additional variant for guitar is the full dataset. Instead of the 131 tracks used to keep it in line with the piano dataset, here all tracks from dataset 1 were used.

Datasets are referenced as indexed in section 4.4. Training results are grouped by instrument to allow for easier comparison of the changes in the scores.

Additionally, we ran several test separations for each training and compared the listening experiences. Keep in mind that while we try to find and note obvious and objective differences between these, human hearing is highly subjective. Depending on familiarity with the tracks or the instrument in question these observations may vary greatly.

[5] the size of the base piano training set

[6] A Captiva gaming pc with a Nvidia GeForce RTX 3090

For each training, the median of SDR, SIR and SAR of all validation tracks are given, once for the separated instrument track and once for the rest track containing everything else. Results are rounded to two decimal places. Additionally, the time required to train for one epoch is given.

For trainings with mixed datasets, a ratio between the datasets as well as the total track counts used from each dataset are given. These may not always overlap perfectly when the total requested track count for the entire training could not be satisfied with the given datasets.

Mixing pure datasets with regular datasets resulted in highly unstable trainings. While we could not find the reason for this, we think these results are still relevant since some of them greatly affect separation. See section 8.3 for further discussion of this.

7.1 Guitar Trainings

7.1.1 Training on Dataset 1

Training of each epoch took about 1 1/4 hours.

Track	**SDR**	**SIR**	**SAR**
Guitar	−1.27	4.89	1.47
Rest	8.09	9.81	13.27

Table 2: Guitar results

7.1.2 Training on Dataset 2

Training of each epoch took about 1 1/4 hours.

Track	**SDR**	**SIR**	**SAR**
Guitar	−1.83	5.34	0.44
Rest	7.73	8.76	14.94

Table 3: Silence adjusted guitar results

7.1.3 Training on a Mix of Datasets 1 and 6

The ratio between dataset 1 and 6 was 3 to 1. Amount of total tracks used was 174, with 131 from dataset 1 and 43 from dataset 6. Stopped early after 33 epochs. Training of each epoch took about 1 1/2 hours.

Track	**SDR**	**SIR**	**SAR**
Guitar	−3.50	−2.20	8.42
Rest	5.50	17.27	6.09

Table 4: Guitar & pure guitar results

7.1.4 Training on a Mix of Datasets 2 and 6

The ratio between dataset 2 and 6 was 3 to 1. Amount of total tracks used was 174, with 131 from dataset 2 and 43 from dataset 6. Stopped early after 32 epochs. Training of each epoch took about 1 $^1/_2$ hours.

Track	**SDR**	**SIR**	**SAR**
Guitar	−6.69	−6.06	9.21
Rest	−1.08	5.85	1.20

Table 5: Silence adjusted guitar & pure guitar results

7.1.5 Training on Full Dataset 1

Training of each epoch took about 5 hours.

Track	**SDR**	**SIR**	**SAR**
Guitar	0.73	4.26	4.40
Rest	8.92	11.61	13.06

Table 6: Full guitar results

7.1.6 Comparisons

The baseline guitar training provides passable results, although separations in general are rather quiet. Bleed from other sound sources is noticeable here as well as in all other trainings, mostly from drums.

As seen in tables 2 and 3, adjusting for silences reduces most scores, but none to great amounts. Listening to example separations of these trainings shows similar results, with the silence adjusted training missing the guitar slightly more often.

Both trainings in tables 5 and 4 with the mixed in pure datasets had worse scores. When listening to example separations the reason for this seems clear, with a lot more bleed from vocals being audible.

Meanwhile, the results in table 6 of the training with all available guitar training data greatly improves SDR and SAR, while listening examples provide much better separation.

7.2 Piano Trainings

7.2.1 Training on Dataset 3

Training of each epoch took about one hour.

Track	**SDR**	**SIR**	**SAR**
Piano	−13.62	−3.01	−6.82
Rest	11.35	15.09	16.19

Table 7: Piano results

7.2.2 Training on Dataset 4

Training of each epoch took about 50 minutes.

Track	**SDR**	**SIR**	**SAR**
Piano	−8.46	1.93	−5.60
Rest	13.15	15.82	17.53

Table 8: Silence adjusted piano results

7.2.3 Training on a Mix of Datasets 3 and 7

The ratio between dataset 3 and 7 was 3 to 1. Amount of total tracks used was 174, with 131 from dataset 3 and 43 from dataset 7. Stopped early after 26 epochs. Training of each epoch took about 1 1/4 hours.

Track	**SDR**	**SIR**	**SAR**
Piano	−13.62	−13.30	12.01
Rest	2.31	14.29	3.53

Table 9: Piano & pure piano results

7.2.4 Training on a Mix of Datasets 4 and 7

The ratio between dataset 4 and 7 was 3 to 1. Amount of total tracks used was 166, with 123 from dataset 4 and 43 from from dataset 7. Stopped early after 30 epochs. Training of each epoch took about one hour.

Track	**SDR**	**SIR**	**SAR**
Piano	−16.09	−12.44	3.44
Rest	−0.15	15.04	0.52

Table 10: Silence adjusted piano & pure piano results

A second training with the same configurations provided much better results. Ratio between dataset 4 and 7 was 3 to 1. Amount of total tracks used was 170, with 123 from dataset 4 and 47 from dataset 7. Training of each epoch took about one hour.

Track	**SDR**	**SIR**	**SAR**
Piano	−7.28	4.89	−4.53
Rest	13.17	15.69	16.82

Table 11: Silence adjusted piano & pure piano results

7.2.5 Comparisons

Listening to examples of the baseline piano training, it is not quite capable of separating piano sounds cleanly. At best it reduces all other instruments to a quiet but still audible level. During busy section in a track, bleed from other sources greatly increases.

While scores from the silence adjusted dataset in table 8 are greatly improved in comparison to the base results in table 7, the listening experience is worse. A piano playing by itself is almost completely inaudible and other parts are full of artifacts and bleed.

Simply mixing in the pure piano dataset with the baseline piano dataset resulted in worse scores as seen in table 9. Example separations were full of bleed, with almost no actual separation happening.

When listening to the better scoring training on the mixed dataset of silence adjusted piano and pure piano from table 11, solo piano is cleary audible and other instruments are mostly reduced to background noise. Artifacts are still common and busier sections can also still throw off the separation.

