HydraNet: A Network For Singing Voice Separation

Master Thesis

Esbern Torgard Kaspersen

Aalborg University
Architecture and Mediatechnology

Architecture and Mediatechnology
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
HydraNet: A Network For Singing Voice Separation

Theme:
Single-Channel Blind Source Separation

Project Period:
Spring Semester 2019

Project Group:
1

Participant(s):
Esbern Torgard Kaspersen

Supervisor(s):
Tsampikos Kounalakis
Cumhur Erkut

Copies: 1

Page Numbers: 74

Date of Completion:
May 28, 2019

Abstract:

This paper proposes a new model, called HydraNet, for solving the problem of single-channel blind source separation. The model is based on two other models called Chimera and Wave-U-Net. By combining these two it was hoped that the signal-to-distortion ratio (SDR) would increase. HydraNet was implemented in Python PyTorch, and evaluated on the DSD100 dataset for singing voice separation. It reached a SDR of 9.78dB for instrument separation and 3.46dB for singing voice separation. Chimera and Wave-U-Net were also implemented in Python PyTorch and tested on DSD100.

Contents

Chapter 1

Introduction

Source separation is a an area of research that is applicable in a wide array of other fields. The area focuses on the development of techniques whose goals are identifying, and separating, the individual signals in a mixture of signals. This is best illustrated in the "cocktail party problem", which concerns the separation human speech. This problem was first described thoroughly in a paper by Cherry [10] in 1953. It states that at a cocktail party, where many people are speaking at the same time, in every direction, it is possible for humans to clearly distinguish the speech of a single person. This is significant as there is no clear mathematical model for how this could be achieved, yet our brains are doing it with ease, with only limited information. In essence this problem states that we know a mixture of voices can be divided into its constituent parts, however we do not have a clear model for how this is done. The paper also lays out 5 factors humans might use in this process:

1. "The voices are coming from different directions"
2. "Lip-reading, gestures, and the like"
3. "Different speaking voices, mean pitches, mean speeds, male and female, and so forth"
4. "Accents differing"
5. "Transition probabilities (subject matter, voice dynamics, syntax)"

These are direct quotes from the paper, where Cherry also states that the only factor that can be worked on is number 5, as recording two statements on the same magnetic tape would be "babel". However, since then technology has improved and the field of source separation has expanded. Instead of focusing on factor 5, most modern techniques focus on factor 1 and 3, the phase and frequency content of the different source signals. These are also the focuses of this paper.

Newer, more difficult, variations of the field has also been developed as more advanced problems have been encountered. These more difficult areas are called blind source separation and single-channel blind source separation. Blind separation means that no information about the sources is available beforehand, and single-channel means that only one audio track is available, making detecting phase differences in sources difficult.

Traditionally these problems have been tackled by purely statistical methods, such as non-negative matrix factorization [15, 34, 63] (NMF), individual component analysis [3, 1] (ICA) and computational auditory scene analysis [11, 65] (CASA), to a lesser degree of success. However, in recent years many new machine learning techniques have been developed, which has allowed for more powerful models to be constructed, and for more general solutions to be found. Machine learning models have allowed for huge increases in signal-to-noise ratio (SNR) and signal-to-distortion ratio (SDR) on many speech separation tasks. Though they are not limited to speech, and can also be trained to separate other audio or other kinds of signals.

If better models can be constructed, that can generalize the solutions more, it could be very significant for not just audio separation but also a number of other fields. It could be utilized in electroencephalography (EEG), the technique used to record and locate different electrical signals in the brain[32]. In this field a large amount of sensors is placed at various locations on the participants skull, such that all record the mixture of electrical signals in the brain. The phase differences of the signals and the placement on the skull can then be used to triangulate where different signals are emitted inside the brain [9, 6, 33].

However, if some sensors are not making good contact with the skin they will return erroneous signals and potentially make the data useless. If signal separation could be applied efficiently and accurately then much fewer sensors could be used and the effects of one sensor making bad contact could be lessened if the algorithm is robust enough. Another application could be in the development of virtual reality (VR) experiences. In VR, the location of sounds is key to feeling that the environment is realistic. This means if a group of virtual characters is talking, it is important to have a source of sound emitted from each of them, so that we can take advantage of the first factor Cherry described, to distinguish them. Normally in the development of a VR experience this is manageable as the character's voices can be recorded individually. However if a group wants to create a virtual orchestra experience instead of a virtual conversation, this becomes a lot more difficult as it is few who has access to a full orchestra and the resources to record every single musician individually. It is in such instances that a source separation model could be used to separate recordings of other orchestras and thus create better VR experiences. In the same vein small game development companies, DJs or other private users, could use this technology to create novel music pieces from the separated instruments of other songs.

As discussed, source separation is a big field with many applications and a wide array of different techniques. This is why this report will begin with a chapter about the background and concepts used in the report, which will contain a more general description of these fields and theories. Chapter 3 will then describe specific state-of-the-art papers, and will end with a plan for which concepts will be used in the implementation. Chapter 4 will contain the re-implementation of two of these papers and the implementation of a new model called "HydraNet". In chapter 5 these models will be evaluated with different hyper-parameters and model structures, and the results will be shown. The results will be discussed in chapter 6, along with errors that were made and suggestions for improvements in future works. The report will then be summarized and the results reiterated in the conclusion chapter.

Chapter 2

Background And Concepts

In this chapter, some of the general and specific concepts that are utilized by the cited state of the art papers will be explained.

2.1 Source Separation

As stated in the introduction, source separation is a field with a long history and many different techniques for solving it. In this section the main concepts of the field will be explained, along with the difficulties faced by the techniques used and how those techniques are evaluated.

2.1.1 Multi-Channel Source Separation

The best case scenario for source separation techniques is having an equal amount or more signals than the number of sources one is trying to separate. This is because with more signals one gains more spatial information about the mixture, as some sources will decrease in strength in the signal while others will be amplified due to the sensor being moved closer to or further away from the source. This problem is called multi-channel source separation and can handle even noisy signals fairly well because of the extra information gained from the additional channels. However this group of methods fall short when presented with more sources than channels, and can in this case erroneously output signals from multiple sources as a single source. This case can happen if the number of sources in the signal is unknown or uncertain, also called the blind source separation problem.

2.1.2 Single-Channel Blind Source Separation

Blind source separation is one of the hardest problems to solve in the source separation field, as many techniques rely on some certainties in the signal, like the number of sources being constant. However, it is also the most true to real life, as it cannot always be expected that the number of sources will be constant. When faced with the blind source

separation problem, some methods seek to use many channels to be certain that the number of channels will always be above the number of sources [48, 66, 16, 54].This is called overdetermining the problem, i.e. adding so much information that patterns can be easily found. Though having many channels is not always possible in real-world scenarios, often only a single channel is available. This is called an underdetermined problem.
The field that studies this problem is called single-channel blind source separation [22, 34, 30, 17, 40], which, as the name suggests, seeks to find all sources in a signal based on a single channel. This is the hardest version of the source separation problem, and is even more extreme than what humans deal with. Humans have two channels that can detect phase differences in the signals they receive and help separate sources in that way, that is not an option when using only a single channel. This is why techniques in this field have to be inventive and generalize very well, and why machine learning algorithms have been solving this problem better than any other statistical methods.

2.1.3 Signal-To-Noise Ratio And Signal-To-Distortion Ratio

The main ways of calculating the accuracy of source separation techniques were proposed in [61], namely the signal-to-noise ratio (SNR), signal-to-distortion ratio (SDR), signal-to-interference ratio (SIR) and signal-to-artifact ratio (SAR). It should be noted that these metrics were not invented by the paper, though the equations that are used to calculate the metrics were. These formulas have been implemented in a blind source separation toolbox for Matlab, which was then converted to a Python library[51]. This toolbox, and thus the equations of the paper, have been used to evaluate many state-of-the-art papers and used as a standard in the signal separation evaluation campaign, SISEC[41, 59]

The paper describes that there are three main errors in a signal, the noise, the interference and the artifacts, and it is through these that the SNR, SDR, SIR and SAR can be calculated. Interference in a signal is when another unrelated signal is also present. It is the ratio of the clean signal to these other signals that SIR seeks to find. The noise is similar, as it is some other signal that is present in the generated signal, however in the paper [61] it is defined more as sensor noise, or generation noise. The paper therefore makes a clear distinction between SIR and SNR, which is often overlooked by other papers where any interference in the signal is also counted as noise [24], and measured solely by SNR. A similar thing often happens with the SAR, which measures differences in the signal that are neither from noise or other signals interfering, like sudden audio spikes that only last a couple of milliseconds or less, as those are often also simply considered noise and as part of the SNR.

The SDR measures the combined distortion of these other effects in comparison to the clean signal, and is therefore often used as a general measurement of how well the separation is working.

The paper says that if we have j sources s_j and j approximated sources $\hat{s}_j$ then to calculate the SNR, SDR and SAR we need to project the approximated sources into the

space spanned by the real sources. This is done with an orthogonal projection matrix, which is a matrix that is used to find the closest point on some shape from some arbitrary point in the same space. The simplest version of this is projecting a point in 2D space onto a 2D line, or a 3D point onto a plane. They denote making such a matrix with using k vectors with the $\Pi\{y_1, ..., y_k\}$. To get this matrix from the k vectors this calculation is done.

$$P = A(A^T A)^{-1} A^T \tag{2.1}$$

where A is a N×k matrix, with N as the length of each vector, consisting of $y_1, ..., y_k$. P is then an N×N orthogonal projection matrix. The paper states that to calculate the interference, noise and artifacts, three of these orthogonal projection matrices need to be constructed..

$$P_{s_j} := \Pi\{s_j\} \tag{2.2}$$

Is the projection matrix to the space spanned by s_j.

$$P_{\mathbf{s}} := \Pi\{(s_{j'})_{1 \leq j' \leq n}\} \tag{2.3}$$

Is the projection matrix to the space spanned by all the clean sources.

$$P_{\mathbf{s},\mathbf{n}} := \Pi\{(s_{j'})_{1 \leq j' \leq n}, (n_i)_{1 \leq i \leq m}\} \tag{2.4}$$

Where **n** is all the known noise sources, such as known microphone noise or generation noise. These matrices are then used to calculate the four parts that make up $\hat{s}_j$, which were discussed before, interference, noise and artifacts, along with the clean source signal.

$$\hat{s}_j = s_{target} + e_{noise} + e_{interference} + e_{artifact} \tag{2.5}$$

$$s_{target} := P_{s_j} * \hat{s}_j \tag{2.6}$$

This projects $\hat{s}_j$ onto s_j, and is essentially a N dimensional point to line projection. This projection shows how much of $\hat{s}_j$ stems from s_j, and is therfore what we want to maximize.

$$e_{interference} := P_{\mathbf{s}} * \hat{s}_j - P_{s_j} * \hat{s}_j \tag{2.7}$$

This compares the projection of $\hat{s}_j$ onto the single source s_j, to the projection of $\hat{s}_j$ onto the hyper-plane spanned by all the sources. If there are other signals interfering in $\hat{s}_j$ the point on the plane will land closer to those sources, while if there are no other signals in $\hat{s}_j$ then the points will project to the exact same place and this will go to 0. This is also smart as it can show which sources are interfering the most.

$$e_{noise} := P_{\mathbf{s},\mathbf{n}} * \hat{s}_j - P_{\mathbf{s}} * \hat{s}_j \tag{2.8}$$

This projection compares the two hyper-planes the one spanned by the clean sources and the one spanned by both clean sources and noise. By projecting $\hat{s}_j$ onto both hyper-planes the direct contribution of the noise sources to $\hat{s}_j$ can be compared accurately without

considering the interference measurement. If the equation had instead looked like this: $P_{\mathbf{s},\mathbf{n}} * \hat{s}_j - P_{s_j} * \hat{s}_j$, then both the interference from other signals and the noise would have been measured, and contribution of each would not be able to be calculated.

$$e_{artifact} := \hat{s}_j - P_{\mathbf{s},\mathbf{n}} * \hat{s}_j \tag{2.9}$$

This measures the difference between $\hat{s}_j$ and $\hat{s}_j$ projected onto the hyper-plane spanned by all the clean and noise sources. This means that any remainder of this operation is something in $\hat{s}_j$ that does not stem from the clean sources or the noise sources, making it an artifact. After these measures have been calculated they are used to calculate SDR, SNR, SIR and SAR And these measures are used to calculate the SNR, SDR and SAR

$$SDR = 10\log_{10}\frac{||s_{target}||^2}{||e_{interference} + e_{noise} + e_{artifact}||^2} \tag{2.10}$$

As said before, this measures the overall distortion of the signal, from all the error sources, compared to the amount of the clean signal is in $\hat{s}_j$.

$$SNR = 10\log_{10}\frac{||s_{target+e_{interference}}||^2}{|e_{noise}||^2} \tag{2.11}$$

As said, this paper makes a clear distinction between SNR and SIR, which is why interference is in the nominator for this equation. The objective is to calculate how much noise there is in the signal compared to all clean sources, not compared to the correct clean source. This is also because of how noise is calculated in equation 2.8, the point in clean space compared to the point in noisy and clean space. If interference was not included in the SNR calculation then there might be some distortion from how the noise is calculated.

$$SIR = 10\log_{10}\frac{||s_{target}||^2}{|e_{interference}||^2} \tag{2.12}$$

As can be seen SIR is calculated only based on the interference from other signals, without any consideration for noise.

$$SAR = 10\log_{10}\frac{||s_{target+e_{interference}+e_{noise}}||^2}{||e_{artifact}||^2} \tag{2.13}$$

The reason the interference and the noise are in the nominator of this equation is the same reason as for the SNR calculation, if they were not then there might be some distortion in the result due to them being left out.

As can be seen, these measures are separated well and are therefore very powerful when analysing the results of an algorithm. However, noise sources are not always known, and because of that the $P_{\mathbf{s},\mathbf{n}}$ matrix cannot be constructed and e_{noise} and $e_{artifact}$ cannot be

calculated separately. This is why some papers[42, 66] define scale-invariant SNR (SI-SNR) as SDR. This measure is calculated as

$$e_{noise} = \hat{s} - s_{target} \tag{2.14}$$

$$SI - SNR = 10log10(\frac{||s_{target}||^2}{||e_{noise}||^2}) \tag{2.15}$$

where s_{target} is the same as before. This measure is a combination of the e_{noise} and $e_{artifact}$ calculations before. By simply comparing the estimated signal to the signal projected onto the clean signal, this measures all the three error sources, interference, noise and artifacts, as one.
Often in practical implementations of this, s_{target} is defined as

$$s_{target} = \frac{< \hat{s}, s > s}{||s||^2} \tag{2.16}$$

which is another way to write the projection of $\hat{s}$ onto s. And as can be seen this measure is very similar to SDR when considering that e_{noise} covers for both noise, interference and artifacts.

2.2 Neural Networks

Neural networks are powerful statistical models, which are used to find complex mapping or classification functions[7, 21, 52]. A common example of this is modelling the likelihood of a person voting on a certain party based on their income, age, gender and ethnicity. These values individually does not tell much about political orientation, however if they are all given to a neural network it can find patterns in them that can predict voting patterns. A neural network consists of nodes, called neurons or units, that each do calculations based on their input. The most common calculation these neurons are used for is a weighted sum of their input. This allows the network to adjust how much each neuron is affected by a certain input, like the income of a person, by adjusting the weight for that input in the neuron. In this way the appropriate weight can be found such that the network accurately predicts the persons political orientation. The calculation for the output of a neuron looks like this

$$x_j^l = \sigma(b + \sum_{i=0}^{n} w_i x_i^{l-1}) \tag{2.17}$$

Where x_j^l is neuron j in layer l, n is the number of neurons in layer *l-1*, w_i is the weight for the connection to the *ith* neuron in layer *l-1*, x_i^{l-1} is the output of the *ith* neuron in layer *l-1*,

b is a bias and σ is some activation function. Such a function could be a logistic sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.18}$$

which converts its input to a range between 0 and 1. Another useful function is the softmax function which measures the ratio of the outputs of the neurons in the layer.

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_{i=0}^{n} e^{x_i}} \tag{2.19}$$

These equations are useful as they regularize the output of each neuron to some known range.

For the adjustment of the weights, an algorithm called backpropagation [23] is often used, where a "loss" is calculated and propagated backwards in the network. The loss is a function that in some way tells how wrong the network's output is. This can then be used to go back through the network and calculate how much each neuron and each weight has contributed to this loss, and the weights that have contributed the most wrong information can be adjusted to contribute less. The weights are adjusted based on the negative gradient of this loss function at that neuron, which will move it towards a value that will provide the least amount of loss at the output. To calculate these gradients, a partial derivative is used. The partial derivative is taken of the network as a whole with regards to a single weight, which can be done because the network in essence is nothing more than a very complicated equation, with each of its parts being a neuron or a weight.

There are many ways of constructing a neural network, however the most common way is to use "layers" of neurons, that are not connected within the layer but rather connected to all the neurons in the layers before and after. These layers are often referred to as dense layers, as the connections between these layers often look dense when visualized. However, dense layers are not the only layers that exist, and they are not the best for every task.

2.2.1 Recurrent Neural Networks

Recurrent neural networks (RNN), and layers, function similarly to dense layers, in that they are often densely connected to the layers in front and behind them [4, 12]. However, they differ in that they get their previous output as an input. This gives the layer a temporal axis where inputs are no longer separate, but rather seen as a chain of connected events. An example of this could be predicting population size of animals, where the population of the previous year is connected to the population next year, and thus knowledge of all previous years will inform what the population could be next year. Thus these layers are very good at predicting sequences of inputs, also sometimes call time-series.

However, these networks come with a major disadvantage compared to traditional dense layers, when it comes to backpropagation. The problem is that by having the layers connected in time, the error of a single neuron is no longer only depending on its connections to the previous layer but also its own output last sample, and that is dependant on the sample before that and so on. This creates a very long chain that the error has to be propagated through, and in this process the error almost vanishes due to how it is transferred back through the model. This is called the vanishing gradient problem [27, 71], and it has been tackled in many different ways. One of the most popular ways is called the Long Short-Term Memory layer (LSTM).

2.2.2 Long Short-Term Memory Layers

The LSTM "cell" was proposed by [18] as a way to combat this problem of vanishing gradients. It is called a cell because it is not a single neuron, but rather a series of connected neurons that have their own weights and purpose. The LSTM adds a "cell state vector", which does not contain any weights or connections, only values from the previous state of the cell. However, the current state of the cell is then modified by the input to the cell, the previous output and an intermediate value in the cell. This vector is then used to modify the output of the cell. The cell state vector makes it possible for the cell to store information inside itself, and enables it to forget things about previous inputs and states, if they are not relevant to the current problem. This alleviates the vanishing gradient problem as only information that is very relevant will be kept for a long time, while other information is stopped and thus the number of states the backpropagation algorithm has to go through is significantly smaller. The structure of the LSTM can be seen in figure 2.1. The mathematical expressions for an LSTM look like this.

