Audio Seminar WS21/22

speech separation presentation: written elaboration

Johannes Schmidt

December 2021

Introduction

This document is a written elaboration of the presentation hold by Johannes Schmidt (born in Bonn 14.02.1997) in the winter semester 2021/2022 for the course 'Audio Seminar'.
The main objective of the talk and therefore this work will be the theoretical background and results of the paper [4]. Since the authors improve a method proposed by [5], the first part of the elaboration talks about their work. The images for illustrating basic concepts of neural networks are taken from [3] and [1] and the concepts themselves from [2]. As a road map of this document, here is what will be discussed:

1 The definition of the problem: Cocktail party problem

2 Used technique: Black Bock approach - Deep Learning

3 Basic principles of feed forward neural nets: special care on FCNN, CNN, RNN, LSTM (terms explained bellow)

4 Explanation of the used network: Baseline approach

5 What modifications are done in [4] doing compared to the baseline approach

6 conclusion

1 Cocktail Party Problem

The field of *speech recognition* deals with designing algorithms that convert audible speech into written text. Modern achievements made by deep learning techniques make up for more applications, because they are not human understandable. Problems in robustness and accuracy occur when there is background

noise and when the target speaker is overlaid by other speakers. The task of separating a particular speaker from a superposition of multiple speakers and noise is known as *speaker extraction*. This is our objective.
To be more precised, the setup is the following:
Given noisy data we try to extract a target speaker and additional reference audio of this target speaker, the objective (in this case) is to calculate a (soft) mask. This mask applied on the original noisy data shall yield the target speaker extracted from the noisy data.

2 Used Techniques

The proposed technique for solving this task is a deep neural network, so a black box approach. By black box one means that it is not clear to the engineer what features the algorithm actually learned for calculating the output. One can only validate its performance on some test data and estimate the accuracy.
The main focus here is to explain the building blocks of the neural network proposed by [4].
These concepts are Fully-Connected -, Convolutional - and Recurrent neural networks (FCNN, CNN, RNN), with special care to vector embeddings, representation learning and Long Short Term Memory (LSTM) cells.
In the next section these concepts will be explained in more detail.

3 Basic concepts of feed forward neural networks

3.1 FCNN

Neural networks are a class of models that are constructed out of layers. The size of the input and output are of fixed size, specific for the task. The number of hidden layers and neurons in it are so called hyperparameters. It means that they have to be chosen before one can use the network for training and making predictions. There are many heuristics for choosing the right one and also for optimizing them. Each neuron consists of a weight matrix and a bias and is connected to every neuron of the next layer. A visualization of this can seen in Figure 1. Every neuron performs the following calculation,

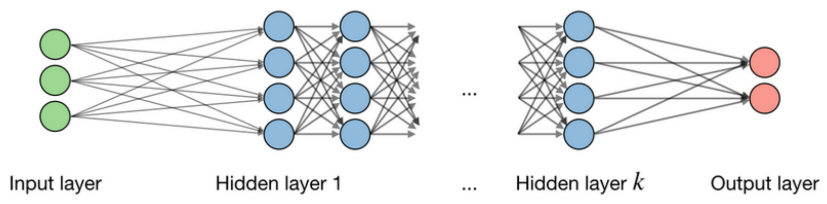


Figure 1: The rough structure of a FCNN

$$z_i^{[i]} = w_j^{[i]} x + b_j^{[i]} \tag{1}$$

with i being the index for the layer and j for the neuron. w and b stands for weight and bias, z for the output of the neuron (it might be one of the neurons that contributes to the input of the next neuron or the output of the entire network) and x the input of the neuron. So we concatenate many affine transformations after each other. For finalizing the network we need to apply an non-linear function $g(x)$ to the output of every neuron, $o_i^{[i]} = g(z_i^{[i]})$. There are many choices for that so called activation function. Some of them are listed in Figure 2.