7.3 Open-Source Software

All code used to create our datasets, train the networks and evalute our results have been made publicly available under the MIT license [38]. All code is available at `https://github.com/splitstrument`, grouped into repositories by topic. Due to licensing and copyright restrictions, we are unable to share the datasets we created. It should be possible to reconstruct the datasets by following our documentation in section 4 and the links to the cited works. We have uploaded the metadata files from our labelling and quality assurance tools to help with that endeavour. The code for our demonstrator contains two models for guitar and piano separations and requires no training.

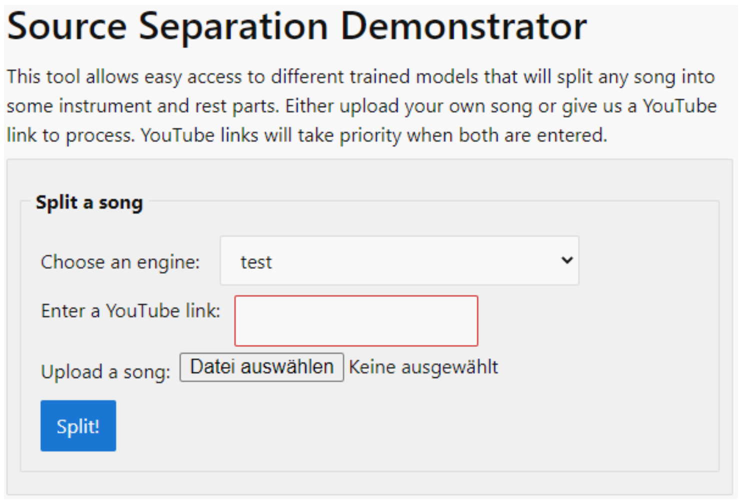
7.4 Demonstrator

A simple web application was developed to easily run separations with the trained models. It is built on the web.py library and can be configured with a list of trainings to use, to allow easy adjustement of the available models.

After choosing the model to use and providing a mix to separate, either through an upload functionality or by providing a link to a YouTube video, the resulting instrument and rest tracks are returned to the user.

The demonstrator is running on the same server as our trainings. This allows separations to run on its GPU, which greatly increases processing speed [7]. Since this server is not publicly available, this instance of the demonstrator can only be used through port forwarding from users with access to the server. The demonstrator itself can run on any system with the code and models from section 7.3, but using a GPU for processing is highly recommended.

[7]About three seconds for a five minute song



Source Separation Demonstrator

This tool allows easy access to different trained models that will split any song into some instrument and rest parts. Either upload your own song or give us a YouTube link to process. YouTube links will take priority when both are entered.

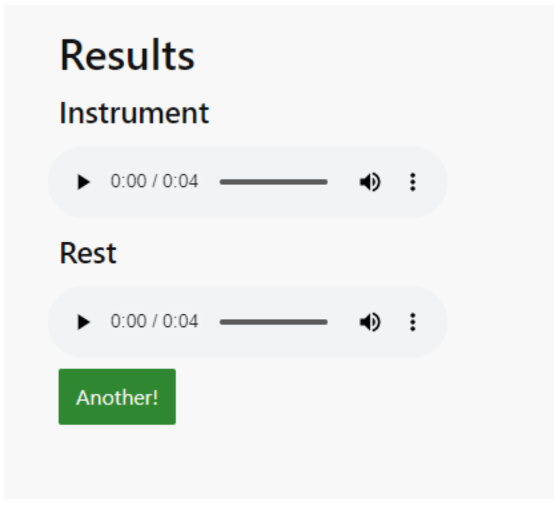
Split a song

Choose an engine: test

Enter a YouTube link:

Upload a song: Datei auswählen Keine ausgewählt

(a) Starting page



Results

Instrument

Rest

(b) Result display

Figure 12: Screenshots of web demonstrator

8 Discussion

8.1 Data Collection

Data collection turned out to be as difficult as we anticipated. While good datasets for BSS exist, only MedleyDB offered detailed enough stems so we could use them for our use case without further adjustement. This led us to make a thorough search to find more tracks to combine into a bigger dataset. This approach resulted in a sizeable dataset for guitar, mostly thanks to Rockband and Guitar Hero tracks, but piano tracks were rather hard to come by. The final piano dataset is smaller than we would have liked, but it sufficed for smaller trainings.

Using our own dataset allowed us to test and compare the efficiency of our different trainings, but objective comparisons with other approaches is difficult when the underlying data differs.

Looking to the future, we think that work on a expanded dataset with the quality and spread of musdb18 and the detailed stems of MedleyDB is warranted. Data is the most important part in any deep learning research and having access to a large amount of stems would make work similar to ours a lot easier. Since the music business is a difficult legal area, a collection of freely distributable tracks is hard to come by and would probably require a lot of negotations with either big labels or smaller independent artists.

8.2 Data Processing

Since we had to build a dataset by ourselves we could not rely on the assurance of quality other datasets had by virtue of their widespread use. This led us to create several tools to label and check tracks by hand for their instruments, quality and usefulness. This process took quite a long time, with us having to manually listen to hundreds of tracks, but gave us the security that all tracks we used were fit to be used for training or validation. We believe these tools to have further use in future work when building music datasets. Expansion of these tools with more dedicated solutions to instrument recognition or quality checking, like recognition clipping or insuffícent audio levels, would also cut down on the manual work required.

When looking at our piano data set we discovered that in many tracks the piano only played for a fraction of the runtime. Since we had a suspicion that this might disturb piano trainings, we also created a tool to cut tracks to those parts where an instrument plays. While it is a rather crude tool, we would be interested in seeing this approach applied to different datasets to analyze the influence of silence in other situations.

8.3 Trainings

When looking at the results in sections 7.1 and 7.2, we can see that the question of whether the previous network used to separate vocals can be easily used to separate melodic instrument is a difficult one. While the base approach seems applicable, there is some tweaking necessary for the results to sound passable. With most of our adjustements and experimentation focused on dataset enhancement, there is still a wealth of potential tweaks available on the neural network side of things, including tuning hyperparameters like learning rate, dropout rate or number of hourglass stacks. One interesting addition we could not further explore is the use of perceptual loss functions. [39]

When comparing the results of our trainings, there are a number of interesting points we can pull from them. Firstly, there seems to be an inherent difference between the two instruments we have tried to separate, even when most other variables, like data size or amount of silences in the tracks, are the same. To our surprise, while we had expected piano to be easier for the network to separate, it was actually guitar, often times even distorted guitar, that produces good results.