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{2.20}$$

Where i_t is the output of the input gate, x_t is inputs at time t, W_{xi} is the weights for connection from x to i, h_{t-1} is the output of the hidden vector at time t-1, c_{t-1} is the cell state at t-1 and b_i is the bias for the gate.σ is the logistic sigmoid activation function described in equation 2.18. In the following equations all the variables and subscripts follow these conventions.

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \tag{2.21}$$

Which is the forget-gate of the cell, which allows it to regulate what it remembers in the cell state, which functions like this.

$$c_t = f_t c_{t-1} + i_t tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{2.22}$$

Where *tanh*is an activation function

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \tag{2.23}$$

Which is the output of the gate that regulates the output of the network

$$h_t = o_t tanh(c_t) \tag{2.24}$$

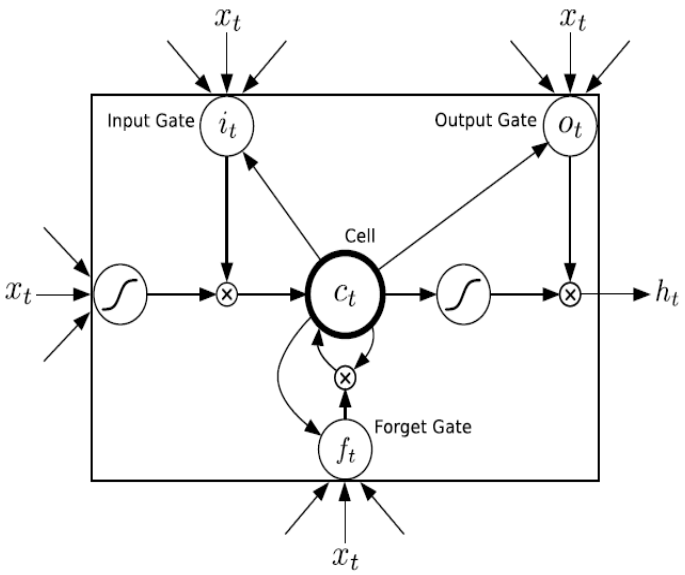

Figure 2.1: A LSTM cell, taken from [18]

Even with the advantages LSTMs provide, they still have a problem which also plagues other RNNs. This problem is that they only process a sequence in one direction, forward, while there might be connections bacwards in the sequence too. The classic example would be analyzing a text. There might be crucial information at the end of the text that will inform the meaning of the words in the beginning. However, traditional RNNs will only process the text once and will not go back to the beginning and change their predictions based on the end of the text. This is the problem bidirectional RNNs seek to solve.

2.2.3 Bidirectional RNN

A bidirectional RNN (BRNN)[55] is, as the name implies, a RNN that processes the input in both directions. This has the advantage that the output will take into account the temporal connections in both directions, and will therefore often be more powerful than regular RNNs. The downside is that it requires double the amount of units as a normal RNN to produce the same size output in either direction. What is meant by this, is that a standard RNN with 20 units will produce an output of $\mathbb{R}^{T\times 20}$, where T is the number of samples in the time series, while a BRNN with 20 units will produce an output of $\mathbb{R}^{T\times 40}$. This is because a BRNN with 20 units actually has 40 units, 20 for the forward direction and 20 for the backwards direction. When two BRNNs are connected this can

become a bigger problem if the input is large. If two BRNNs with 200 units are connected then the first BRNN will give the next BRNN an output of $\mathbb{R}^{T\times 400}$. This BRNN will also have 400 internal units that should all be densely connected to the previous layer, meaning that $400 \times 400 = 160000$ connections are made between the two BRNNs as opposed to the $200 \times 200 = 40000$ connections that would have been made between to RNNs or dense layers. This is a quadrupling in connections for the connected BRNNs. This is why BRNNs can be very memory intensive, however they are also immensely powerful. Any type of RNN can be made bidirectional, even LSTMs in which case they are called BLSTMs.

As discussed RNNs and BRNNs are powerful layers that can analyze time-series and find connections that traditional dense layers cannot, however sometimes the input can also have local dependencies without being a time-series. This is best seen in images where neighbouring pixels often share information, but pixels in opposite corners of the image might not. In this case RNNs are not the best layers to use, though a layer called a convolutional layer has been developed to handle this problem.

2.2.4 Convolutional Networks

Convolutional neural networks take inspiration from the field of image processing, by adopting the concept of kernels or filters[38, 37]. These kernels are in essence matrices of weights that are passed over each pixel in an image and the output is the weighted sum of the pixel itself and the pixels around it according to the kernel. If a 3×3 kernel is passed over an image, and used on a pixel at (x = 10, y = 3), where x and y are pixel coordinates, then the kernel will produce the weighted sum of the pixels at indexes {(9, 2), (10, 2), (11, 2), (9, 3), (10, 3), (11, 3), (9, 4), (10, 4), (11, 4)}. This notion of using a weighted sum to find features, and produce an output, is the same as for neural networks. In a convolutional network the kernel is represented as connections between neurons. This means that the layers are not densely connected, a neuron in one end of the layer does not get an input from a neuron in the other end of the previous layer. Because convolutional layers, as an idea, came from image processing most are 2 dimensional layers. This means that the neurons are no longer in a straight line as in the dense layers, but that they are structured as a grid, with an x and y coordinate. just like an image. Thus if a convolutional layer has a 3×3 kernel, then the neuron at index (10,3) will be connected to the neurons in the previous layer at indexes {(9, 2), (10, 2), (11, 2), (9, 3), (10, 3), (11, 3), (9, 4), (10, 4), (11, 4)} just like the kernel was before.

However, convolutional layers can also suffer from the same problems faced in image processing, namely border conditions. If a neuron lays on the border of its layer, and the previous layer is equally big, then some of its connections might lay outside of the border of both layers, i.e. there are no neurons to connect to. One option is to simply disregard these border neurons, however that will shrink the image over time, since the edges will always be disregarded. Another way this is solved is by using zero-padding, where a line

of zeros is appended to all the edges of the previous layer, and thus the border neurons simply get these zeros as input.

One other thing that convolutional layers can manage easily is what is called channels. Channels are different versions of an image that store different information, for example the red, green and blue channels of a png image. These all store the individual colour values for the image, and should therefore be processed individually by the convolutional layer. The layers themselves can also output multiple channels, also called feature maps, to generate different outputs. In these cases a neuron in a convolutional layer will have connections to the neurons in its kernel, in all the channels of the previous layer.

Convolutional layers can also reduce the size of the input to the output by having a "stride" that is higher than 1. The stride determines where a neuron in the current layer is connected to a neuron in the previous layer based on its position. If a layer has a kernel of 1 and a stride of 1, then a the neuron at (0,0) will be connected to the neuron at (0,0) in the previous layer and the neuron at (1,1) will be connected to the neuron at (1,1) in the previous layer too. However if the kernel is 1 and the stride is 2 then (0,0) will still be connected to (0,0), but (1,1) will be connected to (2,2). This means that a stride of 2 will halve the image, while a stride of 3 will make the image a third as large as originally. This is very useful for reducing the size of the image very quickly and keep the memory usage down, but it is also useful for generalization. By reducing the amount of information so much between each layer, the network is forced to find a good generalization of the input to make the output accurate.

It should also be noted that convolutional layers can also be 3 dimensional and 1 dimensional, it is simply a matter of removing or adding connections and neurons. 1 dimensional convolutions are useful in some cases where input is 1 dimensional but too large to let a dense layer handle, or it is desired to find local features in the input. As said before they can also be used to reduce the size of the input very quickly and find a more abstract representation of the input in a smaller amount of space. This is also the idea behind a type of network called an auto-encoder.

2.2.5 Embedding Spaces And Auto-Encoders

Auto-encoders are neural networks that seek to create an embedding space, such that the original input can be reconstructed from that space[2, 50, 26, 62]. An embedding space, also called latent space, is a space that is an abstract representation of the input, such that useful information from the input is somehow represented in the space while non-important information is discarded. These spaces are often what are created in neural networks in some way, though auto-encoders actively seek to create them in the best way possible. An embedding space could be binary values, where when a 2 is transferred to the embedding space, it becomes a 01. The information is still retained, but it is transformed in some way. This transformation also often means a reduction from the size of

the input to the size in the embedding space. Maybe an input of 10 values will be reduced and represented by 5 values in the embedding space.

An auto-encoder tries to create these embedding spaces by using a funnel shaped network. This network can consist of dense or convolutional layers, that at each layer in the network reduce the size of the output. When the input has been reduced enough the embedding space has been created, and the decoding can begin. The decoder takes the information in the embedding space and passes it through a reverse funnel network, where at each layer the size of the output increases. This is done until the output of the last layer is the size of the original input. The goal of the decoder is to reconstruct the input based on the embedding space.
This type of network is very useful when working with generation algorithms, as after the network is trained, random values in the embedding space can be given to the decoder and it can generate novel images. If an auto-encoder has been trained to produce animals then the locations of a horse and a dog can be found in the embedding space, and can then be smoothly interpolated between to produce a horse dog hybrid. In practical use this could also handle music and interpolate between a Bach piece and a Mozart piece to create a novel symphony, though this is not as easy as the animals example.

2.3 Clustering

Clustering is a statistical technique operating on some $\mathbb{R}^N$ valued dimensional space, where N is some integer. Given a set of points in this space, clustering methods will put the points into some sub-sets, or clusters, based on some metric and criterion. The most common metric to use is the euclidean distance between two points, though other distance metrics and criteria can be used too. The distance measure can be used in different ways to create the clusters. One way is called agglomerative clustering.

2.3.1 Agglomerative Clustering

Agglomerative clustering starts with the assumption that all points are their own individual "clusters", and is given some distance threshold parameter to evaluate by. The algorithm will then take the two clusters with the lowest inter-cluster distance and then merge them into a bigger cluster. This will continue until there are no two clusters with an inter-cluster distance lower than the distance threshold parameter given in the beginning. If this distance threshold is high in comparison to the maximum distance between two points then it is highly probable that all points will be in clusters containing at least 2 points. However if the distance threshold is relatively low then there is a high probability that some points may never be clustered with others. This can be a good thing if finding outliers in the data is desired, but bad if one wants to find connections between all points in the set.

2.3.2 Divisive Clustering

Divisive clustering algorithms start with the assumption that all points belong to a single cluster, and it is also given some distance threshold parameter to evaluate by. The algorithm then finds the point inside the cluster that has the highest distance to the rest of the points of the cluster, and splits it into its own cluster. When this is done a sub-algorithm is used to cluster the found point with all other points in the other cluster that are closer to it than to the cluster itself. This is then redone until the maximum distance between any two clusters reaches the distance threshold parameter. This ensures that most points will always stay inside clusters with other points, unless the distance threshold parameter is lower than the minimum distance between any two points in the set.

2.3.3 Metrics And Criteria

There are multiple different ways of evaluating the distance between two clusters, and they will change the final clusters gotten from the algorithm. Four of the most commonly used metrics are Ward linkage, Complete linkage, average linkage and single linkage.

Ward Linkage

This type of linkage criteria uses the sum of all squared distances between all points of both clusters. This has the effect of clustering clusters that are generally very close together and will not cluster clusters that only have a few points that are far apart.

Complete Linkage

This linkage type uses the maximum distance between points in two clusters to determine if the clusters should be merged. With this approach two clusters that have close means but high variances will not be clustered together as easily, as there might be some points in the clusters that are very far away from each other due to both having high variance. This also allows the distance threshold to be set higher than other methods as the general distances will be higher.

Average Linkage

Average linkage uses the distances between the averages of two clusters to determine if they should be clustered. If the data is uniformly distributed this will be good at finding clusters, however if there are some dimensions where more variance exists then this method might incorrectly cluster some points. Theoretically this method will cluster in a spherical manner, however clusters will often have straight edges where they meet other clusters as the distance from the points to the mean of either cluster will be very similar in that area.

Centroid Linkage

The centroid of a cluster is the point that is closest to the mean of the cluster. Usually using average linkage instead of centroid linkage yields similar results, however if the data contains torus, also called donut, shapes then this method will not act similarly to average linkage, and instead try to split up parts of the toruses into clusters, instead of clustering the entire torus as one.

Single Linkage

Single linkage algorithms take distance between the closest points in two clusters and uses that as the criterion for clustering. This has a very good effect when the data has clearly separated clusters within it, however in uniformly distributed data the method tends to favor only a few clusters and might just cluster the entirety of the data set together.

2.3.4 K-Means

K-means clustering is an agglomerative algorithm that makes use of both average and centroid linkage to cluster n points into k clusters. It does this by putting k random "mean points" into the point space and then clustering the points to the mean closest to them. After doing so k clusters are achieved, however due to the random placement of the initial mean points in the space, these clusters cannot be assumed to have found any meaningful structure in the data. Because of this the process is repeated, however the centroid of each cluster is now used as the new mean of that cluster, instead of using a random point. Points from other clusters are then aggregated based on the new mean of the clusters. This process continues until all points within each cluster are closer to that clusters centroid than to any other clusters' centroid.

2.4 Permutation And Output Dimension Mismatch Problem

The permutation problem is a problem that is often encountered in blind source separation, as methods can sometimes provide the predicted sources in an unknown order. This is a problem as it makes comparing generated sources and clean sources a guessing game, for which two sources are supposed to be the same. The output dimension mismatch problem stems from the rigidity of neural networks, and arises when there is an unknown amount of sources in the signal, as in blind source separation. In this case the network is stuck with a fixed amount of outputs and cannot dynamically adapt to the number of sources, thus making accurate separation of all the sources impossible. This is fixed by [24] by simply creating an embedding space as the output of the network, which can then be analyzed by clustering algorithms such as k-means to find the correct number of sources.

2.4.1 Permutation Invariant Training

The permutation problem can be solved in a couple of different ways. One way is to use permutation invariant training (PIT) as proposed by [72]. The method computes all the ways the estimated sources and clean sources could be compared, and then evaluates those ways using MSE. The combination of estimated and clean sources, with the least MSE is then used for the loss.

2.5 Fourier Transforms, Spectrograms And Scales

A Fourier transform is a function that converts a real-valued signal into a complex number, representing a given frequency's strength and phase in the signal[14]. The original function concerned continuous signals.

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-i2\pi ft} dt \tag{2.25}$$

where f is a given frequency, and $x(t)$ is the magnitude of the signal at time t measured in seconds. This will work for any given continuous signal, and if repeated for all continuous frequency values a spectrum will be obtained. However, it is impossible to store a continuous signal in a computer, it has to be discretized in some way to fit in a RAM module or on a hard disk. To handle these kinds of signals the discrete time Fourier transform (DTFT) was created.

2.5.1 Discrete Time Fourier Transforms

To support discrete time-steps the Fourier transform in equation 2.25 had to be changed. Time is no longer measured in continuous fractions of seconds, but in integer samples instead. This also means that a continuous signal has to be sampled before a DTFT can be applied. To do this, the magnitude of the continuous signal will be measured at points with some set interval, called the sampling rate, and these measured values will be the samples for our discrete signal. After this is done the DTFT can be applied to the signal and a spectrum can be obtained. The DTFT equation now looks like this

$$X(f) = \sum_{n=0}^{N-1} x[n] e^{-i2\pi fn} \tag{2.26}$$

where N is the number of samples in the signal and $x[n]$ is the value of sample n in the discrete signal. The advantage of this approach is that it does not require an infinite continuous signal, however the disadvantage is that its frequency detection is less powerful the lower the sampling rate is. The Nyquist theorem[49] states that for a sampling rate of N the maximum frequency that can be detected accurately is $N/2$. This means that for a sampling rate of 1000 Hz only frequencies up to 500 Hz can be reliably detected. It also means that we have to choose a sampling rate for the signal, which will capture the

frequencies we are interested in.

Another difference between the DTFT and the original continuous Fourier transform is that f is no longer treated as a continuous value, but is rather a discrete value too. Rather than capture all possible frequencies up to the nyquist limit, only key frequencies are measured, and frequencies close together are "binned". This means that the strength of a 100 Hz frequency, a 100.5 Hz and a 99.5 Hz frequency in a signal, might be added together and put in a bin labeled 100 Hz even though two of them are slightly above and below that frequency. In this case 100 Hz is called the center frequency of the bin, each bin will have some width that they span over, which depends on the number of bins and the maximum measurable frequency. If a signal is sampled at 10000Hz and a 1000 bins are used to describe this range, then each bin will have a center frequency that are 10Hz apart and a width of 5Hz in both directions. This ensures that the entire range is covered. However, it also loses information about the precise frequencies, though if more precision is desired more bins can be used and the range can be more accurately divided.

This discrete time Fourier transform enables digital Fourier transforms to be done, which is very useful, however it still has one major problem. It only measures all the frequencies for all of the signal, and does not tell when in the signal those frequencies occur. This kind of information might be very useful if it is desired to know when a person starts and stops speaking, or when a certain instrument plays, as it would allow for the filtering of these events when they happen. This problem is what the short time Fourier transform (STFT) seeks to solve.

2.5.2 Short Time Fourier Transforms And Spectrograms

The short time Fourier transform solves the problem of locality in time by analysing the signal multiple times with a sliding window function applied. This window function's purpose is to diminish the signal around the area that is being analyzed, but still retain the signal within that area. There are many different window functions that can be used, and which all have their separate advantages and disadvantages. However they all have the same objective, to enable analysis of a short portion of the signal. The equation for STFT is almost the same as the DTFT function, but with the addition of the window function.

$$X(f) = \sum_{n=0}^{N-1} x[n]w[n-m]e^{-i2\pi fn} \tag{2.27}$$

Where *w[n-m]* is the value of a window function at *n-m*, with the center at m and an unknown window length. The window length is the number of samples that should be inside the window. This function is applied to the signal iteratively, meaning that the function is applied with a window at m, the result is recorded, m is then incremented and the process is done again. This is repeated until the entire signal has been processed. It should be

noted that m is often incremented by less than the window length, between each iteration. This creates some overlap between the fourier transforms, which has the advantage of recording the gradual change in frequencies over the signal instead of sudden shifts as would be obtained with an increment above the window length. The increment is often called the hop length.