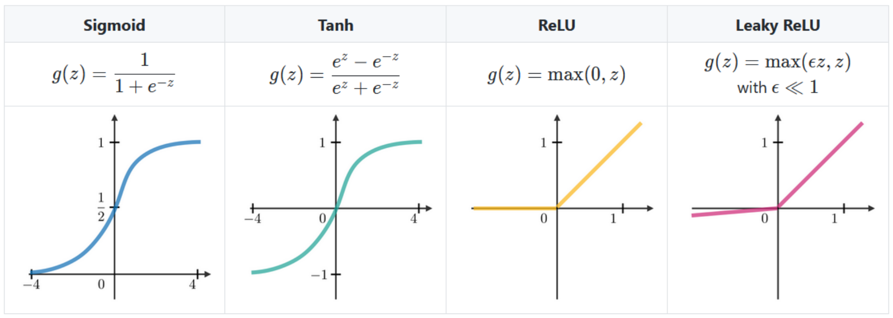


Figure 2: Some common activation functions

Before we can apply the network on some input data we first need to initialize the weights and biases.Usually this is done in a uniformly random fashion.

The procedure of training the neural network in a supervised way can be done in four steps:

Step 1 : Take a batch of (labeled) training data

Labeled data means that we have some inputs with the outputs that we want the network to map to. In our example that would be two audio files. One might be a single speaker with little noise saying something (output) and the same audio file just overlapped with some more speakers and noise (input).
Batch refers to a subset of these training examples. The two extreme cases would be showing the network all examples at once or one at a time. Usually something in between is done. There are multiple reasons for that, for example computational efficiency or fluctuating gradients that might help overcome local minima, but this goes too far for this elaboration.

Step 2 : Perform forward propagation to obtain the corresponding loss

Once the number and sizes of the layers (including input and output layer) and activation functions have been chosen and the weights and biases initialized we can apply the network on some data. This procedure is called forward propagation, since information propagates forward through the network. Since the math is clear now, multiplication/addition and applying non-linear functions, we will obtain some output. However, without any meaning or knowledge yet. So in order to improve the predictions of the network we need a measure for how well it performed to be able to compare its result with the label (what we think is true). This function is called loss function $L(z, y)$, with z being the output of the network and y the label. There are multiple possibilities for choosing one. This depends on the input, output and on what aspect of it to take special care about. We will talk later about which one was chosen in [4].

Step 3 : Backpropagate the loss to get the gradients

The process of learning is associated with updating the weights and biases such that the networks reproduces the training data (so the data we apply on the network and have the solution to). The goal is than to feed unseen data and calculate predictions. So the hope is that the network generalizes and learns the distribution describing the data, not just learns the training data by heart. Backpropagation is how we adjust our network parameters and it is based on a gradient method. This means we optimize (minimize) our loss function by adjusting the parameters of the network based on its derivative at that (data)point. Hence the name gradient descent.

Step 4 : Use the gradients to update the weights of the network

The calculated gradients can then be used to update the weights like this:

$$w \leftarrow w - \alpha \cdot \frac{\partial L(z, y)}{\partial w} \tag{2}$$

α is another hyperparameter to choose before training. Note that gradient decent is not the only optimization algorithm one can use for training a neural network, but the only one presented here.

3.2 CNN and Representation learning

CNNs are a special case of feed forward neural networks: Having sequential layers, connected neurons, non-linearities, forward propagation for applying and backpropagation for training stays the same. We now change the calculation inside the neuron and the way they are connected. Instead of an affine transformation, here we apply discrete convolutions with a collection of learnable parameters, the so called filters or kernels. And instead of connecting them all together, we only have sparse and local connections. Vividly speaking, we use the kernels to 'scan' through the input and 'search' for features. A picture of this is in Figure 3. This additional structure results in additional hyperparam-