Secondly, our different data enhancement techniques are not universally applicable, but rather they need to be used sparingly and only when sensible. These enhancements were experimented with because our initial trainings on the base piano dataset were below our expectations, with silence adjustement meant to focus the network more on actual piano playing and pure datasets on top of that meant to enable it to recognize solo piano playing. So these enhancements should be thought of more as a specific collection of steps to enhance a piano dataset. For instance, adjusting the silence in the guitar dataset, which is already mostly filled with tracks without a lot of silence, as can be seen in appendix A, actually leads to worse results.

Thirdly, mixing our regular datasets with pure datasets of only the instrument in question playing resulted in highly unstable trainings. Since this was only recognized late in our work, with earlier trainings luckily being successful, we were not able to find the cause for this behaviour. Having tested varying ratios between datasets and different total track counts, we are fairly certain that the specific tracks used for training are not to blame, since our datasets were not shuffled when training and sorted in a deterministic manner. Additional exploration in this area might yield improved results, since the scores of the succesful training in section 7.2.4 showed that the general approach can help with shortcomings of previous trainings.

Lastly, in relation to the discussion in section 8.1, the training with the best result is the one with the most available data. It would be interesting to see the different data enhancement techniques applied to this full dataset, but since those trainings take about a week to train we were not able to run those due to time constraints. Additionally, comparing this training to one on a similarly sized piano dataset might give additional information on the inherent differences when training on these two instruments. Similarly, while we had initially researched and wanted to experiment with the Slakh2100-redux dataset, we could not fit it into our work. This is another approach to the problem of small datasets that may be explored. Any exploration of even more additional instruments would as well be highly interesting, but might be stifled by the amount of data available.

8.4 Open-Source Software code

Since we intended to release all our developed code to the public after the conclusion of this thesis, we built all our tools with the purpose of them being easily expandable and usable in other situations. While there exists a wealth of tools in the BSS world, not many of them are used to build new datasets. We hope our tools can assist future researchers in creating better datasets and inspire other developers to build tools to this purpose.

9 Conclusion

In this thesis, we examined the application of a BSS approach using CNNs previously intended to separate vocals from mixes to melodic instruments, specifically piano and guitar. While the general approach seems applicable, it can not be transferred to these instruments directly and without adjustements. Limitations in publicly available datasets forced us to create our own dataset, which limits the comparability to state of the art approaches since these are commonly compared using the same validation dataset. The fast moving field of BSS offers many different approaches, but they are seldomly applied to instruments outside of the usual dataset. Future research into the separation of melodic instruments could investigate the success of other approaches to this problem. Additional work is necessary to build a publicly available dataset for melodic instruments.

To demonstrate the results of this thesis, a simple web application was created. All code for the creation of the datasets, training, evaluation and the web application have been made publicly available.

References

[1] David Flury, Andreas Kaufmann and Raphael Müller. *Vocal Extraction from Music using Deep Learning.* June 2019.

[2] Waqas Akram. *Quantization in ADCs.* May 2008. URL: `https://dwellangle.wordpress.com/2008/05/21/quantization-in-adcs/` (visited on 27/05/2021).

[3] Editorial Staff. *That's Intense: Understanding the Decibel Scale.* Feb. 2014. URL: `https://www.hearinglikeme.com/thats-intense-understanding-the-decibel-scale/` (visited on 23/05/2021).

[4] Keiron O'Shea and Ryan Nash. "An Introduction to Convolutional Neural Networks". In: *ArXiv e-prints* (Nov. 2015), p. 10.

[5] datahacker.rs. *More On Edge Detection.* Nov. 2018. URL: `http://datahacker.rs/edge-detection-extended/` (visited on 04/06/2021).

[6] D. Sundararajan. "Multirate Digital Signal Processing". In: *Digital Signal Processing: An Introduction.* Cham: Springer International Publishing, 2021, pp. 285–318. ISBN: 978-3-030-62368-5. DOI: `10.1007/978-3-030-62368-5_8`. URL: `https://doi.org/10.1007/978-3-030-62368-5_8`.

[7] Hossein Gholamalinezhad and Hossein Khosravi. *Pooling Methods in Deep Neural Networks, a Review.* 2020. arXiv: `2009.07485 [cs.CV]`.

[8] Nitish Srivastava. "Improving neural networks with dropout". In: *University of Toronto* 182.566 (2013), p. 26.

[9] Shibani Santurkar et al. *How Does Batch Normalization Help Optimization?* 2019. arXiv: `1805.11604 [stat.ML]`.

[10] Sergey Ioffe and Christian Szegedy. "Batch normalization: accelerating deep network training by reducing internal covariate shift. arXiv e-prints". In: (2015).

[11] *Music Source Separation on MUSDB18.* URL: `https://paperswithcode.com/sota/music-source-separation-on-musdb18` (visited on 13/05/2021).

[12] Naoya Takahashi and Yuki Mitsufuji. *D3Net: Densely connected multidilated DenseNet for music source separation.* 2021. arXiv: `2010.01733 [eess.AS]`.

[13] Woosung Choi et al. *LaSAFT: Latent Source Attentive Frequency Transformation for Conditioned Source Separation.* 2021. arXiv: `2010.11631 [cs.SD]`.

[14] Naoya Takahashi, Nabarun Goswami and Yuki Mitsufuji. *MMDenseLSTM: An efficient combination of convolutional and recurrent neural networks for audio source separation.* 2018. arXiv: `1805.02410 [cs.SD]`.

[15] Alexandre Défossez et al. *Demucs: Deep Extractor for Music Sources with extra unlabeled data remixed.* 2019. arXiv: `1909.01174 [cs.SD]`.

[16] Romain Hennequin et al. "Spleeter: a fast and efficient music source separation tool with pre-trained models". In: *Journal of Open Source Software* 5.50 (2020). Deezer Research, p. 2154. DOI: 10.21105/joss.02154. URL: https://doi.org/10.21105/joss.02154.