By having this shifting window, and iterative process, a spectrum for each window in the signal is gained. When combined into one plot of time and frequency this is called a spectrogram. Spectrograms are a very used tool when analyzing music as they can be used to give a visual representation of the signal that is easily understandable, and where artifacts in the signal can be identified. However, sometimes the 1024 bins, which is the default for most STFT implementations, are just too much when a program has to analyze them. The answer is often to reduce the number of bins, though this is not ideal either in some cases, as a lot of important information might be stored in the low frequency bins, while irrelevant information might be stored in higher frequency bins. In this case decreasing the amount of bins would expand the bin width equally for all bins, and therefore might group together important low frequency information that is important while not reducing the irrelevant high frequency information enough. This is the problem frequency scales seek to handle.

2.5.3 Frequency Scales And Filter Banks

A frequency scale is a function that describes what the center frequencies of each bin should be. As discussed the STFT uses a linear frequency scale by default, increasing the center frequency of each bin by a set amount between each bin. However, a popular scale to use when handling human voices is the mel frequency scale[64]. This scale is logarithmic as it tries to emulate the human perception system, as it too seems to be logarithmic. This logarithmic approach means that lower frequencies are given smaller bin widths, increasing the resolution of the frequencies in these sections, while higher frequencies are given much wider bin widths, grouping together information that is harder to distinguish for humans. After this transformation the spectrogram is called a mel spectrogram, where mel can be substituted for the name of the applied frequency scale.

In practise this filter is applied with filterbank, a set of M filters, where M is the number of desired bins after the transormation. This filterbank is often calculated as a matrix of the filters, such that a matrix multiplication between the spectrogram and filterbank can be done. This can also be undone by applying the inverse of the filterbank to the mel spectrogram. However this process is lossy in the sense that all the information of the original spectrogram cannot be recovered this way. This is because a given bin in the mel spectrogram is the weighted sum of the values in the original spectrogram, however it does not know which values were originally populated. An example would be a filter being applied to these three bins with values {3,0,1} and with a filter {0.5, 1, 0.5}. The resulting bin will

contain the value 3*0.5 + 0*1 + 1*0.5 = 2. When the inverse filter (in this example we just reuse the same filter) is applied it will populate all the three bins with these values {0.5*2, 1*2, 0.5*2} = {1, 2, 1}. As can be seen this operation does not consider that some bins will be unpopulated in the orignal spectrogram, and the spectrogram produced by the method can therefore have many artifacts. However, this is an issue that is not easy to solve and is outside the scope of the project, though it is important to mention it.

Once a spectrogram, or mel spectrogram, has been created there are many different methods of processing it and retrieving information from it. One of those methods is called spectral clustering.

Chapter 3

Analysis

3.1 Deep clustering: Discriminative embeddings for segmentation and separation

This paper [24] tries to handle the difficult problem of blind single-channel speech separation. Their approach is to combine neural networks and clustering to assign the bins of a spectrogram to the speaker who "dominates" that bin. They call this "deep clustering". Specifically this works by first analyzing the sound using STFT and then fitting that spectrogram to the mel scale to reduce the dimensionality of the frequency bins. The STFT uses a square root Hann window with a window length of 32ms and a hop length of 8ms. The log of the magnitude of each bin is also taken and used instead of the magnitude in the spectrogram. During training the spectrogram is passed through a threshold filter which multiplies all bins under the threshold by 0 and all others by 1. This reduces noisiness in the spectrogram and reduces the risk of noisy miss-classification during training time.

The reduced spectrogram is then split up into 100 frame segments that do not overlap which are then used as input for a neural network. This neural networks consists of two 600 unit BLSTM layers, one used as the input layer and the other as an intermediate layer, and a dense layer as the output layer with N units, where N is the desired embedding dimension. This step creates an N dimensional embedding for each time-frequency (T-F) bin which is then used by the clustering algorithm. The goal is to cluster all bins that belong to each speaker into separate clusters. The algorithm used during training is k-means, however any clustering algorithm can be used during testing. After this has been done for all the 100 frame segments, the algorithm is done and the separated bins can be used to reconstruct the audio in separate channels. This is done using the phase information from the original spectrogram and doing an inverse STFT with that and the separated bins. However during training this introduces the permutation problem when trying to calculate the loss of the model as the clusters are not in any particular order and therefore it is unknown which cluster corresponds to which original source. However only

two and three sources are used for training and therefore only two and six permutations respectively are needed to find the right cluster source match.

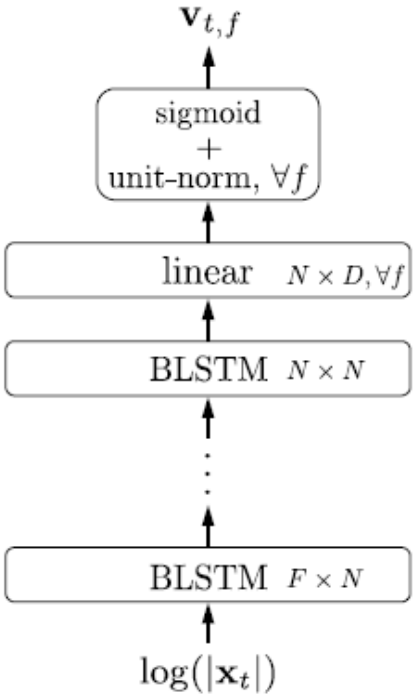

Figure 3.1: The deep clustering model, taken from [66]

The network outputs an embedding matrix the paper calls $V \in \mathbb{R}^{FT \times D}$, where FT is the frequency and time, and D is the embedding dimension, this is the matrix that is used for clustering. The algorithm also has access to an ideal binary mask $Y \in \mathbb{R}^{FT \times C}$, where FT is still frequency and time, but C is the number of sources. Because the two matrices are of different dimensions on one axis they cannot be multiplied directly to get a loss function, they first need to be made into their affinity matrix forms, A and $\hat{A}$. This is done like this

$$A = Y * Y^T \tag{3.1}$$

$$\hat{A} = V * V^T \tag{3.2}$$

This makes both A and $\hat{A}$ into $A \in \mathbb{R}^{FT \times FT}$ and thus they can be multiplied to make a cost function for the network, which the paper calls $C_Y(V)$. The paper states that turning the Y into an affinity matrix also has the added bonus of making it permutation invariant, meaning the permutation problem is solved during training.

$$C_Y(V) = ||\hat{A} - A||_F^2 = ||V * V^T - Y * Y^T||_F^2 \tag{3.3}$$

Where the $||A||_F^2$ of a matrix is the squared Frobenius norm. The paper also proves that this calculation can be further reduced by considering that the problem is low-rank since Y is very sparse. This can improve the performance very much since constructing the $A \in \mathbb{R}^{FT \times FT}$ is no longer necessary and can instead be a $A \in \mathbb{R}^{D \times D}$ for the V matrix and the Y matrix can be $A \in \mathbb{R}^{C \times C}$, where both C and D are significantly smaller than FT. This conjecture leads to a cost function in the form of

$$C_Y(V) = ||V^T * V||_F^2 - 2 * ||V^T * Y||_F^2 + ||Y^T * Y||_F^2 \tag{3.4}$$

This function bypasses the miss-match in dimension between V and Y by only summing their Frobenius norms, and only multiplying them along their shared FT axis. As said before this formulation leads to dense low rank matrices that adequately capture the sparse structures of V and Y.
An efficient gradient calculation is also derived from equation 3.4

$$\frac{\partial C_Y(V)}{\partial V^T} = 4 * V(V^T * V) - 4 * Y(Y^T * V) \tag{3.5}$$

These equations, 3.4 and 3.5, are the loss and gradient calculations used for the training of the model in the paper.

The model is trained on the WSJ0 dataset and reaches a SDR of 6.61dB. The SDR calculation used is from [61], seen in section 2.1.3. Before training, all sounds in the dataset are downsampled to 8KHz. Two or more words from the two same speakers are found and mixed together at -5dB to 5dB SNR, before being processed by the spectrogram process described above. For the implementation of the network, the paper uses the CURRENNT[70] python library.

3.2 Single-Channel Multi-Speaker Separation using Deep Clustering

This work [30] is an extension of [25, 24] and suggests some improvements that can make the algorithm better as a whole. The main improvements come in the training procedure of the model, where the paper suggests that global mean-variance normalization is done on the STFT spectrogram as a preprocessing step before using it as input to the model. They also suggest that a curriculum learning-like aproach is used where the model is first trained with 100 frame windows of the spectrogram as in the original paper and then trained with 400 frame windows. This will increase the temporal resolution of the network without increasing training-time significantly. The third suggestion is that dropout is used in the BLSTM layers to allow for a higher initial learning rate while preventing overfitting. However this is a known problem in LSTM's, that they do not respond well to traditional dropout methods as it can interfere with the memory module. The way this is avoided is to only use dropout on specific connections within the LSTM cells, like output and input channels, and in this way the memory module of the cell is not disrupted[47, 73].

This paper also defines SDR as scale invariant SNR, which is defined in equation 2.15.

Another suggestion by the paper is not to the training procedure but to the model itself. This involved doubling the amount of BLSTM layers to 4 however halving the amount of nodes in each layer to 300. This was tested against the old model and Another suggestion by the paper is not to the training procedure but to the model itself. This involved doubling the amount of BLSTM layers to 4 however halving the amount of nodes in each layer to 300. This was tested against the old model and an improvement, of 8.6dB SDR, was found in same-gender speech separation, in comparison to 7.4dB of the old model. However no difference was found in different gender separation was achieved. The paper also implements soft k-means clustering which uses a similar formula, equation 3.6 , to the softmax function used in neural networks.

$$\gamma_{i,c} = \frac{e^{-\alpha|v_i - \mu_c|^2}}{\sum_{c'} e^{-\alpha|v_i - \mu_{c'}|^2}} \tag{3.6}$$

Where i is the index of a point, c is a cluster index, α is some scalar, μ is the estimated mean of cluster c and v_i is the i'th datapoint of v. This soft k-means is done to get more adjustable assignments of each T-F bin, as the old method is completely binary and non-adjustable.

The last improvement the paper proposes is an end-to-end approach where another neural network is used instead of the clustering step. This network consists of two 300 unit BLSTM layers and a dense layer with a softmax actiavation function and N units, where N is the number of sources in the audio. This approach sacrifices the versatility of the unsupervised clustering, but gets better accuracy when the amount of sources is known in return.

This paper does not have a figure of the model, however much of the structure is the same as seen in figure 3.4. Only the mask inference head in the figure is not part of this mode, and the input has two or more channels instead of one as in the figure.

3.3 Deep clustering with gated convolutional networks

This work [39] builds on [25] and [30] by using gated convolutional layers instead of the BLSTM layers and the dense layer. Gated convolutional layers differ from regular layers in that there is a second layer which also processes the input, which is called the gate. This layer functions like the memory module of a LSTM, by regulating what information is important and should be passed on. This gate uses a sigmoid activation function which allows it to produce 0 values in neurons that are not important. Its output is then multiplied with the output of the regular layer and the product is passed to the next layer. This can be seen in figure 3.2. The σ in the figure, is the sigmoid activation function, W is the weight kernel of the layer, b is the bias, * is the convolution operator and $\otimes$ is a

element-wise multiplication.

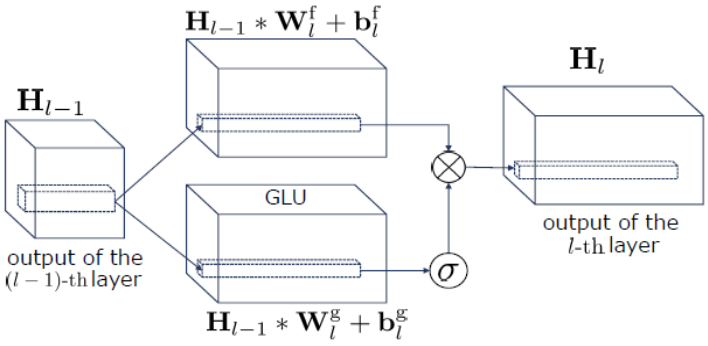

Figure 3.2: A gated convolutional layer, taken from [39]

The paper evaluates multiple different versions of the model, a bottleneck structure, like an auto-encoder, and a dialated convolution structure. It also tests 1D and 2D convolutions to see which produce the best results. It found that 2D convolutions with a dialated structure were the best, however they did not reach the same SDR levels as [24] or [30].

3.4 Multi-Channel Deep Clustering: Discriminative Spectral and Spatial Embeddings for Speaker-Independent Speech Separation

This paper [66] seeks to improve the deep clustering performance by moving away from single-channel separation, and adding more channels. The paper states that adding more channels will provide additional phase information that can be used to distinguish sources more easily. However, the paper does not simply give the extra channels as input to the network, instead a series of spatial features are extracted from the channels and given to the network along with the mel log magnitude spectrogram of one channel. The features are called the inter-channel phase/time/level difference (IPD, ITD, ILD). The paper also states that only IPD is used as the ITD and ILD are very unreliable when used on reverberant recordings. The equation to calculate IPD is this

$$cosIPD(t,f,p,q) = cos(\theta_{t,f,p,q}) \tag{3.7}$$

$$sinIPD(t,f,p,q) = sin(\theta_{t,f,p,q}) \tag{3.8}$$

$$\theta_{t,f,p,q} = \angle x_{t,f,p} - \angle x_{t,f,q} \tag{3.9}$$

where $\angle x_{t,f,p}$ and $\angle x_{t,f,p}$ are the phase of the time, t, and frequency, f, bin in two channels, p and q.

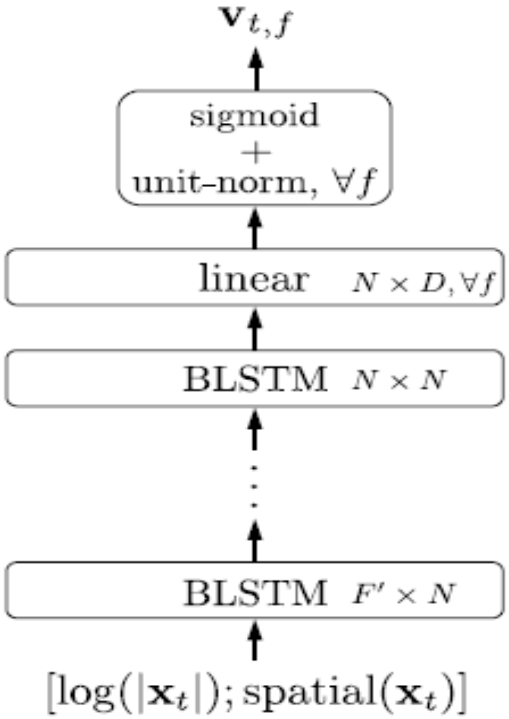

Figure 3.3: Multi-channel deep clustering model, taken from [66]

As can be seen in figure 3.3 is that the IPD, denoted as spatial(x_t), is added on as an extra channel in the input. This IPD can be cosIPD, sinIPD or both, and all three of these possibilities are tested in the paper. The paper uses the WSJ0-2mix speech dataset. For training the sounds from the database were downsampled to 8KHz and mixed together at SNR's of -5dB to 5dB. These mixtures were also passed through a room impulse response generator, which simulated reverberance. The STFT used to generate the spectrograms had a window size of 32 ms and a hop length of 8 ms. This was then passed through a mel filterbank with 129 filters, and the log magnitude of the spectrogram was taken.
The model itself used 4 BLSTM layers with 600 units and 1 dense layer with an unknown amount of units, presumably 10 as that was the best performing size from previous papers. The segments of spectrogram used for the training of the model were 400 frames long, as opposed to the 100 used in [24]. The paper achieved 12.9dB SDR by adding both cosIPD and sinIPD as inputs for sounds made in an anechoic environment with 2 channels. For reverberant recordings the paper achieved 8.9dB, 9.3dB, 9.4dB SDR with 2, 3 and 4 channels respectively, also by adding cosIPD and sinIPD as inputs.

3.5 Deep Clustering And Conventional Networks For Music Separation: Stronger Together

This paper [43] builds on [30] by proposing a network structure that has two "heads", one being the original clustering step, while the other is a fully connected layer with a softmax function. This structure can be seen in figure 3.4. By training the body of the network with a combination of the loss from those two heads the network achieves performance greater than either method on their own. As in [25] they define their deep clustering loss function as in equation 3.3, which they call L_{DC}. However for the softmax head they define the loss as the magnitude spectrum approximation (MSA)[28, 13]

$$L_{MSA} = \sum_c ||R^{(c)} - M^{(c)} \odot S||_2^2 \tag{3.10}$$

where $M^{(c)}$ is the estimated mask for source C, S is the original signal and $R^{(c)}$ is the true signal for source C. $||A||_2^2$ is the squared L_2 euclidean norm of the matrix. However, the paper states that this is not a reliable loss function as the magnitude of S might be smaller than the magnitude of $R^{(c)}$ as there might be destructive interference in S from the other sources. This means that the loss function can never truly 0 as $M^{(c)}$ is only between 0 and 1, and therefore cannot make up for the lower magnitude of S in the subtraction. As a result the paper proposes another loss function

$$L_{mMSA} = \sum_c ||(O^{(c)} - M^{(c)}) \odot S||_2^2 \tag{3.11}$$

Where $O^{(c)}$ is a reference mask such that $O^{(c)} \odot S \approx R^{(c)}$. In this way M can converge to O smoothly with no fear of getting stuck at some point. However this also creates a problem for the accuracy of the algorithm. The generated single source audio will always contain destructive interference from the other sources, because of the network being trained with a loss that converges to an approximation of the true source signal, instead of the signal itself.
They define the total loss of the network as

$$L_{CHI} = \alpha * \frac{L_{DC}}{TF} + (1 - \alpha) * L_{MI} \tag{3.12}$$

where L_{MI} is the loss of the softmax head and α is some scalar between 0 and 1. This allows the network to be trained either fully on clustering, $\alpha = 1$, or on softmax, $\alpha = 0$, or somewhere in between. The paper is evaluated on the DSD100 dataset and formerly public iKala dataset. Before evaluation the songs are downsampled to 16KHz and then analyzed by an STFT with a 512 sample window size and 128 sample hop length. This spectrogram is then filtered with a mel filterbank with 150 filters. The network used 4 BLSTM with 500 units and a dense layer with 20 units, each head used dense layers with D units, where D is the embedding dimension, and the MI head used another dense layer

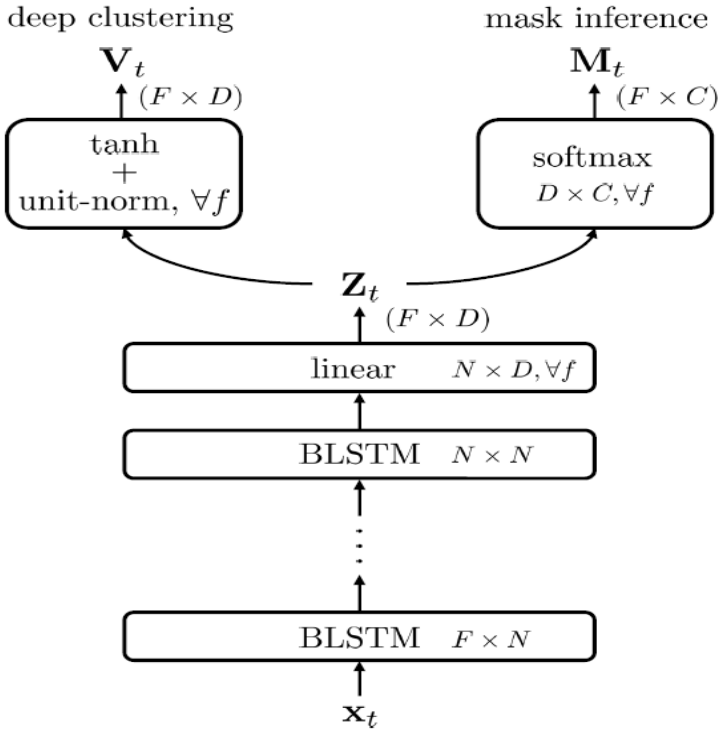

Figure 3.4: The chimera model, taken from [43]

with C units, where C is the number of sources. For training the rmsprop[23] algorithm was used. As a performance measurement they used SDR, which they defined as scale invariant SNR, seen in equation 2.15. The paper obtains a 10.8dB instrument separation and a 6.6dB vocal separation on the iKala dataset, and a 7.9dB instrument and 5.5 vocal separation on the DSD100 dataset.