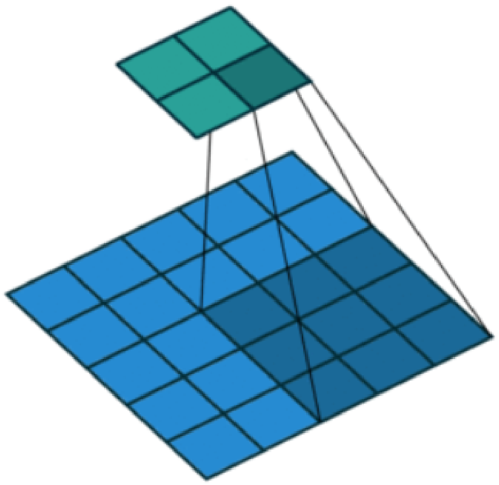


Figure 3: Visualisation of a convolution. A filter scans through an imput array

eters we have to choose before training, like the kernel size, number of filters or dilation rate. The dilation rate describes, roughly speaking, the holes in the filers, so how many blind spots there are 'inside' the image. A visual example for this, can be seen in Figure 4. Originally this technique was developed for imagine classification , so used on 2D data. In our case we will apply it on the spectrum of the noisy data, which is also an image.
However, instead of using it for classification (categorical output) we will use it here to learn a compact representation (continuous output) of the spectrum. Similar to chroma features[1] that compresses the possibility to represent frequencies between roughly 20 and 20.000 Hz in twelve containers representing the musical tone of it: $\{C, C\#, ..., B\}$. In the case of the CNN, however, the resulting features were fitted by the learning algorithm, so there is no human intuition to it. We can only evaluate the output. Still, it has many benefits to use that technique. We will later see, how we will use that compressed representation of the spectrum.

[1]For a clear introduction of that concept check: "Fundamentals of Music Processing" by Meinard Mueller, chapter 3.2.1

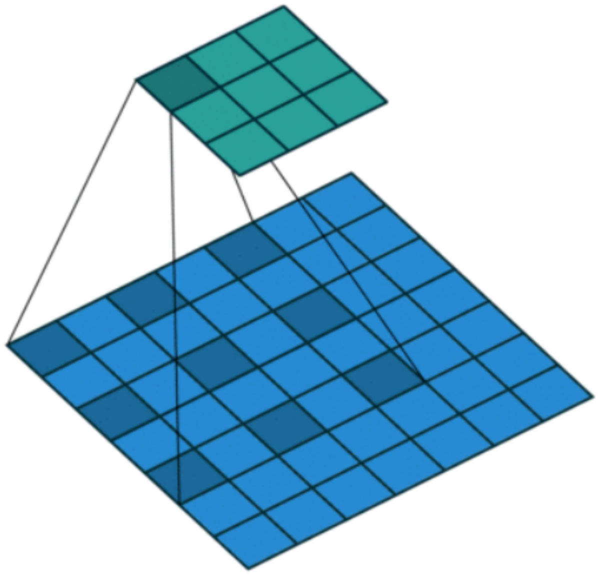


Figure 4: Visualisation of a convolution. A filter scans through an input array

3.3 RNN; LSTM

In RNNs we have another special case of neural networks. In a FCNN there is no information transmitted when the network is applied on two different data points. RNNs differ from that, they additionally provides the network with a hidden state that is feed back into the network. This allows the network to have in general sequential input and output. This is perfect for our application, since an audio file is sequential, so it does matter at which time a certain feature appears.
When studying RNNs one main problem one faces is that gradients vanish or explode, since the same parts or the network appear many times in the back-propagation process. One particular way of dealing with these gradients is a special architecture called LSTM. This is the one used in [1] and thus explained now in some more detail:

A LSTM Network consists of the concatenation of multiple cells, the LSTM cells. They all inherit the same structure, which can be seen in Figure 5. It consists of three gates (forget, input, output) and its (cell) state.
The cell state behaves like the memory of the network with the ability to retain information through time. The gates shape the information flow going into the next hidden state and cell state. Each gate in 5 contains learnable parameters, again in the form of weight matrices. The intuition of the gates is the following:

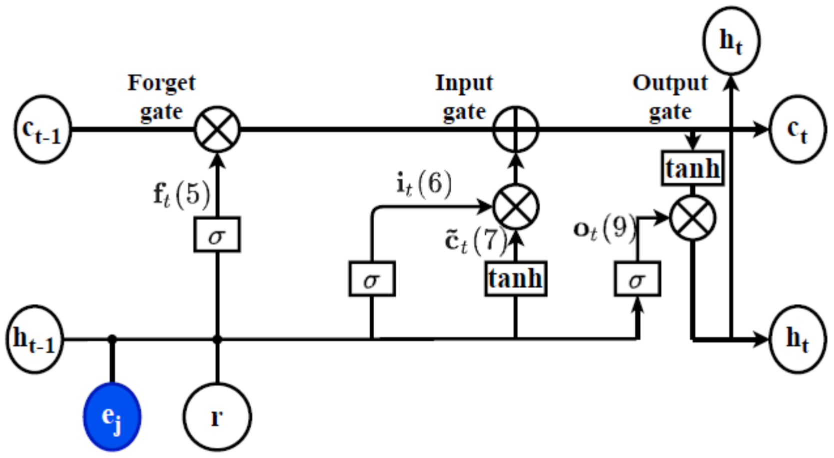


Figure 5: The schematic of a LSTM cell (for our special case of the input consisting of two components. The output of the convolution block r and the speaker encoder e_j)

the forget gate regulates how much of the last cell state contributes to the next. The input gate decides which information is updated and stored in the cell state. And the output gate decides witch part of the cell state is transferred to the next hidden state.
The main paper, however, customizes these LSTM cell structure a bit. More about that in the section called "Modfications made by [4]"
In [5] they also test the performance of so called bidirectional LSTMs. This improves the long term memory and consider information from future and past. Further details on that technique are beyond the scope of this document.

3.4 vector embeddings

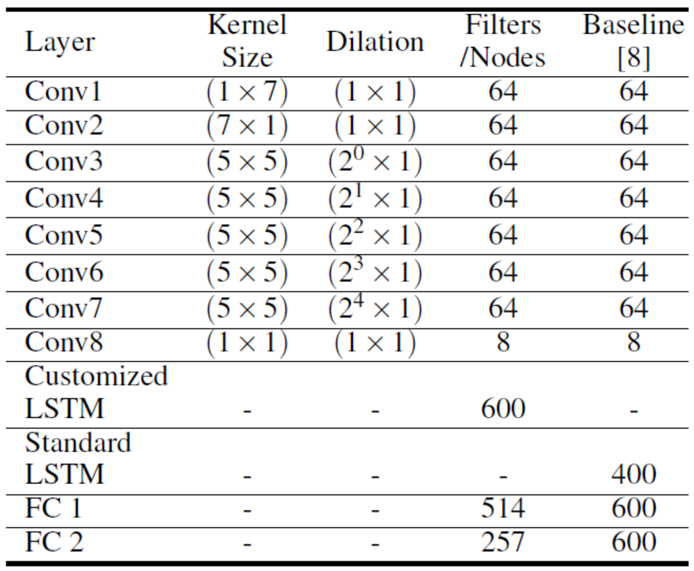
The last 'basic' concept we need to introduce is the concept of vector embeddings. Here is just intuition given instead of mathematical descriptions.
The goal of embeddings is that we want to encode data in a representation that 'similar' data points are close to each other. Closeness in this case means a distance measure on vector spaces, hence we are talking about vector embeddings. So what is meant with similar then?
An easy example for a handcrafted embedding might be sorting people by their age in a line. This would represent one (human understandable) feature, namely the age, where people with similar age are close to each other relative to others with very different age. The other extreme would be a one-hot representation, where there is no relation. One-hot means, that every data point gets its own dimension in a vector space. When choosing the scalar product as a similarity

measure, we would get 0 correlation between all entities.
One can imagine now that by forcing a model to use less and less dimensions to represent the data points some points get closer to each other than others. The hope is than that data points with similar features are close to each other. A practical way to find such feature embeddings is to just interpret the outputs of a hidden layer of dimension d as a vector embedding of a d dimensional vector space.