[17] Yi Luo and Nima Mesgarani. "Conv-TasNet: Surpassing Ideal Time–Frequency Magnitude Masking for Speech Separation". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 27.8 (Aug. 2019), 1256–1266. ISSN: 2329-9304. DOI: 10.1109/taslp.2019.2915167. URL: http://dx.doi.org/10.1109/TASLP.2019.2915167.

[18] David Samuel, Aditya Ganeshan and Jason Naradowsky. *Meta-learning Extractors for Music Source Separation.* 2020. arXiv: 2002.07016 [cs.SD].

[19] Fabian-Robert Stöter et al. "Open-Unmix - A Reference Implementation for Music Source Separation". In: *Journal of Open Source Software* 4.41 (2019), p. 1667. DOI: 10.21105/joss.01667. URL: https://doi.org/10.21105/joss.01667.

[20] Francesc Lluís, Jordi Pons and Xavier Serra. *End-to-end music source separation: is it possible in the waveform domain?* 2019. arXiv: 1810.12187 [cs.SD].

[21] Daniel Stoller, Sebastian Ewert and Simon Dixon. *Wave-U-Net: A Multi-Scale Neural Network for End-to-End Audio Source Separation.* 2018. arXiv: 1806.03185 [cs.SD].

[22] Zafar Rafii et al. *The MUSDB18 corpus for music separation.* Dec. 2017. DOI: 10.5281/zenodo.1117372. URL: https://doi.org/10.5281/zenodo.1117372.

[23] Antoine Liutkus et al. "The 2016 Signal Separation Evaluation Campaign". In: *Latent Variable Analysis and Signal Separation - 12th International Conference, LVA/ICA 2015, Liberec, Czech Republic, August 25-28, 2015, Proceedings.* Ed. by Petr Tichavský et al. Cham: Springer International Publishing, 2017, pp. 323–332.

[24] R. Bittner et al. "MedleyDB: A Multitrack Dataset for Annotation-Intensive MIR Research". In: *15th International Society for Music Information Retrieval Conference.* Taipei, Taiwan, Oct. 2014.

[25] Rachel M. Bittner et al. "MedleyDB 2.0: New Data and a System for Sustainable Data Collection". In: *Late breaking/demo extended abstract, 17th International Society for Music Information Retrieval (ISMIR) conference.* Aug. 2016.

[26] Ethan Manilow et al. "Cutting Music Source Separation Some Slakh: A Dataset to Study the Impact of Training Data Quality and Quantity". In: *Proc. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA).* IEEE. 2019.

[27] *Rockband and Guitar Hero multitracks.* 2016. URL: http://multitrackdownloads.blogspot.com/ (visited on 26/03/2021).

[28] Cambridge Music Technology. *The 'Mixing Secrets' Free Multitrack Download Library*. URL: https://www.cambridge-mt.com/ms/mtk/ (visited on 26/03/2021).

[29] kno Disney Piano Channel. *RELAXING PIANO Disney Piano Collection 3 HOUR LONG (Piano Covered by kno)*. Mar. 2015. URL: https://www.youtube.com/watch?v=5DiMoehAeOU (visited on 06/05/2021).

[30] Dmitry Veremchuk. *1 Hour of romantic Jazz (piano) work, chill out, sleep.* Apr. 2020. URL: https://www.youtube.com/watch?v=MdkLZZ2-V3Y (visited on 06/05/2021).

[31] Acoustic Music Collection. *1 Hour Acoustic Relaxing Music for Relaxation, Sleep, Study & Work*. Aug. 2019. URL: https://www.youtube.com/watch?v=1iM_p0OzMTg (visited on 09/05/2021).

[32] LusitaniaGuitar. *One Hour of Electric Guitar Solos - complete album.* Nov. 2015. URL: https://www.youtube.com/watch?v=DQQELxSWZE8 (visited on 09/05/2021).

[33] painmanist. *Dave Grohl - Play [Isolated Guitar 1]*. Aug. 2018. URL: https://www.youtube.com/watch?v=bJRHqyJCshg (visited on 09/05/2021).

[34] painmanist. *Dave Grohl - Play [Isolated Guitar 2]*. Aug. 2018. URL: https://www.youtube.com/watch?v=SGGHu61WdSU (visited on 09/05/2021).

[35] painmanist. *Dave Grohl - Play [Isolated Guitar 3]*. Aug. 2018. URL: https://www.youtube.com/watch?v=q0ImobY-VlU (visited on 09/05/2021).

[36] Sungheon Park et al. "Music Source Separation Using Stacked Hourglass Networks". In: *CoRR* abs/1805.08559 (2018). arXiv: 1805.08559. URL: http://arxiv.org/abs/1805.08559.

[37] Colin Raffel et al. "mir_eval: A Transparent Implementation of Common MIR Metrics". In: *Proceedings - 15th International Society for Music Information Retrieval Conference (ISMIR 2014)*. 2014.

[38] SPDX. *MIT License*. 2018. URL: https://spdx.org/licenses/MIT.html (visited on 08/06/2021).

[39] Dan Elbaz and Michael Zibulevsky. *Perceptual audio loss function for deep learning*. 2017. arXiv: 1708.05987 [cs.SD].

Glossary

batch normalization a neural network layer that rescales and recenters its inputs. 16, 27, 28

bleed audible audio from other unexpected sources. 22, 23, 30

BSS Blind Source Separation. 1, 3, 8, 9, 18, 19, 20, 30, 40, 42, 43

CMT Cambridge Music Technology. 1, 20, 21, 24

CNN Convolutional Neural Network. 11, 13, 16, 18, 29, 43

convolution a mathematical function used in neural networks to create feature maps by sliding filters over an input. 27, 28, 29

dB decibel. 8, 23, 30, 47

dBFS dB relative to full scale. 23

decibel a relative unit of measurement, used in this paper in relation to audio. 8, 47

deep learning a method of machine learning. 6, 11

DFT Discrete Fourier Transform. 8

dropout a neural network layer that randomly drops connections at some rate to promote generalization. 16, 27, 28

dsd100 a music dataset. 20

epoch number of passes over the entire training dataset. 32

feature map the output of a convultional layer. 13, 15

FFT Fast Fourier Transform. 28, 52

filter a group of weights to combine some inputs together. 13, 27, 28, 29, 47

FLAC a file format containing audio data with a lossless encoding. 20

GPU Graphics Processing Unit. 32, 38

hertz Hertz is a SI unit used to measure repeating cycles, one hertz being defined as one cycle per second. 7, 47