3.6 Alternative Objective Functions for Deep Clustering

This paper [67] builds on [43] by suggesting new loss functions for the model, while retaining the structure of the model. The paper suggests that a graph Laplacian distance can be used as a loss function for the deep clustering head, which takes this form

$$L(Y, V) = ||V^T * D_V^{-1} * V||_F^2 + C - 2 * ||V^T * D_V^{-\frac{1}{2}} * D_Y^{-\frac{1}{2}} * Y||_F^2 \tag{3.13}$$

Where V is $FT \times K$ embedding matrix, where FT is the number of time-frequency bins in the spectrogram and K is the embedding dimension, and Y is the $FT \times C$ ideal binary mask matrix, where C is the number of sources.
And D_V = diag($V * V^T * \mathbf{1}$) and D_Y = diag($Y * Y^T * \mathbf{1}$). The paper states that this function will help normalize the clusters that k-mean will form as it compares the graph Laplacian

of the embeddings to that of the true clusters.

The paper also suggests that deep linear discriminant analysis (deep-LDA) could be used to condition the embeddings. This could be done using this equation

$$L(Y,V) = \frac{\sum_{c=1}^{C}\sum_{i,y_i,c=1}(v_i - \hat{v}^{(c)})^2}{\sum_i (v_i - \hat{v})^2} \tag{3.14}$$

where v_i is an element in V, $\hat{v}$ is the mean of all the embeddings and $\hat{v}^{(c)}$ is the mean of the embeddings belonging to source c. The paper states that this loss focuses on the ratio of within-class variance, which is in the nominator of the equation, to global variance, which is the denominator. The objective here is to have low within-class variance while having large global variance to ensure that embeddings from different classes are not close.

For the mask inference head of the network, the paper suggests a new loss function that does not use an approximate mask like in equation 3.11, but rather accounts for the difference in phase between the source and the mixture. This is done using phase sensitive spectrum approximation[36] and the paper truncates this to between 0 and $|X|$, where X is the mixture signal. They call this tPSA and the equation is this

$$L_{MI,tPSA} = \min_{\pi \in P} \sum_c ||\hat{M}_c \circ |X| - T_0^{|X|}(|s_{\pi_{(c)}}| \circ cos(\theta_X - \theta_{\pi_{(c)}})||^2 \tag{3.15}$$

Where, as said, X is the mixture signal, P is the possible permutations of the sources and predicted signals, π is a single permutation, $\hat{M}_c$ is the generated binary mask, $T_0^{|X|}(X)$ is the truncation operation, $s_{\pi_{(c)}}$ is clean source c in permutation π and θ_X is the phase of signal X.

Models with the new loss functions were trained on the WSJ0-2mix dataset, which underwent the same pre-processing steps as [43]. They achieved a SDR of 11.5 dB using the tPSA method. For this result they also experimented with the griffin-lim algorithm[19] and the MISI algorithm [20], both for phase reconstruction. These algorithms are much better for reconstructing the source signal, than simply using the original phase of the bin in the mixture spectrogram, as [24] and [43] do.

3.7 End-to-End Speech Separation with Unfolded Iterative Phase Reconstruction

Based on the findings of [67] this paper [69] sought to find a better way of reconstructing the original phase of the source signal. To do this it proposes a model based on the MISI algorithm [20], where instead of using the iSTFT operations as an algorithm, they are used as a deterministic layer in the model which can be backpropagated through. The model is seen in figure 3.5, where the chimera model can be seen at the bottom, and the iterative

phase reconstruction is seen above.

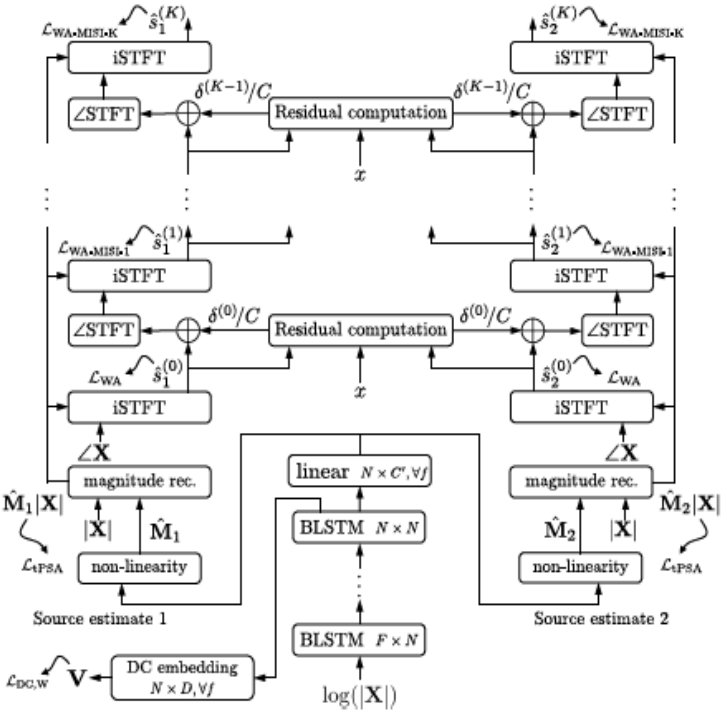

Figure 3.5: The iterative phase reconstruction chimera model, taken from [20]

The MISI algorithm works by continuously computing the iSTFT of the sources and compare their sum to the mixture signal, the resulting signal of this operation is then divided by the number of sources to create an average. After this is done it is added to each other the estimated signals and the phase of that signal is calculated. The iSTFT of this new signal with the new phase is then calculated and the process is repeated. This is called an iteration, and the more iterations the better. In the model an iteration corresponds to one of the layers seen in the top of figure 3.5, the more iterations, the more layers. This model was trained on the WSJO-2mix dataset, with the same preprocessing steps as [43]. The model used 4 BLSTMs with 600 units in each direction, on 400 frame segments of the log mel magnitude spectrograms. As an optimizer this paper used ADAM[35]. The best results came by using 5 of the phase reconstruction layers, and resulted in a 13.2 dB SDR and 12.8 dB SI-SNR.

3.8 TaSNet: Time-Domain Audio Separation Network for Real-Time, Single-Channel Speech Separation

TaSNet [42] is a model developed to handle time-domain audio separation, as opposed to frequency domain separation done in deep clustering and most other recent state-of-the-art methods. This was done because time-domain separation affords near real-time separation, if the sample duration needed for the model is low enough. This paper states that this method can achieve state-of-the-art separation using only 5ms samples, which is low enough latency to be considered real-time.

The model does this by estimating the mixture matrix of the sources in the sample, just like ICA and other matrix factorization methods. It's advantage is that only a single channel is required for this method, while other methods require equal or more channels than sources to work. The model is based on the idea that any signal can be split into basic signal components, and that these components can be added together to create any part of the signal, such as an individual source. The challenge is the find the basis signals and the mixture matrix for them.
The model has four steps; preprocessing,encoding, separation and decoding as can be seen in figure 3.6.
In the preprocessing step, the signal is divided into K non-overlapping L length segments which are then each fed to the network.
The encoding stage contains a 1D gated convolutional layer which finds an embedding for the signal, which is then passed on to the separation stage. This part of the model contains LSTM layers, and a dense layer, whose task it is to find the mixture weights for the basis signals that will be generated in the decoding module. The LSTM layers have skip connections every two layers, such that information is retained more easily between them. It is unclear what the structure of decoding module actually is, the paper simply states that 1D convolutions are used to create the basis signals. It is therefore assumed that by this the paper means that 1D convolutional layers get the signal input from the start of the network, and generate new basis signals based on this. These basis signals are then multiplied by the weights found by the separation stage, and summed together to create the two source signals.

This method uses SI-SNR as a loss function, seen in equation 2.15 and uses permutation invariant training[72] to handle the permutation problem. The model uses 4 LSTM layers with 1000 neurons, and 1000 neurons for the dense layer as well. The paper also suggests that BLSTMs with 500 neurons can be used, however that removes the real-time capabilities of the model. The model was trained on the WSJ0-2mix dataset, which was downsampled to 8KHz as a preprocessing step, and the sounds were mixed at 0dB to 5dB SNR. An ADAM[35] optimizer with a learning rate of $3e^{-4}$ was used for training. The LSTM model reached a SDR of 8.0dB which is better than the original deep clustering paper [24], and the BLSTM model reached 11.1dB SDR, which is better than the Chimera model[43].

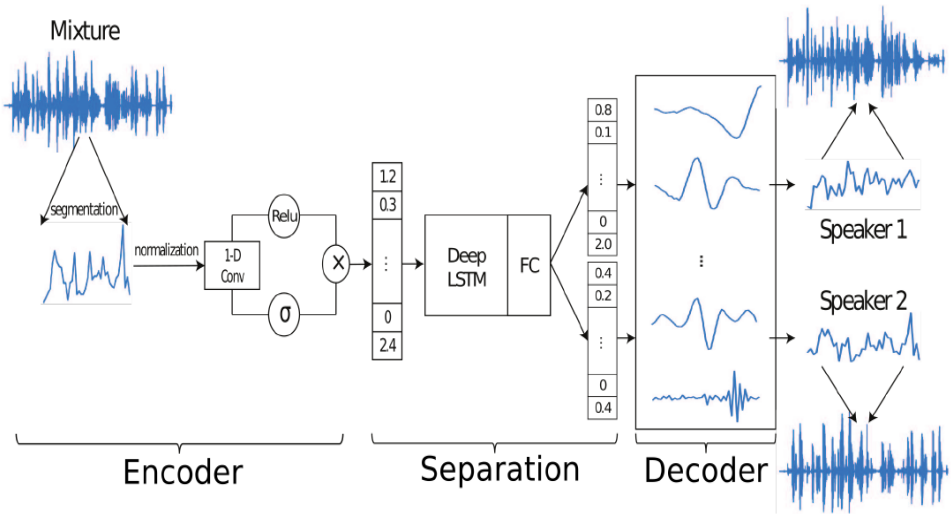

Figure 3.6: The TasNet model, taken from [42]

3.9 FurcaNet: An end-to-end deep gated convolutional, long short-term memory, deep neural networks for single channel speech separation

This paper [57] seeks to solve the single channel blind source separation problem in the time-domain like TasNet[42]. It does this by using a network consisting of 5 1D gated convolutional layers, the internal structure of which is shown in figure 3.2, followed by two LSTM layers and two dense layers. Between all the gated convolutional layers are batch-normalization sections that prevent overfitting. For training the paper uses what it calls the "utterance-level SDR" (uSDR). The equations for this are

$$\hat{x} = \frac{< x, s >}{< x, x >} x \tag{3.16}$$

$$e = \hat{x} - s \tag{3.17}$$

$$uSDR = 10log_{10}(\frac{< \hat{x}, \hat{x} >}{< e, e >}) \tag{3.18}$$

where s is the predicted signal and x is the clean source. Equation 3.18 is actually the same as equation 2.15, as $< x, x >= ||x|| * ||x|| * cos(0) = ||x||^2$. And since equation 3.16 and equation 2.16 are the same too, it can be concluded that uSDR is simply SI-SNR.

The model was trained on the WSJ0-2mix dataset, downsampled to 8KHz, with an ADAM[35] optimizer. The 1D gated convolution layers had kernel sizes of 1000, while the BLSTM layers contained 1000 neurons in each direction, and the dense layers contained 2000 neurons each. The model reached an SDR of 13.3dB.

3.10 Singing voice separation with deep U-Net convolutional networks

This paper [31], seeks to find the ideal binary mask of a spectrogram. It does this using a deep convolutional network that has nearly the same hourglass structure as an autoencoder, however this network uses skip connections between the downsampling layers and the upsampling layers. It does this to retain information throughout the network, and thus make it easier for the upsampling layers to reconstruct the information and find the ideal binary mask. These skip connections go all the way through the network and "binds" it together into a "U" shape. There is also a skip connection from the input to the last layer of the model, which allows the network to focus on finding the best embedding space to store information about the different sources in, without having to focus on storing information for reconstructing the whole spectrogram. All the convolutional layers used a 5×5 kernel, and the downsampling layers used a stride of 2 to halve the image in each direction each layer, but each layer also outputted double the amount of input channels. Except for the input layer, which outputted 16 channels.

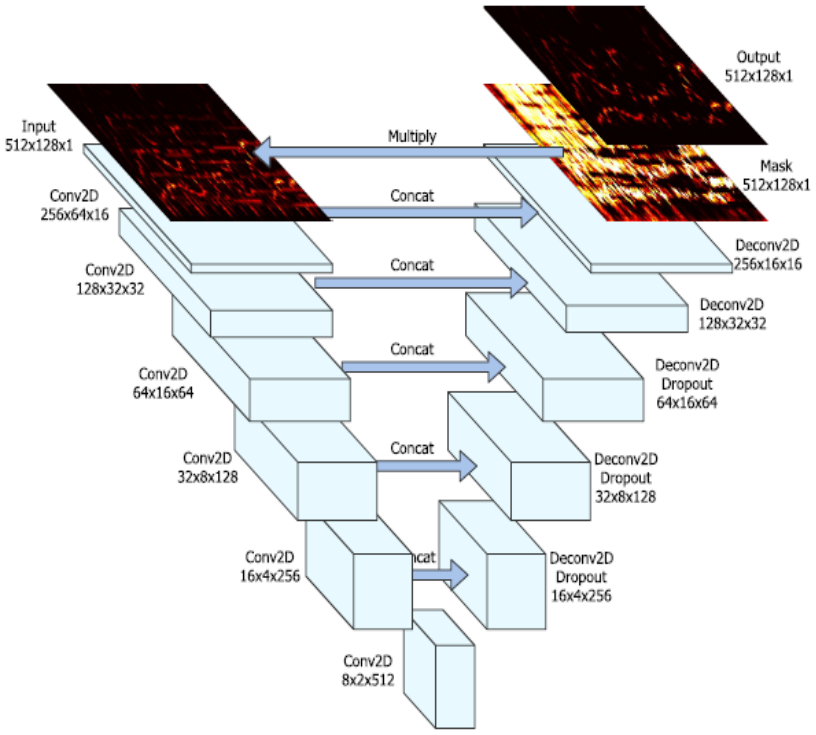

Figure 3.7: The U-net model, taken from [31]

The model was trained on a dataset that is not publicly available, generated in the paper [29]. This dataset consisted of songs from Spotify, and their instrumental versions,

which made it possible to separate the vocals from the instruments, and made it the biggest dataset of its kind. The model was evaluated on the MedleyDB and iKala datasets. It used the normalized SDR (NSDR) as a performance measure together with SIR and SAR. NSDR is calculated as

$$NSDR(S_e, S_c, S_m) = SDR(S_e, S_c) - SDR(S_m, S_c) \tag{3.19}$$

Where S_e is the estimated signal, S_c is the clean source signal c and S_m is the mixture signal. This is in some way another way to phrase the original definition of noise and SNR in equations 2.8 and 2.11, where interference from other signals is not included in the noise. Here the interference from other sources is counted in the second term and subtracted from the first term to leave only distortion caused by noise and artifacts.

All the tracks from the datasets were downsamples to 8KHz before being analyzed by a STFT with window size 1024 samples and 768 sample hop length, giving 512 bins per time-frame. 128 time-frames were given to the network at a time. The network reached a mean NSDR of 11.094dB for vocals and 14.435dB for instruments, on the MedleyDB dataset, compared to 8.749dB vocals and 11.626dB instruments of the Chimera model.

3.11 Wave-U-Net: A Multi-Scale Neural Network for End-to-End Audio Source Separation

This paper [58] is an extension of the U-net structure proposed in [31]. The paper proposes new configurations of the model and how to apply it to a waveform instead of a spectrogram, as the original paper did. This was motivated by other papers such as TasNet [42, 44], which utilizes the waveforms perfect phase information to achieve state-of-the-art results in BSS separation. However, this paper does not generate intermediate waveforms and weights like TasNet, but rather keeps all information internal in the models 1D convolutional layers. This is also the biggest change to the original U-net model, the use of 1D convolutional layers, as opposed to the original 2D layers. Only 1D convolutional layers are required since the waveform is only 1D, and not like the 2D spectrogram used as input for the U-net.

Like U-net, the Wave-U-Net continuously halves the input size, but adds more channels. This process is lossy, as the the amount of channels are not always doubled between layers, meaning that the final latent space is much smaller than the initial input. For an input of size $\mathbb{R}^{16384\times1}$ the output of the final downsampling layer will be $\mathbb{R}^{4\times288}$, for a Wave-U-Net with 12 downsampling layers, and an additional 24 channels per layer of downsampling. This is a latent size of 1152 points, in comparison to the 16384 input points. The benefit of this reduction is that it reduces memory size of the model, while it is assumed that the latent space will still contain enough information to reconstruct the sources.