4 Baseline Approach

In the introduction of [4] the authors state that their work is an extension of (Voice Filter)[5]. So it is convenient to first present the baseline approach (this section) and than show what modifications have been made (next section).

In Figure 6 we can see the network architecture of it. First 8 CNN layers



Layer	Kernel Size	Dilation	Filters /Nodes	Baseline [8]
Conv1	(1×7)	(1×1)	64	64
Conv2	(7×1)	(1×1)	64	64
Conv3	(5×5)	$(2^0 \times 1)$	64	64
Conv4	(5×5)	$(2^1 \times 1)$	64	64
Conv5	(5×5)	$(2^2 \times 1)$	64	64
Conv6	(5×5)	$(2^3 \times 1)$	64	64
Conv7	(5×5)	$(2^4 \times 1)$	64	64
Conv8	(1×1)	(1×1)	8	8
Customized LSTM	-	-	600	-
Standard LSTM	-	-	-	400
FC 1	-	-	514	600
FC 2	-	-	257	600

Figure 6: Network architecture of both papers compared. (In the last row the [8] is equivalent to [5] here)

reshaping the spectrum followed by a LSTM network consisting of 400 cells finished by two layers of FCNNs.
We have two separate networks here working together, the *Speaker Encoder LSTM* and the *VoiceFilter*.

The Speaker Encoder LSTM converts the reference audio to a vector embedding version of it, here called d-vector. The embedding only goes into the

LSTM not the CNNs of our VoiceFilter network. This makes sense because we assume that the embedding is already in a compact form and has to be treated in a different way than the noisy input data. The Encoder is trained separately before the VoiceFilter network. All this is captured in Figure 7
The data is generated by two freely available data sets, LibriSpeech and VCTK.

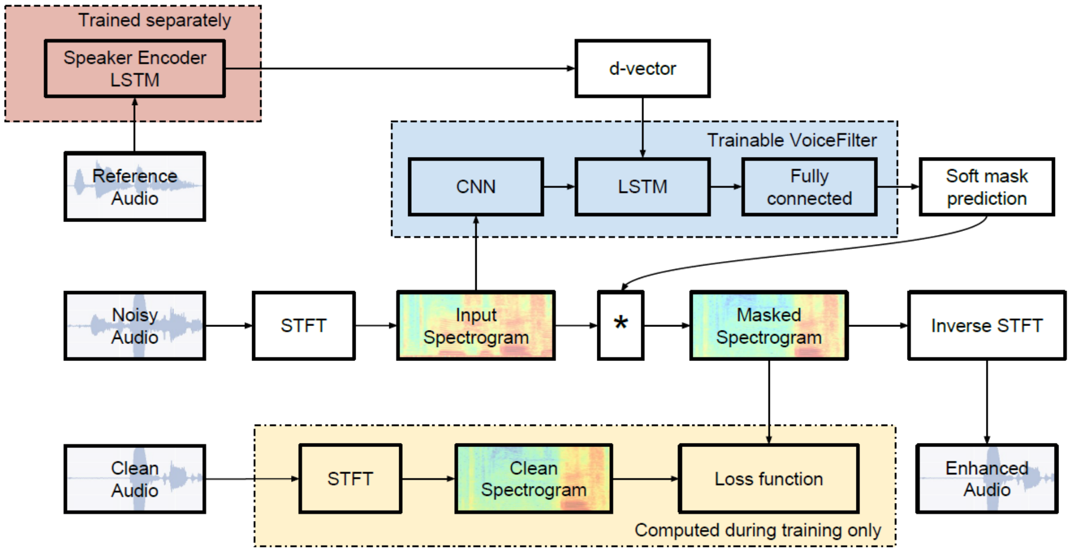


Figure 7: The entire workflow of the VoiceFilter network

From them we create a triplet of training data: reference, clean and noisy audio. The noisy or mixture audio exactly contains two speakers.