Hz hertz. 7, 23, 24, 25, 32, 48

InES Institute of Embedded Systems. 2, 32

learning rate defines how fast the weights in a neural network are updated. 29

loss function a function comparing the expected and actual output of a neural network. 12, 29

machine learning the study of algorithms that improve through experience. 6, 11, 47

MedleyDB a music dataset. 1, 20, 24, 40

MIDI Musical Instrument Digital Interface. 20, 48

MIR Music Information Retrieval. 20

mir_eval a python library used to evaluate MIR systems. 1, 30, 31

musdb18 a music dataset. 20, 30, 40

Musical Instrument Digital Interface a communications protocol for electronic instruments. 20, 48

neural network a network of neurons used to process data. 1, 6, 7, 11, 12, 16, 18, 19, 41, 47, 48

neuron the smallest unit in a neural network. 11, 12, 48

OSS Open-Source Software. 1, 6

Python the programming language used in this thesis. 30, 49

ReLU Rectified Linear Unit. 11, 28

RNN Recurrent Neural Network. 18

sample rate the amount of audio samples in Hz. 7, 22, 23

SAR Source-to-Artifact Ratio. 1, 30, 33, 34, 35, 36, 37, 52, 53, 54

SDR Source-to-Distortion Ratio. 1, 28, 30, 33, 34, 35, 36, 37, 52, 53, 54

SIR Source-to-Interference Ratio. 1, 30, 33, 34, 36, 37, 52, 53, 54

Slakh2100-redux a MIDI music dataset. 20, 25, 42

spectrogram a visual represenation of frequencies as they vary through time. 3, 8, 10, 12, 18, 27, 29

Stacked Hourglass a network structure utilising convolutions, down- and upsampling. 1, 18, 27, 29

stem a part of a mix, usually containing audio from exactly one instrument. 20, 21, 22, 23, 24, 40

STFT Short-Time Fourier Transform. 8, 18, 27, 29

WAVE a file format containing pure audio data. 20, 22, 23

web.py a simple web framework written in Python. 38

YAML YAML Ain't Markup Language. 20

YAML Ain't Markup Language a file format for structured data, similar to JSON. 49

A Silence Adjustement Statistics

Outputs from our silence detection tool on different datasets as described in 4.4.

A.1 Recorded Guitar

```
Songs with less than 10% silence: 496
Songs with less than 20% silence: 61
Songs with less than 30% silence: 17
Songs with less than 40% silence: 9
Songs with less than 50% silence: 4
Songs with less than 60% silence: 3
Songs with less than 70% silence: 1
Songs with less than 80% silence: 1
Songs with less than 90% silence: 1
Songs with more than 90% silence: 0
```

A.2 Recorded Guitar (Silence Adjusted)

```
Songs with less than 10% silence: 502
Songs with less than 20% silence: 61
Songs with less than 30% silence: 24
Songs with less than 40% silence: 5
Songs with less than 50% silence: 0
Songs with less than 60% silence: 0
Songs with less than 70% silence: 0
Songs with less than 80% silence: 0
Songs with less than 90% silence: 0
Songs with more than 90% silence: 0
```

A.3 Recorded Piano

```
Songs with less than 10% silence: 39
Songs with less than 20% silence: 23
Songs with less than 30% silence: 14
Songs with less than 40% silence: 8
Songs with less than 50% silence: 10
Songs with less than 60% silence: 11
Songs with less than 70% silence: 15
Songs with less than 80% silence: 13
Songs with less than 90% silence: 11
Songs with more than 90% silence: 9
```

A.4 Recorded Piano (Silence Adjusted)

```
Songs with less than 10% silence: 53
Songs with less than 20% silence: 22
Songs with less than 30% silence: 21
Songs with less than 40% silence: 12
Songs with less than 50% silence: 15
Songs with less than 60% silence: 0
Songs with less than 70% silence: 0
Songs with less than 80% silence: 0
Songs with less than 90% silence: 0
Songs with more than 90% silence: 0
```

A.5 Synthezised Piano

```
Songs with less than 10% silence: 116
Songs with less than 20% silence: 37
Songs with less than 30% silence: 19
Songs with less than 40% silence: 18
Songs with less than 50% silence: 19
Songs with less than 60% silence: 22
Songs with less than 70% silence: 16
Songs with less than 80% silence: 18
Songs with less than 90% silence: 24
Songs with more than 90% silence: 11
```

B Experimental Trainings

Datasets are referenced as indexed in section 4.4. Unless noted otherwise assume configurations from section 5.3 were used.

B.1 Amount of Silences in Dataset

All trainings were ran on variations of the dataset 3 with a initial filter size of 256.

B.1.1 Slight Silence Adjustement

Adjusted tracks with a silence content of 70% or more to 50% or less. Training of each epoch took slightly less then one hour.

Track	**SDR**	**SIR**	**SAR**
Piano	−8.42	0.23	−5.31
Rest	12.83	15.76	15.87

B.1.2 Harsh Silence Adjustement

Adjusted tracks with a silence content of 30% or more to 30% or less. Training of each epoch took about 45 minutes.

Track	**SDR**	**SIR**	**SAR**
Piano	−7.47	−0.51	−2.59
Rest	12.38	16.36	16.50

B.1.3 Silence Adjustement With a Lower Bound Ratio

Adjusted tracks with a silence content of 30% or more to 30% or less. Any track that could not be adjusted to be below 50% silence content were discarded afterwards. Training of each epoch took about 50 minutes.

Track	**SDR**	**SIR**	**SAR**
Piano	−7.99	2.79	−3.94
Rest	13.22	15.85	16.91

B.2 Different FFT Window Sizes

All trainings were ran on the entirety of the dataset 1 but with a limit of 50 items per track with a initial filter size of 256. FFT window sizes were chosen as powers of 2, as is usually done in audio processing. The exception is 1536 used in the previous thesis.

B.2.1 Window of 512 Samples

Ran for 42 epochs because of misconfiguration. Training of each epoch took about 1 1/4 hours.

Track	**SDR**	**SIR**	**SAR**
Guitar	−0.27	2.88	4.97
Rest	8.71	10.94	12.72

B.2.2 Window of 1024 Samples

Training of each epoch took about 1 1/4 hours.