The paper also suggests ways to improve the upsampling steps, as it is suspected

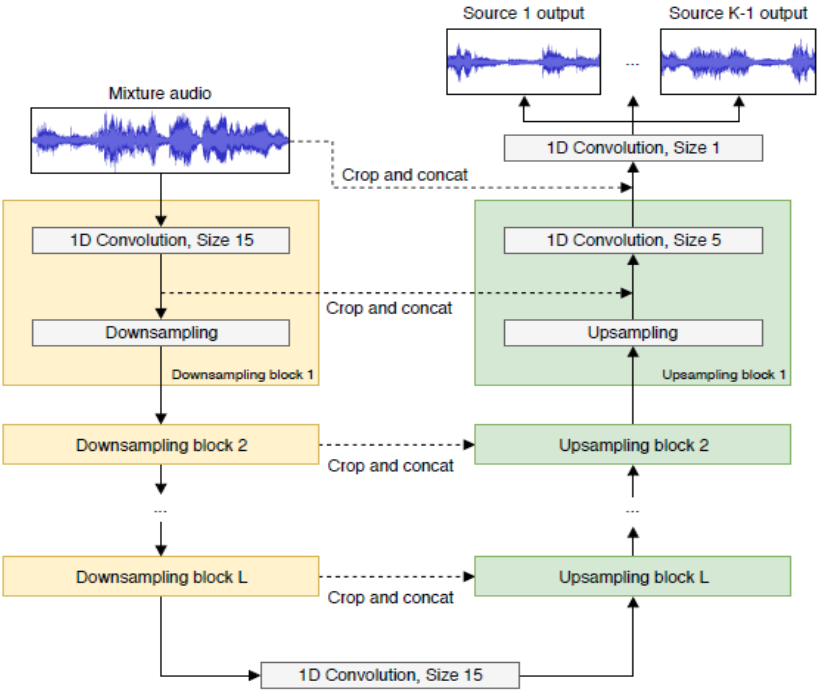

Figure 3.8: The Wave-U-Net model, taken from [58]

the traditional transposed convolutional technique used in other papers produces high-pitched noise in the final output. Instead of upsampling the paper suggests that linear interpolation is used. The paper also states that this interpolation might not be an ideal solution as it is unknown if the latent space between layers is linear or convex. Therefore the paper also proposes that learned interpolations could be used instead, to ensure that the latent space geometry is considered when interpolating. The proposed learned interpolation equation takes this form

$$f_{t+0.5} = \sigma(w) \odot f_t + (1 - \sigma(w)) \odot f_{t+1} \tag{3.20}$$

Where $\sigma(w)$ is a sigmoid activation function with w parameters. This makes non-linear interpolation possible and ensures that the upsampling will always generate latent points that make sense.

Multiple different sizes of input to the network were tested, from 16384 samples to 233459 samples. It was also tested whether or not having more input samples than expected output samples would be beneficial. This was done because having more context might enable the network to make a more accurate prediction about the samples in the middle of the signal.

The network was trained on the MUSDB dataset and evaluated on the MUSDB and CCMixter datasets, all of which were downsampled to 22050 Hz as a preprocessing step. The network was trained with one less output than there were sources, as the last source could be seen as the mixture signal minus the sum of the other estimated signals.
The network was trained using a mean-squared-error (MSE) loss function, and an ADAM[35] optimizer with learning rate $1e^{-4}$ and betas of $\beta_1 = 0.5$ and $\beta_2 = 0.999$. The paper states that is defines one epoch as 2000 "iterations", though it is unclear what is meant by this. Most likely it is 2000 samples, which is actually not a lot when considering some of the sample lengths used, since 2000 * 16384 samples at 22050 Hz sampling rate corresponds to roughly 25 minutes of audio. Compared to the 10 hours and 30 hours of audio per epoch used by other papers [24].

The paper found that using more samples as input than what is required for the output increased the performance of the model. The best model got a mean vocal separation SDR of 0.65dB and a mean instrumental SDR of 11.85dB.

3.12 Final Problem Specification

Based on these papers it was seen that FurcaNet reached very good results by combining convolutional layers and BLSTM layers. It was also seen that Wave-U-Net and Chimera had similar ideas of giving the input of the network to the output, and have it modulated by the output in some way. The Wave-U-Net had a connection to the output layer from the input, such that it was processed in the last convolutional layer of the network, while Chimera predicted a mask which was used on the input mel spectrogram. This ensures that the network finds ways to modulate the input instead of training to reconstruct it, which makes training the model easier.
This trend of combining LSTM/BLSTM layers and convolutional layers was also seen in TasNet, where it also performed very well.
All the above lead to the conclusion that a new network should be constructed that works with these concepts, by combining the spatial resolution of convolutional nets with the temporal resolution of BLSTM layers, and having these layers interconnected, which can greatly benefit the separation of sources. If this network is similar to another state of the art structure then it should also make use of one of the loss functions that has been successfully used by that paper. This is because having different loss functions were shown to have a great effect on the SDR the model could achieve, and if the model structures are similar, using a loss that is good for one will be good for the other.
It was also decided that the most interesting problem to work on would be signing voice separation, and musical separation in general, as it is a field that could also be used to separate a human voice from any given background signal, thus it can have many different applications.

Chapter 4

Implementation

In this chapter the proposed model, HydraNet, will be discussed and implemented. This model will be the combination of two other models, Chimera [43] and Wave-U-Net [58]. The HydraNet is based on these models as they are both state-of-the-art models that solve similar problems with good results. They also both contain elements described in section 3.12. Wave-U-Net uses convolutional layers to seperate a signal, while Chimera uses LSTM layers to generate embeddings for a spectrogram which separates it into spectrograms for each source. Wave-U-Net is also structured like an auto-encoder, in that there is a defined place in the model for the latent space, where LSTM layers can be added to add to that latent space and make it a better representation of the audio. By doing this it is expected that the separation of the model will surpass that of both Chimera and Wave-U-Net.

For the purpose of implementing HydraNet, both the Chimera model and Wave-U-Net will be implemented first, such that it is certain that those models are accurate before combining them to HydraNet.

4.1 Environment, Libraries And System

For this project it was decided to use the programming language Python, version 3.5, for the implementation of the models. This language was chosen due to its large user-base and its big selection of libraries. The libraries used for this project were Numpy, Librosa, Scikit-learn, Tensorflow, Keras and PyTorch 1.0.1, together with visualisation libraries such as Matplotlib and Tensorboard. Only Numpy, Librosa and PyTorch were used for the final implementation, however Tensorflow, Scikit-learn and keras were used for experimentation during the project development phase.

All computation will be done on this system. CPU work will be done on a quadcore Intel i5-4690K 3.60 GHz processor[1] with 16 GB of RAM. All GPU work is done on a

[1] ark.intel.com/content/www/us/en/ark/products/80811/intel-core-i5-4690k-processor-6m-cache-up-to-3-90-ghz.html

NVIDIA GTX 970[2] with 4 GB of GDDR5 VRAM and 1664 CUDA cores at 1.05 GHz.

4.1.1 Tensorflow and Keras

Tensorflow is a machine learning library made by Google, which has many optimizations of tensor operations to make training models faster. It also has integration with NVIDIA's CUDA toolbox/drivers for GPU computation, meaning it can take advantage of the heavily paralellized processing capabilities of the GPU to optimize tensor operations even more. Tensorflow allows for fast computation of tensor operations such as dot products and matrix multiplications, which are heavily used in most neural networks and other machine learning techniques.

Tensorflow has a deferred execution style that is very different to Python's normal eager execution. This means that in Tensorflow a model is first defined by defining each layer as an object with certain hyper-parameters, such as number of nodes in the layer and the shape of the input. Then the connections between these layers is defined, together with other operations, such as activation functions, that should be done on the output of the layers. After this is done the computation graph, i.e. the graph of dataflow through the model, has been defined and data can be given to the model. This is deffered execution as the same functions are called multiple times in different contexts and give different outputs as a result. During the second stage, the compilation of the computation graph, no data is actually flowing in the model and the only information each function is getting is the meta-data about the layer or function it is supposed to operate on. This means that a sigmoid activation function will first be given the meta-data about the layer it is supposed to operate on, and then it will be given the output of that layer once data has been given to the model.

Keras is a higher level library that can operate with multiple different backend libraries such as Tensorflow or Theano. It allows for single line definitions of layers with activation functions and is therefore very useful for increasing the understandability of the model. It also has an optimized workflow for creating sequential models, as it can then take care of the definition of the dataflow through the model, by itself. This makes it a very popular library to use in conjunction with Tensorflow, or other supported libraries, and ensures that most sequential models will be immediately understandable for other users.

4.1.2 PyTorch

PyTorch is similar to Tensorflow, however it is different in some key aspects. PyTorch uses same deferred execution style as Tensorflow, but it also uses a dynamic model definition style. This means that an initial model is first defined, it is then executed on some data and

[2] www.geforce.com/hardware/desktop-gpus/geforce-gtx-970/specifications

provides some resulting prediction that can be used for loss calculation and backpropagation. After this is done, the model can be changed and be executed again on other data. Before this execution, PyTorch will recompute the graph structure, with respect to the input shape, and execute this new graph structure. This is very different to Tensorflow's single definition approach, where the model is defined once and if anything about input changes then the graph breaks. The dynamic nature of PyTorch is especially useful when it comes to RNN layers, as it allows the structure to adapt to variable length time series, and with convolutional layers it allows for some differently shaped images to be computed by the same network structure. However, it is also something to be kept in mind when minimizing run-time, as it adds overhead to the model execution by continuous graph re-definition.
Another benefit of PyTorch's dynamic nature is that data is easily accessible, it can be manipulated by other libraries, such as numpy, and it can be reentered into the graph at any point one wants. Any tensor can also be moved freely between the CPU and the GPU, provided that the GPU has CUDA. This can be used to integrate a CPU method into an efficient GPU graph structure.

4.2 Chimera network Implementation

4.2.1 Tensorflow and Keras Implementation

After the analysis it was decided that the Chimera network model[43], should be re-implemented. Other implementations were available online, however a re-implementation would give more knowledge about the model, and enable further development on the model. It was also decided that Tensorflow and Keras should be used for this implementation. This was due to the fact that other Python implementations of Chimera network had been found online, all of which used Tensorflow and Keras.
The main Python Chimera network implementation that was found, was from Github user "arity-r"[3] who was initially believed to be one of the authors of the paper. Because of this, this implementation was used as a reference for the initial re-implementation.

Initially it was believed that the k-means clustering step, described in the Chimera network and Deep Clustering papers[43, 24], was vital to the training of the model, and to the loss function described in [24]. To integrate the k-means into the Tensorflow computation graph was conceptually difficult, and it was therefore decided that this approach should be avoided.

It was found that there were some key differences between the model described in [43] and the implementation by "arity-r". The most major difference were the shape of the input to the model and the structure of an intermediate layer immediately after the bi-directional

[3]https://github.com/arity-r/ChimeraNet

LSTMs. As can be seen in figure 4.1a the input to the model is of size $\mathbb{R}^{FT\times 1}$, where FT is the size of the flattened mel spectrogram with F frequency bins and T time-steps. This means that when this is passed through an LSTM, each FT index will be treated as a time-step and an embedding is created such that the output of the LSTM is $\mathbb{R}^{FT\times L}$, where L is the amount of neurons in the LSTM.

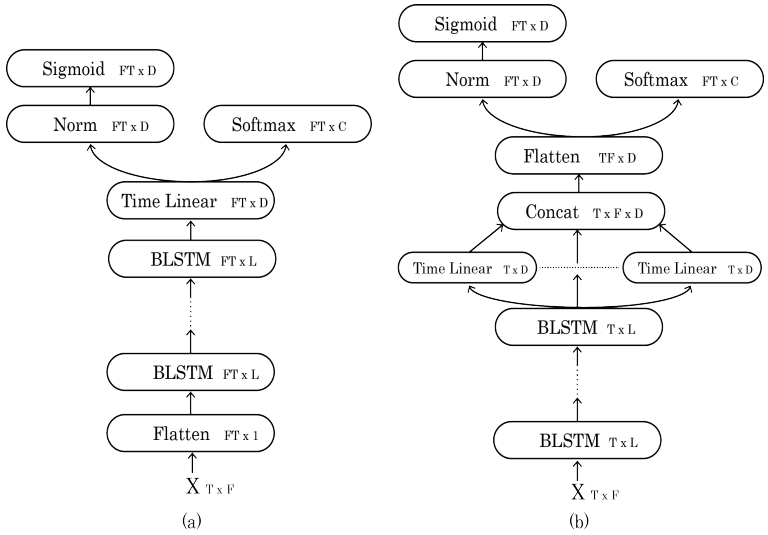

Figure 4.1: (a) is the original chimera model, (b) is the modified chimera model.

In figure 4.1b it can be seen that the input to the model is of size $\mathbb{R}^{T\times F}$ meaning that the output of the LSTM layer will be $\mathbb{R}^{T\times L}$, where F is the number of frequency bins, T is the number of time-steps and L is the number of neurons in the LSTM. This is a significant loss of information, compared to the original structure, which is why the model needs an additional layer to reconstruct this information. This additional layer is the one seen in figure 4.1b right after the bi-directional LSTM (BLSTM) layers . This layer utilizes F different densely connected channels to construct an embedding for each frequency bin in each time-step. Each dense module takes in the L values for each time-step and outputs D values, where D is the embedding dimension, thus generating a $\mathbb{R}^{T\times F\times D}$ space when the outputs are concatenated on the second dimension. Because this is a three dimensional space and the cost functions need a two dimensional input space, this output is flattened in the first two dimensions before being passed on in the model.

These deviations from the model lead to the decision to change from Tensorflow to PyTorch, where no other implementations of Chimera network were available but the integration of k-means clustering into the training graph would be possible.

4.2.2 PyTorch Implementation

As said earlier in the description of PyTorch, the advantage of PyTorch over Tensorflow is its flexibility in graph construction and data handling. It is the ability to convert tensors into numpy arrays and back into tensors, that allows PyTorch to handle integration of k-means clustering into the training graph. This ability was utilized in the initial implementation of the original Chimera network structure. The model is the one seen in figure4.1a, with a flattening layer in the beginning to reshape the $\mathbb{R}^{T\times F}$ mel spectrograms into a flat series of $\mathbb{R}^{TF\times 1}$. This flat spectrogram was then used as input into the BLSTM layers with 600 units in each direction, which was connected to another similar LSTM layer, after which a time-distributed dense layer created embeddings of size D for each of the TF points.

However, it was found that this model was very memory intensive due to how BLSTMs, and LSTMs in general, process data. As discussed in section 2.2.3 a BLSTM outputs a vector of size $\mathbb{R}^{T\times 2L}$ where L is the amount of LSTM units in one direction. This is because each input is processed by two LSTM units, and the outputs are concatenated into one output vector. It also means that when two BLSTMs are connected, and they have the same amount of units, amount of connections between them will be $\mathbb{R}^{T\times 4L}$, since each of the outputs of the first BLSTM will be connected to two units in the next BLSTM. In practise this means that if two connected BLSTMs each have 600 units, the space between them will be of size 2400. If we also consider the length of the time series, 100×150 in this case, we can see that space between the two BLSTMs must reach $\mathbb{R}^{15000\times 2400}$. When the byte size of data types is considered, in this case 4 byte floats are used, then we can calculate that this space will fill 144MB in memory. This is for a single sample, not considering the size of all the weights in each BLSTM, or the size of the spaces in the rest of the network. The size of this space limits the batch size to 3 samples, on the GPU described in section 4.1. It was also observed that for a single batch of 3 samples, the time to execute the model and do backpropagation was on average around 55 seconds. This means that for a dataset of 1600 samples, the average epoch time would be around 24 hours. Of course the average batch run time was only based on less than 10 batches, and is therefore not the true average run time, though it is a good enough estimate.

Loss Functions

This implementation used the same loss functions as described in [43] and [24]. However the modified mask inference loss function loss function described in [43] cause problems as it was not clear how exactly it should be calculated. This is the loss function in equation 3.11. The confusion stemmed from the calculation of O, the modified ideal ratio mask, as there are multiple ways to calculate it. The problem is that when sources have the same magnitude in a frequency bin, but opposite phases, then in the mixture spectrogram this bin will have 0 magnitude as the two sources cancel out, but the two source magnitudes

have to be reconstructed as a ratio of the mixture spectrogram magnitude in that bin. This is a problem as that is not possible. In the less extreme case where the sources don not quite cancel out, but just produce a mixture magnitude that is lower than on one of the sources, this source cannot be reconstructed as a ratio of the mixture magnitude below 1. This is the problem that O seeks to rectify as it is supposed to be a ratio mask of the mixture which approximates the source, but does not reconstruct it fully. However, as stated there are multiple ways of calculating this O, but the paper does not state which they use. In implementation the O was calculated as this

$$R^c = \frac{S^c}{S^m} \tag{4.1}$$

where R^c is the ideal ratio mask for source C, S^c is the spectrogram of source c and S^m is the spectrogram of the mixture

$$O = \frac{R^2}{||R||^2} \tag{4.2}$$

In this way O is the square of the normalized ideal ratio mask. The norm of R is taken along the second dimension, as R is of size $\mathbb{R}^{FT \times C}$, where FT is the number of time-frequency bins and C is the number of sources. This normalization is done because, as discussed before, some of the magnitudes of the sources might be bigger than the magnitude of the mixture, meaning that R^c might contain values that are bigger than 1. The squared normalization ensures that proper fractions are created in every bin.

This might also create some artifacts in the training, because if more sources have bigger magnitudes than the mixture, in the same bin, then those sources will be seen as contributing only half or less to that bin. This is due to the normalization term.

Signal Reconstruction

The signal reconstruction is something that later papers[69, 45, 68] have worked on, by trying to more accurately reconstruct the original phases of each time-frequency bin. However, Chimera network simply uses an inverse mel-filterbank to translate the magnitude mel-spectrogram back to a full magnitude spectrogram. The original phase of each bin, from the original full spectrogram, is then applied to the magnitude spectrogram and the inverse STFT is then taken of this. This is also the approach taken in this implementation. It should be noted that this is a naive reconstruction that is guaranteed to cause errors and artifacts in the reconstructed signal, due to limited phase information about the true sources.

4.3 Wave-U-Net Implementation

Because Wave-U-Net's structure was so simple and easily modifiable it was decided that it should be re-implemented in PyTorch. This was also decided due to the insights that

could be gained into the practical issues of the model, and how it could be optimized or changed.

4.3.1 Model Structure

As stated before, PyTorch uses dynamic models and layers, meaning that convolutional layers do not need to know the size of the sequence that they are processing, only the number of input and output channels. This is also true for recurrent layers, however here channels are exchanged for input feature size and number of recurrent units in the layer. In a sequential model, convolutional layers in the middle of the model do not need a specified number of input channels, only output channels, as the amount of channels from the previous layers is assumed to be the amount of input channels for the current layer. However, in the case of Wave-U-Net this is not true and thus all input and output channels have to be defined. For the downsampling layers this was done using an array, in which a series of values could be defined manually if wanted. However, in this case a desired channel increase per layer was chosen, and a simple algorithm was used to populate the array with values incremented by this amount. This means that if a channel increase of 24 per layer was desired for a 3 layer model, the array would be "1, 24, 48, 72". The 1 is added in the beginning of the array as an initial input channel size, and can be changed if more channels are given as input.

