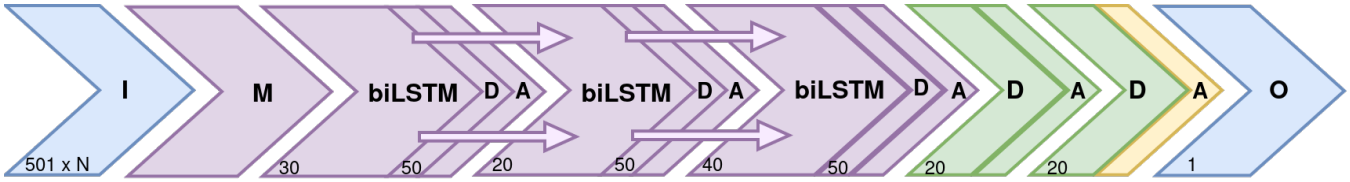
Speaker count estimation based on an artificially generated data set

Aucke Bos

Radboud University - Automatic Speech Recognition (LET-REMA-LCEX10)

s4591496

Figure 1. The structure of our RNN. I, M, biLSTM, D, A, O are Input, Masking, Bi-directional LSTM, Dropout, Activation, and Output layers respecitvely. Output sizes are provided. All but the last LSTMs return sequences instead of single outputs, they all use Tanh activation functions. The first Dense layer uses Relu activation, while the second uses Exponential activation, as this combines well with the Poisson loss.



Abstract

Speaker count estimation is the task of counting the maximum number of active speakers within a time frame. Although it remains an open problem[3][17][18], recent research made much progress using deep learning[11][18]. In some cases, deep learning approaches even outperform humans[2]. In this paper, and its corresponding source code[1], we implement and describe an RNN for speaker count estimation. The network is trained with a synthetically created dataset, based on single-speaker recordings of the TIMIT dataset[10]. We evaluate performance on a dataset with multi-speaker recordings[19].

1 Introduction

Solving the problem of speaker count estimation can have a high impact on the performance of other speech-related tasks. It can improve ASR performance by enabling speaker adaptation[8]. It's also an important starting point in the task of speaker diarisation: deciding who is speaking when. According to Stöter et al[17], many ASR applications build on the assumption that the number of speakers in an audio fragment is known. Thus for these systems to have a decent performance, we must have a well-performing speaker count estimation algorithm. As in many machine-learning fields, deep learning recently started to outperform the highest performing algorithms from the past. Pan et al[15] show that DNNs consistently outperform GMMs.

The topic of this experiment is thus related to solving the speaker count estimation problem. To solve such a problem, we need a dataset. We chose to use a synthetically created dataset, for two reasons:

- Many datasets with audio recordings can be found online, but few are focused on the problem of speaker count estimation. Those sets that are, often lack accurate annotations or contain recordings where only minor parts of the recordings have overlapping speakers[18].
- We want to investigate whether it is possible to train a NN on a dataset initially containing single-speaker recordings, and obtain reasonable accuracy on datasets of multi-speaker recordings. To this end, we found a dataset based on Librispeech[14] that focused on speaker count estimation: Libricount[19]. We use this dataset for testing only. We use the TIMIT dataset[10] to create and build our training and validation sets.

In line with this, our research question states:

Can we train a Neural Network to estimate the number of speakers in a multi-speaker recording, by artificially creating multi-speaker sources?

2 Related work

Earlier research uses approaches based on Gaussian Mixture Models[4] or i-vectors[7]. As in many machine learning problems nowadays, much recent research tries to solve this task using deep learning: Convolutional Neural Networks (CNN)[8], Recurrent Neural Networks (RNN)[18], or a combination (CRNN)[11]. Other research often uses Hidden

[1] https://github.com/AuckeBos/Speaker-count-estimation-based-on-an-artificially-generated-data-set

Markov Models in tasks related to ASR[12]. It has been found that neural networks can also be combined with HMMs[6][1].

Stöter et al investigated different deep learning approaches to solve this task[18]. The research tests different network structures, different problem formulations, and different input representations. They also investigated different ways to interpret the outputs of the network[17].

We use the paper of Stöter et al[18] as a basis for our experiments. Whereas they compare several network structures and types, several problem formulations, and several input representations, we restrict ourselves to one network type, one problem formulation, and several input representations. These design choices are discussed extensively in Section 4. In addition to Stöter et al, we investigate the ability of our network to generalize to another dataset. How we selected the datasets, and how well our network generalizes, are elaborated upon in sections 4.4 and 5.