Track	**SDR**	**SIR**	**SAR**
Guitar	0.15	3.95	4.41
Rest	8.76	10.82	13.41

B.2.3 Window of 1536 Samples

This is the window size used in the previous implementation. [1, p. 29] Stopped early after 35 epochs. Training of each epoch took about 1 1/2 hours.

Track	**SDR**	**SIR**	**SAR**
Guitar	−0.71	4.03	4.54
Rest	8.79	10.29	14.38

B.2.4 Window of 2048 Samples

Stopped early after 30 epochs. Training of each epoch took about 1 1/4 hours.

Track	**SDR**	**SIR**	**SAR**
Guitar	0.58	3.83	4.65
Rest	8.70	10.87	13.80

B.3 Different Initial Filter Sizes

All trainings were ran on the entirety of the dataset 1 but with a limit of 50 items per track. Filter sizes were chosen to reduce the amount of trainable parameters of the network by about factor 2 and 4.

B.3.1 256 Filters

This is the amount of filters used in the previous implementation. [1, p. 25] Count of trainable parameters in network were 12’546’689. Stopped early after 35 epochs. Training of each epoch took about 1 1/2 hours.

Track	**SDR**	**SIR**	**SAR**
Guitar	−0.71	4.03	4.54
Rest	8.79	10.29	14.38

B.3.2 192 Filters

Count of trainable parameters in network were 7’713’857. Training of each epoch took about 1 1/4 hours.

Track	**SDR**	**SIR**	**SAR**
Guitar	0.49	3.59	4.72
Rest	8.74	10.84	13.76

B.3.3 92 Filters

Count of trainable parameters in network were 2’671’757. Training of each epoch took about one hour.

Track	**SDR**	**SIR**	**SAR**
Guitar	0.24	3.50	4.97
Rest	8.78	10.88	13.68

C Weekly meeting protocols

C.1 01.03.2021

Wunsch nach zusätzlichem Reiter mit Wiki, wichtigen Links wie Repository-Link, Zeitplan, Trainingsdaten, relevante Papers, etc.
Server für Training ist im Aufbau, Liefertermin bald fällig
Kurzer Beschreib zu MedleyDB v1 und v2
196 Tracks
Alle möglichen Instrumente
Scaper als mögliches Tool zum Mixing
Potentielle selber generierte Datenquellen aus unabhängigen Spuren
Vergleich von Training auf echten Songs, Midi-Songs, generierten Datenquellen
Organisation der Datenquellen nach Song und Instrumenten
Datenquellen werden auf dem Server des ZHAWs gelagert
Arbeiten / Processing an den Daten als Metadaten sonst wo speichern für den Fall von Datenverlusten
Stems zusammenfassen zu Gruppen von gleichen Instrumenten
Qualität von Stems von Hand / Ohr prüfen
Fokus auf gute Aufbereitung der Datenquellen legen
Rosenthal schickt Mail wenn Server ready sind und LateX-Template
Erledigt
Theorie Neural Networks und Source Separation
Vorgängertool zum Laufen gebracht
Migration Tensorflow v1 zu v2
Datenquellen (MedleyDB und Slakh) heruntergeladen
Ziele
Nächste Woche
Zusätzliche Datenquellen
Datenquellen auf Instrumenten prüfen
Prüfen: Mixes aus langen einzelnen unabhängigen selber zusammenstellen
Weiter
Stems zusammenfassen zu Gruppen von gleichen Instrumenten
Datenquellen auf Bleed und sonstige Qualität filtern (eigenes Quality-Check-Tool)
Post-Processing-Tool

C.2 08.03.2021

Report zum richtigen Zeitpunkt abgegeben :-)
Scaper Experiment kurz erklärt
Die einzelnen Quellen präsentiert:
http://musicstems.org/artists: knappe Auswahl, aber könnte nutzbar sein.
https://www.native-instruments.com/en/specials/stems-for-all/free-stems-tracks/: Nur Electronic Tracks

https://isolated-tracks.com/: Alles versteckt hinter Premium Download Service. Extrem mühsam für die Datensammlung
Guitar Hero Stems nochmals suchen. Könnten sehr nützlich sein.
Daten Statistik Tool und Quality Tool präsentiert.
ZHAW Maschine sollte ende Woche Einsatzbereit sein.
MedleyDB Daten aufbereiten und ein erstes Training machen für einen PoP (Proof of Principle)
Ein-Zeiler falls ne gute Quelle gefunden
Erledigt
Statistiken über MedleyDB und Slakh gesammelt
Tool für Datenumstrukturierung geschrieben
Tool für Qualitätyanalyse geschrieben (mit Metadata-Export im Falle von Datenverlust)
Erster Versuch mit Scaper erstellt
Ziele
Nächste Woche
Guitar Hero Stems nochmals suchen. Könnten sehr nützlich sein.
Trainingsdaten aufbereiten.
Erste Trainingsversion in Angriff nehmen.
Weiter
Proof of Principle vervollständigen

C.3 15.03.2021

GPU-Auslastung prüfen oder ob Festplatte-Auslastung
Stems nach Mixing vor Spektogram als Wav-File abspeichern (einmal Instrument einmal Rest)
Vorstellung von Proof of Concept und Datensuche
Optionen für Instrument auf Piano und Gitarre legen, je nach Menge im Datenset
Auf nächste Woche Datensuche finalisiert, weiter Datenverarbeitung
Training möglicherweise direkt auf WAV-Files und Spektograme live generieren
Vorgemixte Files ignorieren und selber mixen
Pfad zu Trainingsdaten auf dem Server schicken
Potentielles Training auf Vocals einfach Bruteforce ohne Quality Check
Einzeiler zu Status sind willkommen

C.4 22.03.2021

Gitarre Daten runtergeladen
Manuell via Quality Tool alle Songs überprüft
Mixing Tool ist in der Nacht abgestürzt. Es hat noch einen Fehler drin
425 Songs konnten allerdings bereits gemischt werden
Gitarre Training machen mit ca. 100 Songs für nächste Woche
Bei schlechten Trainingsresultaten prüfen ob eine modifizierte FFT Länge einen