To calculate the number of input and output channels of the convolutional layers in the upsampling part of the network, the array storing the channels for the downsampling network was reversed and doubled. Meaning that for a 3 layer model with a 24 channel increase per layer the upsampling channel array would be "144, 96, 48, *C*", where *C* is the number of output sources. This doubling is done due to the residual connections from the downsampling to the upsampling layers, which concatenate the output of the downsampling layer to the input of the upsampling layer, doubling the amount of channels in the input.

4.3.2 Differences

This implementation does not use the proposed "learned upsampling" method from the Wave-U-Net paper[58], and instead uses a linear interpolation upsampling method. Another key difference is that this implementation has *C* outputs, corresponding to the amount of sources that should be separated, and not *C*-1 as the Wave-U-Net paper. This was done so the difference in performance could be measured between the two approaches. It also makes the model end-to-end, as all the sources are be generated directly by the model. This is not true for the original Wave-U-Net as the last source was generated by subtracting the sum of the model outputs from the mixture signal, thus the remaining signal should be the last source.
This implementation also does not use the dialeted convolutional layers that the Wave-

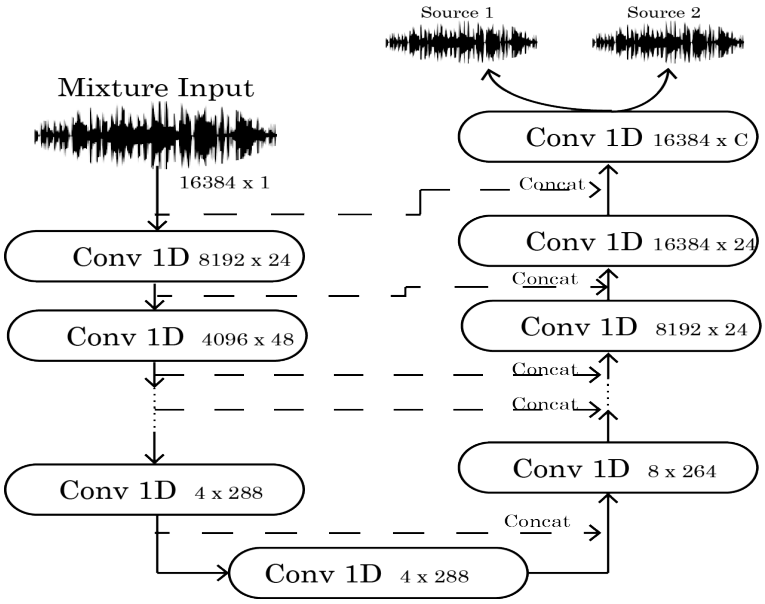

Figure 4.2: The Wave-U-Net Model

U-Net paper proposes for one of its model variations. This is due to the fact that this model will be combined with the Chimera model, and thus the most simple version of the Wave-U-Net model was wanted.

4.4 HydraNet Implementation

HydraNet is the main contribution this paper makes to the field of source separation. The model was inspired by the Chimera networks use of BLSTM layers to expand a latent space, to make it coherent over the time series, and the Wave-U-Net's use of the time domain representation of the signal as input. Thus the HydraNet was a combination of the two networks.

4.4.1 Concept

It was observed, that the latent space generated at the "bottom", or middle, of the Wave-U-Net was very similar in shape to the mel magnitude spectrogram used as input for the Chimera Network. This is also true for the latent spaces in higher layers of the Wave-U-Net model. Because of this similarity it was decided that the body of the Chimera network should be inserted before the middle layer of the Wave-U-Net, taking input from the last downsampling layer and outputting into the middle layer of the model. The idea was that the Chimera network body would generate a latent space that was more coherent in time than the space generated by the Wave-U-Net without dialated convolutional layers. It was

also thought that BLSTM, and LSTM layers in general, would capture long term dependencies better than dialated convolutional layers, as the BLSTM layers process the entire time series in one layer, while the dialated convolutions need many layers to process the entire series. In this way BLSTMs will capture the time dependencies with a single pass of the signal, while the dialated convolutions will need many layers to process the signal before time dependencies can be found.

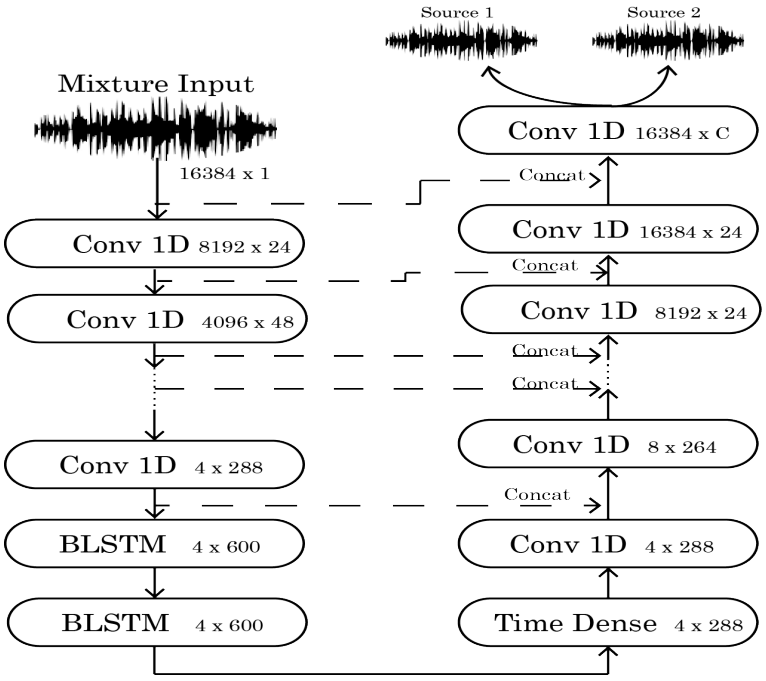

Figure 4.3: The full HydraNet model

In this HydraNet model the normal convolutional layers will find the short term dependencies of the signal while the BLSTM will capture the longer dependencies, that can then be processed by the convolutional layers in the upsampling. This is also why BLSTMs are used as opposed to LSTMs, as there might be critical information in the last part of the signal which will help separation in the first part, and vice versa.

4.4.2 Implementation

The code and logic for the Wave-U-Net model was reused for this model, together with the code for the first three layers of the Chimera network seen in figure 4.1b. This Chimera model was used as it did not require flattening of the input before processing, making it work well with the 2 dimensional input of the model. It was decided that two BLSTM layers with 600 units would be used, however it was also decided that the following array of dense layers was not needed, and that a single dense could be used instead. This was

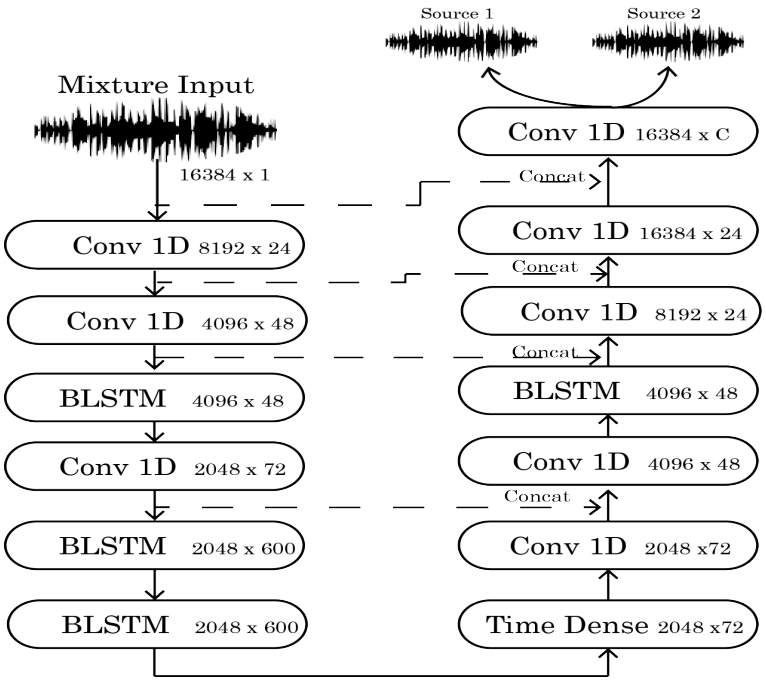

Figure 4.4: The short HydraNet model

because only a 2 dimensional output was desired, not a 3 dimensional output as the array of dense layers would produce. The array of dense layers was designed such that from an input of size $\mathbb{R}^{T\times N}$, T is the amount of time-steps and N is the amount of BLSTM units, an output of size $\mathbb{R}^{T\times F\times D}$ could be constructed, where T is the same as before, F is the number of channels and D is the desired embedding dimension. However in this case only a output of $\mathbb{R}^{T\times F}$ was desired, thus a timedistributed dense layer with F units could be used.

As with the Wave-U-Net model, this model was made to be easily re-sizable, with a parameter for the amount of downsampling and upsampling layers, and a generated array keeping track of the input and output channels for those layers. The BLSTM layers, and timedistributed dense layer, also refer to this array for the size of their input and output channels, thus keeping the model re-sizable. This also gives the opportunity to test different depths of the model.

A variant of the HydraNet was also created, which placed BLSTM layers before the first downsampling layer, after the last upsampling layer, and after every two downsampling or upsampling layers. This is called the "time sensitive HydraNet" as its goal is to more easily capture long term dependencies. This architecture was chosen to allow for a shallower network with less convolutional layers, and will therefore not be tested with more than 4 convolutional layers, which is the structure seen in figure 4.4

Chapter 5

Evaluation And Results

All the implemented models were evaluated using different optimizers, loss functions and hyper-parameters, which were either based on the original papers of the models or found by experiment. The dataset used for all three models was DSD100 [41], however Chimera and the two other models have different preprocessing procedures that will be described in their sections. The mir_eval [51] python library from MUSDB18 [59] was used for all evaluation of the models.

5.1 Chimera Evaluation

Several tests were performed on the Chimera model. Two different loss functions were tested for the binary mask head, mean squared error and the sparse matrix multiplication loss described in [24]. For the mask inference head only a mean squared error was tested. The model was also trained with three optimizers, stochastic gradient descent (SGD), rmsprop and ADAM. This was done as SGD and rmsprop were used by [24] for the deep clustering model, and rmsprop was used by [43] for Chimera. ADAM was chosen as an experiment to evaluate if a decay would increase the SDR or make training faster.

5.1.1 Data Preprocessing

All songs were loaded using the librosa [46] library, and the two channels of each "wav" file were averaged to a single channel. The signals were also downsampled from a native 44.1KHz sampling rate to a 8KHz sampling rate. The source sound files for the instruments were added together to create one instrumental track per song, and one vocal track. A short time fourier transform (STFT) was then applied to the signals, with 1024 frequency bins, and a mel-filter bank with 150 filters was applied to each spectrum to reduce the number of bins to 150. A window size of 8 milliseconds was used for the STFT, with a hop length of 4 milliseconds. The magnitude of the complex value in each bin in the resulting mel spectrogram was then calculated to create a real-valued magnitude mel spectrogram.

This spectrogram was then split into non-overlapping windows of 100 time-frames, like [24] describes, and the resulting array of windows was saved as a numpy file for easy access during training.

5.1.2 Training Procedure

abandoned. The second approach of using a non-flattened spectrogram was pursued instead. This model was trained with SGD and rmsprop to verify the results with both [24] and [43], and with ADAM for experimentation. The models were trained until it was observed that training loss for one head had plateaued for 10 epochs. For SGD a learning rate of 10^{-4} was used with a momentum of 0.9, based on the hyper-parameters used in [24]. This was then trained for 70 epochs before training loss plateaued. For rmsprop a learning rate of 10^{-4} was used with no other hyper-parameters. This was trained for 30 epochs. For ADAM a decay of 10^{-6} was used, with a delta1 of 0.5 and delta2 of 0.999. This was trained for 70 epochs before training loss plateaued. The loss functions described in section 4.2.2 were used for all training.

5.1.3 Results

All three models were used to predict the DSD100 test data, which was preprocessed in the same manner as the training data- The resulting mel magnitude spectrograms, with 100 time-steps each, were recombined to their original length and their signals reconstructed as described in section 4.2.2. This signal reconstruction step was also performed on the clean sources, such that any detected SDR would be not be affected by the reconstruction process. Results are reported for both original and reconstructed clean sources. The results

	Cl. Instruments		Cl. Vocal		Gen. Instruments		Gen. Vocals	
	mean	SD	mean	SD	mean	SD	mean	SD
SGD	-21.58	5.67	-34.69	1.89	-25.36	6.08	**-29.80**	8.05
rmsprop	**-20.75**	5.52	**-34.68**	1.90	-25.42	6.58	-29.81	8.06
ADAM	-20.77	5.53	-34.68	1.90	-25.41	6.59	-29.81	8.07

Table 5.1: Chimera mean SDR (dB) and standard deviation (SD), Cl. is the results comparing to clean tracks and Gen. is the results comparing to clean tracks that have been transformed to a mel spectrogram and generated again in the same manner as the output of the model

of this network are not good, as higher SDR means a better separation, and these methods produce results far below the 8.1dB instruments SDR and 6.1dB vocals SDR of the Chimera paper[43]. It should also be noted that the "Gen." sources should be closer to the estimated sources of the network, due to being processed by the same signal reconstruction method, however the instrument tracks seem to produce worse results for the the Gen. sources than the "Cl." sources. Though vocal seperation results are consistently higher when compared to the "Gen." tracks rather than the "Cl." tracks. It can only be assumed that SDR calcu-

lations cannot handle the added noise of the reconstruction method, in supposedly clean sources.

5.2 Wave-U-Net Evaluation

The several versions of the Wave-U-Net model were trained using two different loss functions. The optimizer used was ADAM. The two loss functions were mean squared error (MSE) and scale-invariant signal-to-noise-ratio (SI-SNR) which was used by TasNet [42] and which is shown in section 3.8 equation 2.15. SI-SNR was not originally used as a loss function in Wave-U-Net [58], however since the output of Wave-U-Net and TasNet is the same, this loss function could be used to improve the performance of Wave-U-Net without changing the model structure. However, using SI-SNR as a loss function introduced some restrictions on the training data. These restrictions were that no section of the audio could be silent, else a negative infinity would be generated by the function, due to it being a logarithmic function. This problem was handled during the preprocessing of the data. All models described in this section used 12 downsampling and upsampling layers, and a channel increment of 24.

5.2.1 Data Preprocessing And Generation

The data used for these models was also DSD100, as Chimera used. However, all songs were resampled to 22KHz, as opposed to the 8KHz used by Chimera. For the MSE training data the songs were then split up into non-overlapping sections of size 16384, as that was one of the sample sizes that worked well in the Wave-U-Net paper [58]. The paper also described that using big sample sizes, over 100.000, improved performance, however as seen with the Chimera network these sample sizes are very memory intensive and can slow down training a lot. As with the Chimera network, the instrumental tracks were added together to create a single instrumental track for each song. The vocal and instrumental tracks were then added at their native SNR, to create a mixture track. The arrays of the slices of the instrumental, vocal and mixture tracks were then saved in numpy files to make loading faster during training. This data will be called "sequential" data.

For the SI-SNR training data no silent sections could be present in either vocals or instruments, as is also described in the TasNet paper[42], thus all silent sections had to be removed. As the TasNet paper also notes this is mostly a problem in the vocal tracks, as there are often long breaks of silence between verses. Removing silent sections in the vocal tracks was done with a simple onset detection algorithm. This algorithm calculated the envelope of the signal, and found places where the value of that envelope was above or below a threshold. The signals were normalized before this calculation to ensure that the threshold value would produce good results for all signals. Once the vocals had been sep-

arated they were saved in a numpy array file, and the same was done for the instrument tracks.

During training these files were then loaded and all tracks that had a length below the sample size were discarded. The sample size used for these experiments was 16384. After tracks that were too short had been discarded, a random set of vocal and instrumental tracks were chosen. For both the chosen vocal and instrumental track, a random index value was then chosen between 0 and the length of the track minus the sample size. The tracks were then sliced from this index to the index plus the sample size, to generate a new track with the same length as the sample size, in this case 16384. 10000 pairs of vocal and instrumental tracks were generated this way. It should be noted that the vocal and instrumental tracks were chosen randomly from the list of all the songs, meaning it is unlikely that the chosen vocals and instruments are from the same original song. The implications of this will be discussed more in section 6.4. Three experiments were done on the mixing of the resulting slices of vocals and instrumentals, mixing them at their native dB, mixing them at an SNR of 1 with overall dB between -5dB and 30 dB, and mixing them at a SNR of -5dB to 5dB with the overall dB of each signal ranging from 0dB to 25dB. This was done to see if mixing with a low SNR but random overall dB would improve results, since most other papers [24, 42, 58] take this approach, or if it is inconsequential.These approaches to data generation will be called "native", "random loudness" and "fixed range SNR" respectively.

5.2.2 Training Procedure

As said Wave-U-Net was trained with two different loss functions, MSE and SI-SNR, and four different data generation methods, the sequential method and the three generated methods. All models were trained with the ADAM optimizer and a batch size of 50.

Two models models were trained with the MSE loss function, but with different hyper-parameters. The first model (**M1**) was trained with the same hyper-parameters as the Chimera model, 10^{-4} learning rate, 10^{-6} decay, a β_1 of 0.5 and β_2 of 0.999. The second model (**M2**)) was trained using the recommended hyper-parameters from the Wave-U-Net paper, 10^{-4} learning rate, a β_1 of 0.9 and β_2 of 0.999. Both models were trained with sequential data. Both models were trained for 100 epochs

One model (**M3**) was trained using the SI-SNR loss function, with the same hyper-parameters of 10^{-4} learning rate, a β_1 of 0.9 and β_2 of 0.999. This model used the native generation technique for its training data. This model was trained for 100 epochs.

Four models were trained using a hybrid training regimen with both SI-SNR and MSE loss functions. Fixed range SNR data was used for these tests. The first of these models (**M4**) was trained with a mixed loss function where the two loss functions were added

together before backpropagation. However, it was observed that the MSE loss was increasing with each batch, instead of decreasing as it should, and that the SI-SNR loss was not decreasing much either. Because of this, this approach was abandoned quickly, though it is still a valuable result and informed the next test of the second model (**M5**). Because of M4's poor results with a mixed loss, M5 was trained in an alternating regimen with SI-SNR loss being used in even epochs and MSE being used in odd epochs. This model used a native generation for its data. The third of the models (**M6**) used the same training regimen, but used random loudness generation, while the last model (**M7**) used fixed range SNR generation. These models were traning for 100 epochs.