Results of [5]

To understand the results we first explain our objectives and the evaluation measures.
Since the motivation of speaker extraction is improving speech recognition we aim for a network that improves performance in noisy situations and doesn't make performance on already well separated data worse. For that reason we care about the word error rate (WER), which captures the ratio of correctly recognized words. We compare clean and noisy WER before and after applying VoiceFilter.
Furthermore, we evaluate the source distortion ratio (SDR) which is a typical measure for source separation that describes the ratio between the energy of the target signal and the energy of the errors.
The results can be seen in 8.
In Table 2 we see significant improvements of the noisy WER while the clean WER impairs only by a bit. So the model (especially in the case of bi-LSTM)

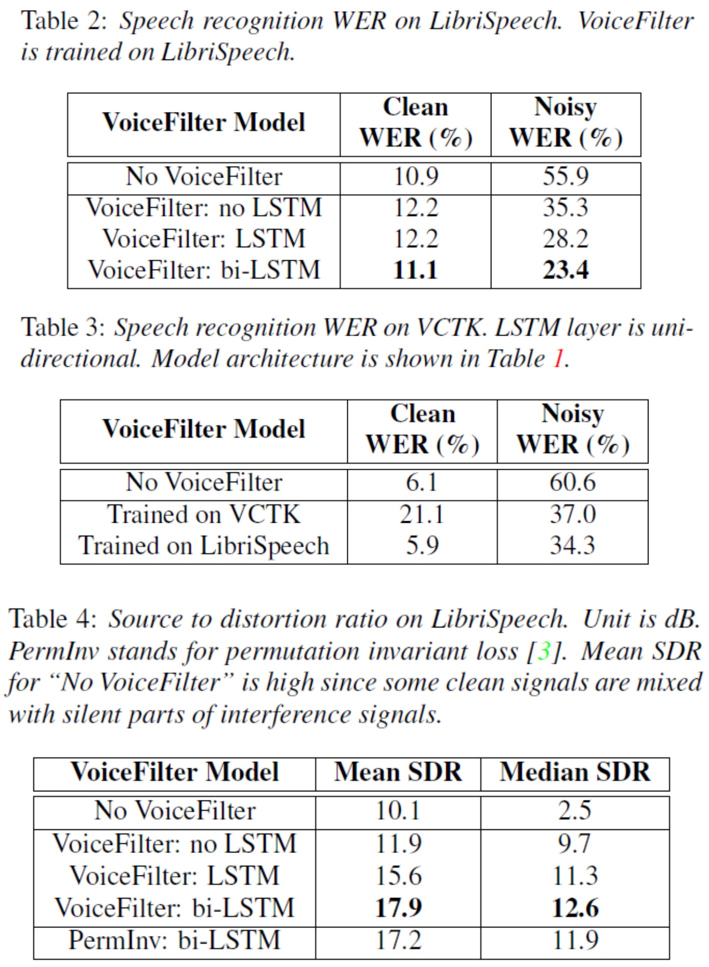


Table 2: *Speech recognition WER on LibriSpeech. VoiceFilter is trained on LibriSpeech.*

VoiceFilter Model	**Clean WER (%)**	**Noisy WER (%)**
No VoiceFilter	10.9	55.9
VoiceFilter: no LSTM	12.2	35.3
VoiceFilter: LSTM	12.2	28.2
VoiceFilter: bi-LSTM	**11.1**	**23.4**

Table 3: *Speech recognition WER on VCTK. LSTM layer is uni-directional. Model architecture is shown in Table 1.*

VoiceFilter Model	**Clean WER (%)**	**Noisy WER (%)**
No VoiceFilter	6.1	60.6
Trained on VCTK	21.1	37.0
Trained on LibriSpeech	5.9	34.3

Table 4: *Source to distortion ratio on LibriSpeech. Unit is dB. PermInv stands for permutation invariant loss [3]. Mean SDR for "No VoiceFilter" is high since some clean signals are mixed with silent parts of interference signals.*