Unterschied macht oder nicht
Zum aus Musikfiles lernbare Spektogramme zu generieren:
1. Spektogram für einen Track generieren
2. Spektogram für zweiten Track generieren
3. Für Mix Spektogramme addieren
4. Den Betrag der Spektogram-Werte ins Netz füttern
Erste brauchbare Trainingsresultate sollten nach ca. 36 Epochen vorhanden sein
Ziele
Gitarren Daten fertig mischen, Training starten
Piano Daten aufbereiten, damit sie bereit sind
Daten Teil in den Report schreiben
Evtl. einen Blick auf Loss Functions werfen

C.5 29.03.2021

Cambridge Piano Songs heruntergeladen und aufbereitet
Trainings auf Gitarre und ein Test auf Piano
Spektrogramm-Länge beeinflusst Trainingsdatengrössen
FFT-Untergrenze 512, Obergrenze 4096
Chopper umbauen dass es immer gleich viele Daten nimmt
Penalties auf Frequenzbereiche wären eine Möglichkeit zur Verbesserung
Piano-Daten auf Stille korrigieren, mindestens 20% Piano vorhanden
Genre-Zusammenstellung für Cambridge-Daten
Gitarren-Trainings mit verschiedenen FFT-Längen und Loss-Funktionen
Piano mehr ausprobieren
mir_eval automatisieren (FFT-Längen und Instrumente vergleichen)
Slakh-Daten anschauen für Piano (tiefe Prio)
Nächste Sitzung in 2 Wochen, bis da hin Statusupdates zu Trainingsdaten und Separations
Filenamen bei Prediction korrigieren
Ziele
1. Gitarre ausprobieren
2. Piano aufhübschen
3. Validationsets (15% des Trainingssets)
4. Automatische Qualitätskontrolle (mir_eval)
5. Slakh

C.6 12.04.2021

Genres der Cambridge Songs herausgeschrieben
Silence Filtering von Piano Songs gemacht
Slakh Datenset genauer geprüft
Variable Anzahl Items pro Song angepasst je nach FFT Länge
mir_eval erstellt

Trainings mit 50 Items pro Song: FFT Längen: 1536, 512, 2048
Performance Problem mit 2048 erzählt
mir_eval besprochen: SDR zeigt am ehesten Unterschiede, 1536 bis jetzt am besten
Report wird von den Betreuern bewertet.
Soll Arbeit präsentieren
Source-Separation Grundlagen sollen in den Report (Was es ist, schwer, neuronale Netze)
Netz kleiner machen könnte man als nächstes machen
Visualisierungen von Resultaten erstellen
Subjektive Empfindungen mit mir_eval vergleichen
Für nächste Woche:
Netz kleiner machen für Gitarre
Weniger Parameter verwenden/ Filtergrösse anpassen
Bleibt Quali gleich oder wird es schlechter?

Piano Trainings
Medley DB Datensatz
Separaten Slakh Datensatz
Gemischter Datensatz

Andere Loss function verwenden (tiefe Prio)
Breite der Items prüfen

C.7 19.04.2021

Zeiten für verkleinerte Netzwerk an Timestamps
Trainings für verkleinerte Netze erzählt
Piano-Datensatz erläutert
Besprechung wegen leeren Ordnern
Evaluation mit mir_eval und eigenen Ohren
192 Filter gut für Trainable Params
Bei Evaluations mit mir_eval ist Median gut bei vielen Songs
Daten-Teil vom Report abschliessen
Training-Teil bearbeiten im Report (speziell Gitarre)
Nächste Woche Pfad fokussieren und wo genau weiter machen
Potentielle weitere Trainings:
Loss-Funktionen
Datensätze vergleichen
Andere Netzwerk-Strukturen
Fully connected
Gating

Tool für verschiedene Spektrogramme
Alles zusammen

Ideales Instrument
Predicted Instrument
Präsentation wird an Zivilschutz irgendwie vorbeigedeichselt werden können

C.8 26.04.2021

Piano Daten die Silence noch mehr rausgeschnitten
mir_eval knapp besser
Evaluation Tool erweitert. Macht nun Spectogram Plots direkt auf den Server
Theorie angeschaut damit Ideen vorhanden sind für die Loss Functions
Die verschiedenen Piano Resultate präsentiert
Versuch: Beim Validation set die Silence rausschneiden und nochmals evaluieren
Nächste Schritte:
Reines Piano set hinzufügen und dann auf allem Trainieren
Training erweitern damit dies konfigurierbar ist
SDR auf geschnittenem Piano und vergleichen
Piano als Input und Output im Training
Variable Datensets im Netzwerk
30% pures Piano beimischen
Auf 192 Netz trainieren
Security auf Evaluation-Files prüfen
Thresholding wenn Silence Probleme macht

C.9 03.05.2021

Besprechung zu purem Piano
Link zu Download-Tool posted
Pures Piano verfolgen
Ratios ausprobieren
50:50, ¡3:1, einfach paar verschiedene Ratios
Ausprobieren mit anderem Content als einfach Songs
Guitar Only Dataset erstellen
Gitarre und Piano vergleichen mit gleicher Datenmenge
Vergleich mit grösserem Gitarren-Training
Vergleich mit entfernter Silence
Vergleich mit zugemischtem purem Instrument
Gitarren mit Augmentations priorisieren
Message der Arbeit ist Unterschiede in Extraktion bei Gitarre und Piano
Demonstrator simple halten, webpy.org als potentielles Framework
Für nächste Woche:
Liste erstellen von Trainings zu vergleichen
Trainings durchführen und evaluieren (so viele wie halt gerade geht)
Zur Unterstützung: Tool für konsekutive Trainings ohne User-Input?