The idea behind this hybrid loss stems from the scale-invariance of the SI-SNR function. An interpretation of this function could be that it favors the phase difference between the signals more than the magnitude difference, due to its use of the scalar product in equation 2.16. This might cause the model to achieve good phase accuracy but non-optimal magnitude accuracy. This is the problem that MSE might solve, as it only looks at magnitude differences and will condition the model to achieve similar magnitude levels.

5.2.3 Results

The evaluation was done on the DSD100 test dataset and SDR was calculated using the mir_eval library. All the songs were loaded like the sequential training data, and the outputs of the models were then combined to create a reconstructed signal. The SDR was then taken of this full reconstructed signal. For the vocals the original Wave-U-Net method of generation was also tested, by subtracting the generated instrumental track from the mixture. The results from these are called subtracted vocals, (Sub. Vocals), while the generated vocals are called (Gen. Vocals).

	Instruments		Gen. Vocal		Sub. Vocal	
Models	mean	SD	mean	SD	mean	SD
M1	9.69	2.55	1.31	2.84	1.52	2.73
M2	9.84	2.55	**2.60**	2.72	**2.59**	2.72
M3	7.54	1.76	1.73	2.61	1.41	2.50
M5	**9.89**	2.50	1.42	2.63	1.22	2.26
M6	5.97	1.21	0.96	2.43	-2.86	2.01
M7	6.31	1.47	1.61	2.57	-3.51	2.00

Table 5.2: Wave-U-Net mean SDR (dB) and standard deviation (SD), Instrument is the instrumental track generated by the network, Gen. Vocal is the vocal track generated by the network, Sub. Vocal is generated by subtracting the generated Instrument track from the mixture

As said before higher SDR means better separation of the sources, and in table 5.2 it can be seen that all models reach much better results than the Chimera model in table 5.1.

However, these results are still shy of the results produced by the original Wave-U-Net [58]. Though it should be noted that all these results are from models that have only been trained for 100 epochs as opposed to the unknown amount of epochs the original Wave-U-Net was trained. It can also be seen that the models trained with random loudness and fixed SNR data performed significantly worse in this test than the others that were trained with the native SNR of the songs. These same models also obtained very poor results on the subtracted vocal tracks, which indicates that there was still a lot of vocal information in the instrumental track. It can also be seen that models using MSE did better than the model using purely SI-SNR as a loss. Because of this it is suspected that in **M5**, which used hybrid loss of MSR and SI-SNR, the MSE is mostly responsible for the performance increase for the instrument separation. **M2** was the model using the hyper-parameters from the paper, which might be why it is performing so well on the vocal separation.

5.3 HydraNet Evaluation

HydraNet was evaluated in a similar way as Wave-U-Net, with some modifications. The three loss function strategies were also tested with Hydranet, only MSE, only SI-SNR and hybrid training regimen with MSE and SI-SNR. Only the ADAM optimizer was used for the training. These models all used 12 upsampling and downsampling layers, and a channel increment of 24.
The time sensitive HydraNet was tested with 4 upsampling and downsampling layers, and a channel increment of 24, and the hybrid training regimen.
All models were trained with a batch size of 50.

5.3.1 Training Procedure

The first model (**M1**) used MSE loss and a learning rate of 10^{-4}, decay of 10^{-6}, β_1 of 0.5 and β_2 of 0.999 were used. For the second model (**M2**) SI-SNR was used. The third model (**M3**) used the hybrid training regimen. Both **M2** and **M3** were trained with a learning rate of 10^{-4}, β_1 of 0.9 and β_2 of 0.999 were used.
M1 was trained for 400 epochs, and **M2** and **M3** for 100 epochs. However, to make comparison with Wave-U-Net fair the results from the 100th epoch of **M1** and **M3** will also be shown in table 5.3.

A fourth model (**M4**) was also trained with the same parameters and loss as **M3**, however every 30 epochs the training data was generated again, creating a new novel dataset for the model to train on. The reasoning for this was that continuously giving the model new data would prevent overfitting and generalise the model more. This approach is akin to curriculum learning[5], where a model is trained on simple data to begin with, and then trained with new, more difficult data once a threshold has been reached. The major difference here is that in our case the new data is not more "difficult" to learn, and thus a threshold technique is most likely no better than a simple timed technique that is

used here. This model was trained for 300 epochs, and as with the other models the results from the 100th epoch will also be shown.

The fifth model (**M5**) was the time sensitive HydraNet, seen in figure 4.3b. It was trained exactly like **M4**, though only for 40 epochs due to time-constraints.

M3 to **M5** were all trained using fixed SNR data generation. **M2** used native SNR generation.

5.3.2 Results

All of the models were evaluated on the DSD100 dataset with the sane procedure as Wave-U-Net. The "Sub." tracks were generated by subtracting the estimated instrument track from the mixture. This was done to compare the results with the Wave-U-Net implemented in this project, and with the Wave-U-Net paper. This also gives a good measure for interference, how much of the vocals are still in the instrument signal after separation. If there still is some then, as we can see in table 5.2, the SDR will be low in the subtracted signal.
The results are shown in table 5.3.

		Instruments		Gen. Vocal		Sub. Vocal	
Model	epoch	mean	SD	mean	SD	mean	SD
M1	100	**9.33**	2.57	2.16	2.63	**2.17**	2.61
M2	100	8.28	1.97	2.16	2.54	1.42	2.41
M3	100	8.93	2.02	2.51	2.57	-2.82	2.05
M4	100	8.94	2.03	**2.79**	2.53	-2.96	2.05
M5	40	8.97	2.40	2.26	2.49	0.75	2.71
M1	400	**9.83**	2.60	2.07	2.72	2.15	2.66
M4	300	9.78	2.13	**3.46**	2.52	-2.67	2.10

Table 5.3: HydraNet mean SDR (dB) and standard deviation (SD), Instrument is the instrumental track generated by the network, Gen. Vocal is the vocal track generated by the network, Sub. Vocal is generated by subtracting the generated Instrument track from the mixture

The results show that the model that used MSE loss, **M1** is much better at separating instruments than its counterparts that use SI-SNR or hybrid loss. However, it also shows that the SI-SNR, and hybrid loss, are much better for generating vocals. This is especially seen with **M4**, both at 100 epochs and 300 epochs it outperforms the other models in this measure, even when having only been trained for 300 epochs rather than the 400 epochs of **M1**. It is also shown that the short HydraNet, **M5** performs equally well to models trained for 100 epochs when it has only been trained for 40. **M5** also has significantly higher SDR in the subtracted vocals section, compared to the other methods that use hybrid loss. In this regard the other hybrid loss models, **M3** and **M4**, perform surprisingly poorly which

suggests that there is still a significant amount of vocals in the instruments track, or that the instruments track is somehow not captured fully these methods.

5.4 State-Of-The-Art Comparisons

The best models from Chimera, Wave-U-Net and HydraNet were taken and compared to the results of the papers they were based on, and other state-of-the-art papers. This can be seen in table 5.4.

		Instruments		Vocal	
Model	Epoch	mean	SD	mean	SD
Chimera [43]	-	8.1	-	6.1	-
Wave-U-Net [58]	-	11.9	7.0	0.7	13.7
TAK-3 [60]	-	**12.7**	-	**6.1**	-
2DFT [56]	-	5.1	2.5	2.7	1.6
MELO [53]	-	-	-	3.7	5.0
ChimeraNet*	70	-20.8	5.5	-34.7	1.9
Wave-U-Net* M2	100	9.8	2.6	2.6	2.7
HydraNet M1	400	9.8	2.6	2.1	2.7
HydraNet M4	300	9.8	2.1	3.5	2.5

Table 5.4: Overall result comparisons of mean SDR and standard deviation (SD), the "-" are values that the papers do not state, thus they are unknown. The models with stars "*" are the models implemented in this paper

As can be seen in the table, the models evaluated in this project fall in the middle of the performance field, with the exception of the Chimera model which performs very poorly. HydraNet and Wave-U-Net surpass some of the other papers, such as the Chimera paper and 2DFT, in instruments separation, while also surpassing the Wave-U-Net paper in vocal separation. It is uncertain if this projects Wave-U-Net performance is generally better than the original papers performance, as the paper reports a 13.7dB standard deviation, meaning that some songs will have much better separation SDR than our best SDR. As can also be seen in the table, the number of epochs each model was trained is unknown, which makes it very difficult to compare results as the number of epochs they used might be orders of magnitude larger than what we have used. Because of this the two models Wave-U-Net and HydraNet should be trained more to see if they can reach the same level of SDR as the state of the art papers. It can also be seen in the table that the HydraNet **M4** almost reaches the same level of vocal separation as MELO[53], an algorithm specifically for separating vocals from songs, but which does not separate the instruments as HydraNet does.

Chapter 6

Discussion

In this chapter the results will be discussed more, along with the possible causes for these results and things that could be done to improve the models.

6.1 Main Findings

The main findings of the paper are that considering the low training time, the HydraNet and Wave-U-Net models did surprisingly well, and that if trained more they could reach the state-of-the-art performance of other papers. It is also thought that the performance of the models might be increased by separating the instruments into their separate tracks, and training the networks to estimate all the instruments, instead of a single track. It is thought this might improve performance since all models trained with the hybrid loss function and non-native generation methods, performed very poorly on the subtracted vocal track. This was a track that was generated by subtracting the estimated instruments track from the mixture, thus ideally leaving only vocals. However, as seen in tables 5.2 and 5.3, these models performed badly on those tracks. The reason for this will be discussed in section 6.5.

Another finding was that all trained versions of Chimera performed very badly. This is also surprising since the short HydraNet **M5** was trained for a similar amount of time and achieved almost the same SDR as models that had been trained longer. A possible explanation for this lapse between the Chimera paper stated SDR and the SDR of the Chimera model in this paper could be that the Chimera model implemented here was altered slightly to make it less memory intensive, and make training much faster. This modification and its effects will be discussed more in section 6.2.

It was also surprising how quickly the short HydraNet **M5** actually learned to distinguish instruments and singing voices, as it was assumed beforehand that this would probably take many 100's of epochs. This was thought due to the complexity of the prob-

lem, and the amount of overlap between singing voices and instruments, which might confuse networks. It should be noted here that the tracks the networks performed worst on were often tracks where vocals and instruments were in the same frequency range, such as heavy metal and techno music. This result was obtained qualitatively finding the test tracks that got the lowest SDR or SI-SNR and listening to them.

6.2 Modified model embedding space

One reason the modified Chimera model is performing worse than the version in the paper, might be the modification of the third layer in the model. As seen in figure 4.1b the third layer is array of dense layers connected to the BLSTM layers. As stated in section 4.2.1 the purpose of this layer is to reconstruct the frequency bins that were translated into a latent space by the BLSTM layers. The second purpose of the layer is to also create an D dimensional embedding for each reconstructed bin, to create a bigger latent space for the k-means and softmax heads to classify. However, this is done by each of the dense channels based on the same embedding for each time-step, meaning that each channel has to extract a different thing from the same L dimensional embedding. The embedding also has to contain enough information for this process. This is opposed to the original implementation where each frequency bin, for each time-step, was transformed to an L dimensional embedding, which could then be translated to D dimensions by the next time-distributed dense layer. The requirement for the L dimensional embedding to contain more information in the modified model is probably detrimental to the accuracy and will probably require a longer training time to get similar results to the original model. This could be one of the reasons for the extremely poor results. However this cannot be properly tested by training the original model, due to its extreme training times.

6.3 Different datasets

Another reason why the results of this project are different from, or worse than, other papers might be because the other papers were evaluated on a slightly larger dataset, the MUSDB18[59] dataset, which combined the DSD100[41] dataset with some tracks from the MedleyDB[8] dataset. The MUSDB18 dataset takes 50 tracks from MedleyDB and adds them to 100 tracks in DSD100. These tracks were mostly added to the training data, and the total amount of test songs, 50, remained the same. Thus it is unlikely that these songs could account solely for the differences in SDR between this projects models and the models of the other papers. However, in future this dataset should be used for training and evaluation.

6.4 Data Generation

By doing a random generation algorithm instead of training on slices of the original tracks, this project might have simplified the dataset and made it easier for the models to learn, but harder for them to master. This is because the original instruments with their belonging vocals have rhythmic features that might not be represented by the generated dataset. What is meant by this is that the vocals and instruments of the songs might have moments where both vocals and instruments increase and decrease in volume together, and where they have similar spectra of frequencies. However, this is not necessarily retained when pairing random vocals with random instruments, as the similarities will not be guaranteed anymore. To fix this while still keeping silent sections out of the dataset, the silent sections should be found in both instrumental and vocal tracks, and these silent sections should be discarded for both tracks. This will heavily reduce the dataset though, as there are often silent sections in vocal tracks.

6.5 Generated Vocals And Subtracted Vocals

As discussed, the HydraNet and Wave-U-Net were okay at generating instruments and vocal tracks, while some of the models did not perform well on the subtracted vocals tracks, though some did. This suggests that there is either still a lot of vocal content in the generated instruments track, or that the instruments track is not being reconstructed well. This reconstruction could either simply be a decrease, or increase, in magnitude of the signal, which would either take too little of the instruments from the mixture or too much. If that is the case then it might be due to the tracks being shifted too much in the dB range during the random loudness and fixed SNR generation. This is consistent with the results, as it was only models trained with these methods that performed poorly on those tracks. With these generation methods the models might not learn a good estimate of the loudness that should be produced, as they are trained on tracks that are between 0 and 30dB, while normal loudness tracks might be -10dB to 10dB. If this is the case, this is clearly an error in the generation algorithm that should be fixed.

Chapter 7

Conclusion

From the results it can be concluded that neither the Wave-U-Net, nor the HydraNet reached state-of-the art performance in instrument separation, though they did surpass the original Wave-U-Net results in vocal separation.
That the Wave-U-Net did not reach the same level of instrument SDR as the paper, is indicative of the network not being trained for long enough. Thus the same can be said for the HydraNet, and it can be concluded that these networks should get longer training to test if they can reach state-of-the art performance. It can also be concluded that the hybrid training regimen used for training HydraNet **M4** did have a significant effect on the vocal separation SDR, when compared to the other models that were trained for the same amount of epochs.

Not being trained for enough epochs is most likely also the cause of the Chimera models poor performance, though it is still uncertain what exactly factors contributed to this.

The final conclusion of this project is that single-channel blind source separation, which used to be an extremely difficult problem, is now becoming a field that can be solved easily by these hybrid networks consisting of convolutional, dense and recurrent layers. Thus more focus should be put on these models, and how to develop more accurate and fast models, like TasNet, such that they can be used in real-time applications. More research should also be focused on making these models more general, not only solving singing voice separation from music, but also separating EEG readings, or noisy recordings, and furthering usefulness of this field in many other areas research and in daily life.

7.1 Acknowledgements

I would like to thank my supervisors Tsampikos Kounalakis and Cumhur Erkut, who have guided me through this project and field, and steered me clear of bad ideas. I would also like to thank Hendrik Purwins for introducing me to the field, and for helping throughout the project with useful papers and theories.

Chapter 8

Future Works

8.1 Change Generation Methods

The datset generation methods of random loudness and fixed SNR should be changed such that the loudness of the entire signal never surpasses a certain determined threshold which will be found experimentally. It should also be changed to analyze the distribution of volumes of the dataset it generates from, and try to generate tracks with similar loudness. The generation method could even be altered to find segments with similar spectra and pair them together, to create a more difficult dataset to solve.

8.2 Single Output For Wave-U-Net And HydraNet

An interesting experiment for Wave-U-Net and HydraNet could be to follow the Wave-U-Net paper and generate one less source than is required, and then define the last source as the sum of the other generated sources. In this way the other sources might learn more about the missing source than if all sources are predicted.

8.3 Multi-Instrument Training

Another experiment that could be done is to train HydraNet to predict all instruments as well as the vocals, to see if the network can achieve better results in the same amount of time, 100 epochs. This might teach the network more about the different spectra of the instruments and might help it seperate the spectra better than now.

8.4 Dual Training With Voices

This is a similar idea to training with all the instruments to learn their spectra, in this case the suggestion is simply to also use another spoken voice in the signal. This will make the

network have to solve both singing voice separation and the cocktail-party problem at the same time, and might lead to some interesting results. The hope is that this will teach the network more about the characteristics of human voices, and might enable it to separate out back-up singers in some songs, or noise from other humans in a concert recording.

8.5 Deep Clustering Network

Another interesting experiment would be to make the HydraNet into an deep clustering network that uses the error functions defined in [24] to create embeddings for the waveform that can then be clustered by a clustering algorithm. The advantage of this is that the number of sources in the signal will be determined by the number of clusters in the space, nothing in the model needs to be changed to handle more or fewer sources.

8.6 3D Latent Space

As the HydraNet is now, the latent space generated at the bottom of the model is 2D, however it could easily be made into a 3D latent space, just like it is in the Chimera model. By doing this more information could be retrieved from the space, however the risk of overfitting the network is also increased due the the larger amount of parameters the model would have.

Bibliography

[1] Shoko Araki et al. "Underdetermined blind sparse source separation for arbitrarily arranged multiple sensors". In: *Signal Processing*. Independent Component Analysis and Blind Source Separation 87.8 (Aug. 2007), pp. 1833–1847. ISSN: 0165-1684. DOI: 10.1016/j.sigpro.2007.02.003. URL: http://www.sciencedirect.com/science/article/pii/S0165168407000680 (visited on 05/25/2019).

[2] Suzanna Becker and Mark Plumbley. "Unsupervised neural network learning procedures for feature extraction and classification". en. In: *Applied Intelligence* 6.3 (July 1996), pp. 185–203. ISSN: 1573-7497. DOI: 10.1007/BF00126625. URL: https://doi.org/10.1007/BF00126625 (visited on 05/26/2019).

[3] Anthony J. Bell and Terrence J. Sejnowski. "An Information-Maximization Approach to Blind Separation and Blind Deconvolution". In: *Neural Computation* 7.6 (Nov. 1995), pp. 1129–1159. ISSN: 0899-7667. DOI: 10.1162/neco.1995.7.6.1129. URL: https://doi.org/10.1162/neco.1995.7.6.1129 (visited on 05/25/2019).

[4] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[5] Yoshua Bengio et al. "Curriculum Learning". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML '09. event-place: Montreal, Quebec, Canada. New York, NY, USA: ACM, 2009, pp. 41–48. ISBN: 978-1-60558-516-1. DOI: 10.1145/1553374.1553380. URL: http://doi.acm.org/10.1145/1553374.1553380 (visited on 05/23/2019).