VoiceFilter Model	**Mean SDR**	**Median SDR**
No VoiceFilter	10.1	2.5
VoiceFilter: no LSTM	11.9	9.7
VoiceFilter: LSTM	15.6	11.3
VoiceFilter: bi-LSTM	**17.9**	**12.6**
PermInv: bi-LSTM	17.2	11.9

Figure 8: Results of the VoiceFilter network

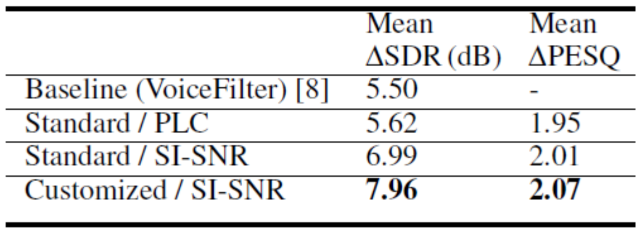
does what is expected. Table 3 shows two things. First, the VCTK dataset is to small for achieving a decent noisy WER. When trained on LibriSpeech but evaluated on VCTM, we improve again. So the model generalizes between datasets. Second, the clean WER even improves when trained on LibriSpeech and evaluated on VCTK. So, even if it was not told to do that explicitly, the network improves on the audibility of the data. In Table 4 we further see an indication for the capability of the system to extract the signal of interest. Here we compare the SDR in dB and observe that the network improves on that quantity as well.

5 Modifications made by [4]

There are two main differences to the baseline approach here: a customization of the LSTM cells and the additional use of another evaluation measure.
The LSTM cell is modified in the simple way that we now set the weights that decide the contribution of the convolved spectrum going into the forget gate of the LSTM cells, is set to zero. It can be intuitively understood in the way that the LSTM cell is supposed to learn to retain information related to the target speaker and disregarding information unrelated to the target speaker.
Furthermore, in [4] they do not evaluate WER, but perceptual evaluation of speech quality (PESQ). This is a standard measure for how well humans might understand an audio signal. Simple metrics like the SNR, have proven to be ineffective at predicting user experience. One reason for this is that SNR for example does not differentiate between audible and inaudible distortions. PESQ is a algorithmic estimation of the mean opinion score (MOS) which is a value determined by humans. 1 (worst) means 'no communication possible' and 5 (best) means 'easily understandable'
Also they change the number of neurons in the LSTM and FCNN. Details can be seen in Figure 6.

Results of [4]

The results are depicted in Figure 9. One can clearly see the slight improvement of the modifications made. In table there also appear the names PLC and SI-SNR which are the names of the used loss functions. This will not be explained in more detail.



	Mean ΔSDR (dB)	Mean ΔPESQ
Baseline (VoiceFilter) [8]	5.50	-
Standard / PLC	5.62	1.95
Standard / SI-SNR	6.99	2.01
Customized / SI-SNR	**7.96**	**2.07**

Figure 9: Results of the modified VoiceFilter using custom LSMT cells

6 Conclusion

The proposed modifications of the VoiceFilter baseline approach show some slight improvements so it suggests that their attempt to improve the hyperparameters was a success. Unfortunately they didn't compare the WER so it is not clear if this will bring improvements for practical applications.

References

[1] Shervine Amidi. Teaching. `https://stanford.edu/~shervine/`. [Online; accessed 28-December-2021].

[2] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. `http://www.deeplearningbook.org`.

[3] Paul-Louis Pröve. An Introduction to different Types of Convolutions in Deep Learning. `https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d`. [Online; accessed 28-December-2021].

[4] Ragini Sinha, Marvin Tammen, Christian Rollwage, and Simon Doclo. Speaker-conditioned target speaker extraction based on customized lstm cells, 2021.

[5] Quan Wang, Hannah Muckenhirn, Kevin Wilson, Prashant Sridhar, Zelin Wu, John Hershey, Rif A. Saurous, Ron J. Weiss, Ye Jia, and Ignacio Lopez Moreno. Voicefilter: Targeted voice separation by speaker-conditioned spectrogram masking, 2019.