Trainings vergleichen
Evaluation-Tool optimieren für Ausgabe mit gemischten Datensets

C.10 10.05.2021

Pure Gitarre Quellen erläutert
Erste erkenntnisse der ersten 4 Trainings gezeigt
Am Report wurde weitergeschrieben
Generalisieren der Resultate soll erreicht werden, damit die Ergebnisse besser wertbar sind.
Unsere Lösung mit anderen Systemen vergleichen
Pures Datenset validieren auf Silence in rest
Report:
Schreibhilfe kontaktieren, nur "fertige" Kapitel zu den Betreuern schicken
Wer soll es lesen: Mitstudenten, nicht-Informatiker sollen es lesen können
Goal bei Kapitel 3
Kapitel zwischen Theorie und Training data: Konzept der Arbeit (Vorgehensweise)
Theoretical Foundation soweit gut. Besser bündeln: 1. Audio, 2. Neural Networks
Network: Hourglass Beschreibung hinzufügen, was wurde hinzugefügt von uns
Besonderheiten Hourglass notieren
Hierarchisch
Skip Connections

welche Loss function wurde verwendet
Evaluation Kapitel name ändern: wie wurde die Auswertung gemacht
Kapitel: Results muss hinzugefügt werden
Für nächste Woche:
Gitarre mit vollem Datenset noch zur Trainingsliste hinzufügen
Evtl. die "Dose voll aufmachen" für Gitarre. D.h. alles plus pure guitar

C.11 17.05.2021

Fragen:
Loss Function L1,1
Report
Changes to network -¿ Optimization?
Concept Data
Wie Daten processed
Was Input Netzwerk
Spleeter Docker Files analysieren
Netzwerkstruktur
Andere State Of The Art vergleichen

Für nächste Woche:
Gitarren-Training mit vollem Datensatz
Demonstrator
Mit anderen Tools vergleichen
Struktur
Resultate

Spleeter-Analyse

C.12 31.05.2021

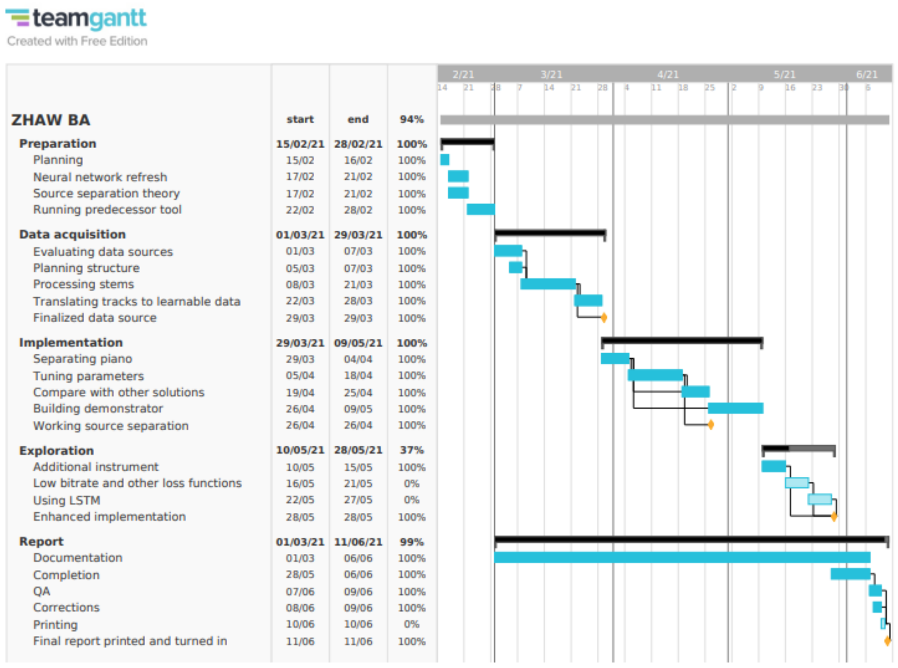
Demonstrator gezeigt
Report fortschritt erwähnt
Alle Resultate aufschreiben, weniger bei dem weniger “guten” zeugs
Open Source
MIT oder ähnlich permissive Lizenz
Piano Trainings
Reinschreiben auch wenn nicht 100% klar
Betrieb Demonstrator
Auf Server laufen lassen
Demonstrator aus Appendix raus
Thresholding bei Training?
Nur Validation

C.13 07.06.2021

Report erzählt
Bis Mittwoch erste Version vom Publikationstools eingegeben
Demonstrator mit Betrieb präsentiert
Auf MIT-Lizenz entschieden
ZHAW-Logo in READMEs verpacken
Code trotz Github-Veröffentlichung in Zip? Nein
Neural Network und ML immer gross? Nein, nur in Titel
Protokolle auf Deutsch? Ist okay
Präsentation Guidelines
$\frac{3}{4}$ Stunde maximal
15 - 20 Minuten Präsentation, Vorstellung der Arbeit
Was haben wir gemacht? Wie haben wir es gemacht? Was ist die Eigenleistung? Was ist das Resultat?
1 - 2 Themen die speziell interessant / schwierig sind auswählen und im Detail erzählen
Storytelling wichtig
Demonstration anschliessend
Sprache soweit frei (auch schweizerdeutsch)

Fliessend zum Experten übergeben mit Fragen wie was gelöst wurde
Bis Mittwoch:
Liste von Ordner auf dem Server
Publikationstool
Nach Abgabe kurz Message schicken und das was abgegeben wurde zur Verfügung stellen

D Gantt chart



ZHAW BA	start	end	94%
Preparation	**15/02/21**	**28/02/21**	**100%**
Planning	15/02	16/02	100%
Neural network refresh	17/02	21/02	100%
Source separation theory	17/02	21/02	100%
Running predecessor tool	22/02	28/02	100%
Data acquisition	**01/03/21**	**29/03/21**	**100%**
Evaluating data sources	01/03	07/03	100%
Planning structure	05/03	07/03	100%
Processing stems	08/03	21/03	100%
Translating tracks to learnable data	22/03	28/03	100%
Finalized data source	29/03	29/03	100%
Implementation	**29/03/21**	**09/05/21**	**100%**
Separating piano	29/03	04/04	100%
Tuning parameters	05/04	18/04	100%
Compare with other solutions	19/04	25/04	100%
Building demonstrator	26/04	09/05	100%
Working source separation	26/04	26/04	100%
Exploration	**10/05/21**	**28/05/21**	**37%**
Additional instrument	10/05	15/05	100%
Low bitrate and other loss functions	16/05	21/05	0%
Using LSTM	22/05	27/05	0%
Enhanced implementation	28/05	28/05	100%
Report	**01/03/21**	**11/06/21**	**99%**
Documentation	01/03	06/06	100%
Completion	28/05	06/06	100%
QA	07/06	09/06	100%
Corrections	08/06	09/06	100%
Printing	10/06	10/06	0%
Final report printed and turned in	11/06	11/06	100%

Figure 13: Gantt chart