[6] C. D. Binnie and P. F. Prior. "Electroencephalography." en. In: *Journal of Neurology, Neurosurgery & Psychiatry* 57.11 (Nov. 1994), pp. 1308–1319. ISSN: 0022-3050, 1468-330X. DOI: 10.1136/jnnp.57.11.1308. URL: https://jnnp.bmj.com/content/57/11/1308 (visited on 05/19/2019).

[7] Christopher M. Bishop and Professor of Neural Computing Christopher M. Bishop. *Neural Networks for Pattern Recognition*. en. Google-Books-ID: T0S0BgAAQBAJ. Clarendon Press, Nov. 1995. ISBN: 978-0-19-853864-6.

[8] Rachel M. Bittner et al. "MedleyDB: A Multitrack Dataset for Annotation-Intensive MIR Research." In: *ISMIR*. Vol. 14. 2014, pp. 155–160.

[9] A. J. Casson et al. “Wearable Electroencephalography”. In: *IEEE Engineering in Medicine and Biology Magazine* 29.3 (May 2010), pp. 44–56. ISSN: 0739-5175. DOI: 10.1109/MEMB.2010.936545.

[10] E. Colin Cherry. “Some Experiments on the Recognition of Speech, with One and with Two Ears”. In: *The Journal of the Acoustical Society of America* 25.5 (Sept. 1953), pp. 975–979. ISSN: 0001-4966. DOI: 10.1121/1.1907229. URL: https://asa.scitation.org/doi/abs/10.1121/1.1907229 (visited on 05/19/2019).

[11] Daniel P. W. Ellis. “Prediction-driven computational auditory scene analysis”. en. PhD thesis. Columbia University, 1996. DOI: 10.7916/D84J0N13. URL: https://doi.org/10.7916/D84J0N13 (visited on 05/25/2019).

[12] Jeffrey L. Elman. “Finding structure in time”. In: *Cognitive science* 14.2 (1990), pp. 179–211.

[13] H. Erdogan et al. “Phase-sensitive and recognition-boosted speech separation using deep recurrent neural networks”. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Apr. 2015, pp. 708–712. DOI: 10.1109/ICASSP.2015.7178061.

[14] Jean Baptiste Joseph baron Fourier. *The Analytical Theory of Heat*. en. The University Press, 1878.

[15] Cédric Févotte, Emmanuel Vincent, and Alexey Ozerov. “Single-Channel Audio Source Separation with NMF: Divergences, Constraints and Algorithms”. en. In: *Audio Source Separation*. Ed. by Shoji Makino. Signals and Communication Technology. Cham: Springer International Publishing, 2018, pp. 1–24. ISBN: 978-3-319-73031-8. DOI: 10.1007/978-3-319-73031-8_1. URL: https://doi.org/10.1007/978-3-319-73031-8_1 (visited on 03/01/2019).

[16] Sharon Gannot et al. “A Consolidated Perspective on Multimicrophone Speech Enhancement and Source Separation”. In: *IEEE/ACM Trans. Audio, Speech and Lang. Proc.* 25.4 (Apr. 2017), pp. 692–730. ISSN: 2329-9290. DOI: 10.1109/TASLP.2016.2647702. URL: https://doi.org/10.1109/TASLP.2016.2647702 (visited on 05/26/2019).

[17] Emad M. Grais et al. “Single-Channel Audio Source Separation Using Deep Neural Network Ensembles”. English. In: Audio Engineering Society, May 2016. URL: http://www.aes.org/e-lib/online/browse.cfm?elib=18193 (visited on 02/05/2019).

[18] Alex Graves. “Generating Sequences With Recurrent Neural Networks”. In: *arXiv:1308.0850 [cs]* (Aug. 2013). arXiv: 1308.0850. URL: http://arxiv.org/abs/1308.0850 (visited on 05/26/2019).

[19] D. Griffin and Jae Lim. “Signal estimation from modified short-time Fourier transform”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 32.2 (Apr. 1984), pp. 236–243. ISSN: 0096-3518. DOI: 10.1109/TASSP.1984.1164317.

[20] D. Gunawan and D. Sen. "Iterative Phase Estimation for the Synthesis of Separated Sources From Single-Channel Mixtures". In: *IEEE Signal Processing Letters* 17.5 (May 2010), pp. 421–424. ISSN: 1070-9908. DOI: 10.1109/LSP.2010.2042530.

[21] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. 1st. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1994. ISBN: 978-0-02-352761-6.

[22] Pengju He et al. "Single channel blind source separation on the instantaneous mixed signal of multiple dynamic sources". In: *Mechanical Systems and Signal Processing*. SI: IMETI-MechElectro 113 (Dec. 2018), pp. 22–35. ISSN: 0888-3270. DOI: 10.1016/j.ymssp.2017.04.004. URL: http://www.sciencedirect.com/science/article/pii/S0888327017301905 (visited on 03/01/2019).

[23] Robert Hecht-nielsen. "III.3 - Theory of the Backpropagation Neural Network". In: *Neural Networks for Perception*. Ed. by Harry Wechsler. Academic Press, Jan. 1992, pp. 65–93. ISBN: 978-0-12-741252-8. DOI: 10.1016/B978-0-12-741252-8.50010-8. URL: http://www.sciencedirect.com/science/article/pii/B9780127412528500108 (visited on 05/26/2019).

[24] J. R. Hershey et al. "Deep clustering: Discriminative embeddings for segmentation and separation". In: *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Mar. 2016, pp. 31–35. DOI: 10.1109/ICASSP.2016.7471631.

[25] John R. Hershey et al. "Deep clustering: Discriminative embeddings for segmentation and separation". In: *arXiv:1508.04306 [cs, stat]* (Aug. 2015). arXiv: 1508.04306. URL: http://arxiv.org/abs/1508.04306 (visited on 01/06/2019).

[26] Geoffrey E Hinton and Richard S. Zemel. "Autoencoders, Minimum Description Length and Helmholtz Free Energy". In: *Advances in Neural Information Processing Systems 6*. Ed. by J. D. Cowan, G. Tesauro, and J. Alspector. Morgan-Kaufmann, 1994, pp. 3–10. URL: http://papers.nips.cc/paper/798-autoencoders-minimum-description-length-and-helmholtz-free-energy.pdf (visited on 05/26/2019).

[27] Sepp Hochreiter. "The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 06.02 (Apr. 1998), pp. 107–116. ISSN: 0218-4885. DOI: 10.1142/S0218488598000094. URL: https://www.worldscientific.com/doi/abs/10.1142/S0218488598000094 (visited on 05/26/2019).

[28] P. Huang et al. "Singing-voice separation from monaural recordings using robust principal component analysis". In: *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Mar. 2012, pp. 57–60. DOI: 10.1109/ICASSP.2012.6287816.

[29] Eric J. Humphrey et al. "Mining Labeled Data from Web-Scale Collections for Vocal Activity Detection in Music." In: *ISMIR*. 2017, pp. 709–715.

[30] Yusuf Isik et al. “Single-Channel Multi-Speaker Separation using Deep Clustering”. In: *arXiv:1607.02173 [cs, stat]* (July 2016). arXiv: 1607.02173. URL: http://arxiv.org/abs/1607.02173 (visited on 02/05/2019).

[31] A. Jansson et al. *Singing voice separation with deep U-Net convolutional networks*. en. conference. Suzhou, China, Oct. 2017. URL: https://ismir2017.smcnus.org/ (visited on 03/01/2019).

[32] Tzyy-Ping Jung et al. “Removing electroencephalographic artifacts by blind source separation”. en. In: *Psychophysiology* 37.2 (Mar. 2000), pp. 163–178. ISSN: 1469-8986, 0048-5772. URL: https://www.cambridge.org/core/journals/psychophysiology/article/removing-electroencephalographic-artifacts-by-blind-source-separation/2548D35629CAE17E6956C2FFF1B6C8AB (visited on 05/26/2019).

[33] L. G. Kiloh, A. J. McComas, and J. W. Osselton. *Clinical Electroencephalography*. en. Butterworth-Heinemann, Oct. 2013. ISBN: 978-1-4831-9215-4.

[34] H. Kim et al. “Single channel blind source separation based on probabilistic matrix factorisation”. In: *Electronics Letters* 53.21 (2017), pp. 1429–1431. ISSN: 0013-5194. DOI: 10.1049/el.2017.2013.

[35] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *arXiv:1412.6980 [cs]* (Dec. 2014). arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980 (visited on 05/27/2019).

[36] Morten Kolbaek et al. “Multitalker Speech Separation With Utterance-Level Permutation Invariant Training of Deep Recurrent Neural Networks”. In: *IEEE/ACM Trans. Audio, Speech and Lang. Proc.* 25.10 (Oct. 2017), pp. 1901–1913. ISSN: 2329-9290. DOI: 10.1109/TASLP.2017.2726762. URL: https://doi.org/10.1109/TASLP.2017.2726762 (visited on 02/19/2019).

[37] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf (visited on 05/26/2019).

[38] Yann LeCun and Yoshua Bengio. “Convolutional networks for images, speech, and time series”. In: *The handbook of brain theory and neural networks* 3361.10 (1995), p. 1995.

[39] L. Li and H. Kameoka. “Deep Clustering with Gated Convolutional Networks”. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Apr. 2018, pp. 16–20. DOI: 10.1109/ICASSP.2018.8461746.

[40] Yevgeni Litvin and Israel Cohen. “Single-channel source separation of audio signals using Bark Scale Wavelet Packet Decomposition”. In: Oct. 2009, pp. 1–4. DOI: 10.1109/MLSP.2009.5306232.

[41] Antoine Liutkus et al. “The 2016 Signal Separation Evaluation Campaign”. en. In: *Latent Variable Analysis and Signal Separation*. Ed. by Petr Tichavský et al. Lecture Notes in Computer Science. Springer International Publishing, 2017, pp. 323–332. ISBN: 978-3-319-53547-0.

[42] Y. Luo and N. Mesgarani. “TaSNet: Time-Domain Audio Separation Network for Real-Time, Single-Channel Speech Separation”. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Apr. 2018, pp. 696–700. DOI: 10.1109/ICASSP.2018.8462116.

[43] Y. Luo et al. “Deep clustering and conventional networks for music separation: Stronger together”. In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Mar. 2017, pp. 61–65. DOI: 10.1109/ICASSP.2017.7952118.

[44] Yi Luo and Nima Mesgarani. “TasNet: Surpassing Ideal Time-Frequency Masking for Speech Separation”. In: *arXiv:1809.07454 [cs, eess]* (Sept. 2018). arXiv: 1809.07454. URL: http://arxiv.org/abs/1809.07454 (visited on 02/27/2019).

[45] P. Magron, R. Badeau, and B. David. “Model-Based STFT Phase Recovery for Audio Source Separation”. In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 26.6 (June 2018), pp. 1095–1105. ISSN: 2329-9290. DOI: 10.1109/TASLP.2018.2811540.

[46] Brian McFee et al. “librosa: Audio and music signal analysis in python”. In: *Proceedings of the 14th python in science conference*. 2015, pp. 18–25.

[47] T. Moon et al. “RNNDROP: A novel dropout for RNNS in ASR”. In: *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. Dec. 2015, pp. 65–70. DOI: 10.1109/ASRU.2015.7404775.

[48] Francesco Negro et al. “Multi-channel intramuscular and surface EMG decomposition by convolutive blind source separation”. en. In: *Journal of Neural Engineering* 13.2 (Feb. 2016), p. 026027. ISSN: 1741-2552. DOI: 10.1088/1741-2560/13/2/026027. URL: https://doi.org/10.1088%2F1741-2560%2F13%2F2%2F026027 (visited on 05/26/2019).

[49] H. Nyquist. “Certain Topics in Telegraph Transmission Theory”. In: *Transactions of the American Institute of Electrical Engineers* 47.2 (Apr. 1928), pp. 617–644. ISSN: 0096-3860. DOI: 10.1109/T-AIEE.1928.5055024.

[50] Erkki Oja. “Neural networks, principal components, and subspaces”. In: *International Journal of Neural Systems* 01.01 (Jan. 1989), pp. 61–68. ISSN: 0129-0657. DOI: 10.1142/S0129065789000475. URL: https://www.worldscientific.com/doi/abs/10.1142/S0129065789000475 (visited on 05/26/2019).

[51] Colin Raffel et al. “mir_eval: A transparent implementation of common MIR metrics”. In: *In Proceedings of the 15th International Society for Music Information Retrieval Conference, ISMIR*. Citeseer, 2014.

[52] Brian D. Ripley and N. L. Hjort. *Pattern Recognition and Neural Networks*. en. Google-Books-ID: 2SzT2p8vP1oC. Cambridge University Press, Jan. 1996. ISBN: 978-0-521-46086-6.

[53] J. Salamon and E. Gomez. "Melody Extraction From Polyphonic Music Signals Using Pitch Contour Characteristics". In: *IEEE Transactions on Audio, Speech, and Language Processing* 20.6 (Aug. 2012), pp. 1759–1770. ISSN: 1558-7916. DOI: 10.1109/TASL.2012.2188515.

[54] H. Sawada, S. Araki, and S. Makino. "A Two-Stage Frequency-Domain Blind Source Separation Method for Underdetermined Convolutive Mixtures". In: *2007 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*. Oct. 2007, pp. 139–142. DOI: 10.1109/ASPAA.2007.4393012.

[55] M. Schuster and K. K. Paliwal. "Bidirectional recurrent neural networks". In: *IEEE Transactions on Signal Processing* 45.11 (Nov. 1997), pp. 2673–2681. ISSN: 1053-587X. DOI: 10.1109/78.650093.

[56] P. Seetharaman, F. Pishdadian, and B. Pardo. "Music/Voice separation using the 2D fourier transform". In: *2017 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*. Oct. 2017, pp. 36–40. DOI: 10.1109/WASPAA.2017.8169990.

[57] Ziqiang Shi et al. "FurcaNet: An end-to-end deep gated convolutional, long short-term memory, deep neural networks for single channel speech separation". In: *arXiv:1902.00651 [cs, eess]* (Feb. 2019). arXiv: 1902.00651. URL: http://arxiv.org/abs/1902.00651 (visited on 02/26/2019).

[58] Daniel Stoller, Sebastian Ewert, and Simon Dixon. "Wave-U-Net: A Multi-Scale Neural Network for End-to-End Audio Source Separation". In: *arXiv:1806.03185 [cs, eess, stat]* (June 2018). arXiv: 1806.03185. URL: http://arxiv.org/abs/1806.03185 (visited on 02/28/2019).

[59] Fabian-Robert Stöter, Antoine Liutkus, and Nobutaka Ito. "The 2018 Signal Separation Evaluation Campaign". en. In: *Latent Variable Analysis and Signal Separation*. Ed. by Yannick Deville et al. Lecture Notes in Computer Science. Springer International Publishing, 2018, pp. 293–305. ISBN: 978-3-319-93764-9.

[60] N. Takahashi and Y. Mitsufuji. "Multi-Scale multi-band densenets for audio source separation". In: *2017 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*. Oct. 2017, pp. 21–25. DOI: 10.1109/WASPAA.2017.8169987.

[61] E. Vincent, R. Gribonval, and C. Fevotte. "Performance measurement in blind audio source separation". In: *IEEE Transactions on Audio, Speech, and Language Processing* 14.4 (July 2006), pp. 1462–1469. ISSN: 1558-7916. DOI: 10.1109/TSA.2005.858005.

[62] Pascal Vincent et al. "Extracting and Composing Robust Features with Denoising Autoencoders". In: *Proceedings of the 25th International Conference on Machine Learning*. ICML '08. event-place: Helsinki, Finland. New York, NY, USA: ACM, 2008, pp. 1096–1103. ISBN: 978-1-60558-205-4. DOI: 10.1145/1390156.1390294. URL: http://doi.acm.org/10.1145/1390156.1390294 (visited on 05/26/2019).

[63] T. Virtanen, J. F. Gemmeke, and B. Raj. "Active-Set Newton Algorithm for Overcomplete Non-Negative Representations of Audio". In: *IEEE Transactions on Audio, Speech, and Language Processing* 21.11 (Nov. 2013), pp. 2277–2289. ISSN: 1558-7916. DOI: 10.1109/TASL.2013.2263144.

[64] J. Volkmann, S. S. Stevens, and E. B. Newman. "A Scale for the Measurement of the Psychological Magnitude Pitch". In: *The Journal of the Acoustical Society of America* 8.3 (Jan. 1937), pp. 208–208. ISSN: 0001-4966. DOI: 10.1121/1.1901999. URL: https://asa.scitation.org/doi/abs/10.1121/1.1901999 (visited on 05/25/2019).

[65] DeLiang Wang and Guy J. Brown. *Computational Auditory Scene Analysis: Principles, Algorithms, and Applications*. Wiley-IEEE Press, 2006. ISBN: 978-0-471-74109-1.

[66] Z. Wang, J. Le Roux, and J. R. Hershey. "Multi-Channel Deep Clustering: Discriminative Spectral and Spatial Embeddings for Speaker-Independent Speech Separation". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Apr. 2018, pp. 1–5. DOI: 10.1109/ICASSP.2018.8461639.

[67] Zhong-Qiu Wang, Jonathan Le Roux, and John R. Hershey. "Alternative Objective Functions for Deep Clustering". In: *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018.

[68] Zhong-Qiu Wang, Ke Tan, and DeLiang Wang. "Deep Learning Based Phase Reconstruction for Speaker Separation: A Trigonometric Perspective". In: *arXiv:1811.09010 [cs, eess]* (Nov. 2018). arXiv: 1811.09010. URL: http://arxiv.org/abs/1811.09010 (visited on 02/28/2019).

[69] Zhong-Qiu Wang et al. "End-to-End Speech Separation with Unfolded Iterative Phase Reconstruction". In: *arXiv:1804.10204 [cs, eess, stat]* (Apr. 2018). arXiv: 1804.10204. URL: http://arxiv.org/abs/1804.10204 (visited on 02/25/2019).

[70] Felix Weninger, Johannes Bergmann, and Björn Schuller. "Introducing currennt: The munich open-source cuda recurrent neural network toolkit". In: *The Journal of Machine Learning Research* 16.1 (2015), pp. 547–551.

[71] Ronald J. Williams and David Zipser. "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks". In: *Neural Computation* 1.2 (June 1989), pp. 270–280. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.2.270. URL: https://doi.org/10.1162/neco.1989.1.2.270 (visited on 05/26/2019).

[72] D. Yu et al. “Permutation invariant training of deep models for speaker-independent multi-talker speech separation”. In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Mar. 2017, pp. 241–245. DOI: 10.1109/ICASSP.2017.7952154.

[73] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. “Recurrent Neural Network Regularization”. In: *arXiv:1409.2329 [cs]* (Sept. 2014). arXiv: 1409.2329. URL: http://arxiv.org/abs/1409.2329 (visited on 01/30/2019).