3 Method & Set-up

To limit the scope of the experiment, we limit ourselves to testing a single network structure with a single problem formulation. In this section, we elaborate upon the chosen structure and problem formulation. In the next section (4), we discuss the experiments we have performed on this configuration.

3.1 Network structure

Stöter et al investigated six network structures, containing different CNNs, an RNN, and a CRNN. For this experiment, we have chosen to implement the RNN they proposed. They showed this network performs really well on our problem, and it is fairly easy to implement and fast to train. Moreover, a recurrent approach seems like a natural choice for our task, since estimating concurrent speakers in a time frame requires information about surroundings, both in the past and the future. The structure of our network is shown in Figure 1.

Our network deviates a little from the network proposed by Stöter et al since we omit the Max Pooling layer. Early tests showed that this does not only slow down training but also decreases performance. Instead we set `return_sequences` to `False` in the last LSTM layer, and append a dense layer. We also use a Keras Masking layer to handle variable-length input (see Section 4.4.1). The Masking layer recognizes the padded values for recordings shorter than five seconds and excludes these from the input before passing it through to the LSTM layers.

3.2 Problem formulation

Our task is to determine the maximum number of concurrent speakers in a recording. We restricted our dataset to contain recordings with up to 20 concurrent speakers, but we want our network to be able to determine any number of speakers. Therefore we can not use classification with softmax, since this would require the model structure to change when we would want to change the upper boundary[18]. This is because such an approach defines the upper bound by the size of the last dense layer: A dense layer of output size 20 can define 20 different outputs. To estimate more than 20 speakers, the final layer needs to be changed.

Modeling a Poisson distribution is more suitable for this type of problem. Poisson distributions are used often in machine learning problems where the task is to predict counts or other discrete variables[9][5]. A big advantage with respect to normal classification is that the Poisson distribution keeps the dependencies between predictions: A count of 4 is closer to a count of 5 than to a count of 10. In classification, each value is given a probability, and these probabilities are not directly related. Another option would be Gaussian regression. But since a Gaussian distribution estimates continuous variables, we would need to round the predictions eventually. The Poisson distribution is a discrete one and thus doesn't have this problem. Our network is trained to learn a single parameter λ, which fully describes the Poisson distribution. The most straightforward loss function is to use the Poisson negative log-likelihood, thus we chose that loss function in our experiments. During inference, we pick for our count estimation the median of the Poisson distribution described by λ. Another option would be to pick the mode as prediction instead, however, Stöter et al found that taking the median results in more accurate predictions.

Hereby we conclude our method: We train a recurrent neural network with biLSTMs to predict the scaling parameter λ for a Poisson distribution. Intuitively we use the negative Poisson log-likelihood for the loss function and infer predictions by taking the median of the inferred Poisson distribution.

4 Experiments

The experiments we perform are threefold. Firstly we test five different input representations, they are discussed below. Secondly, we test the generalization performance of our network, by comparing our test score to the score of the test set of LibriCount[2]. Lastly, we compare the performance of the network when trained on two different datasets. The first set contains recordings with up to 10 concurrent speakers, while the second contains recordings with up to 20 concurrent speakers. This comparison is performed solely on our own

[2] https://zenodo.org/record/1216072

dataset since LibriCount only has recordings with at most 10 concurrent speakers.

4.1 Input representations

The most used feature representation in the field of ASR is Mel Frequency Cepstral Coefficients (MFCC)[21]. Other research uses Short Time Fourier Transforms (STFTs) or applies MEL filters over these STFTs to transform them into MEL basis. Therefore we decided to compare performance for these representations. For all representations, we use a frame length of 25ms, with a 10 ms overlap. Since our inputs are all 5 seconds long, this gives us 501 features in time domain.

1. `STFT`: Use the Short Time Fourier Transformations of our data, using Hann windowing. The size in feature domain is 201.
2. `LOG STFT`. Scale the STFT logarithmically. The size in feature domain is 201.
3. `MEL 20`. Convert the STFT into MEL basis, using 20 triangular filters. The size in feature domain is 20.
4. `MEL 40`. Convert the STFT into MEL basis, using 40 triangular filters. The reason we added this representation, is for a fairer comparison with STFT. If MEL performs worse, we'd like to investigate to what extent this is due to fact that we have far fewer features (20 vs 201). If we take 40 features instead, does this improve performance with respect to `MEL 20`, or only slow down training? The size in feature domain is 40.
5. `MFCC 2-13`. MFCC is used very often in literature. Stöter et al excluded this feature[18] representation from their comparison since it performs badly on CNNs[16]. Since we do not use CNN, but only an RNN, we do include this representation in our comparison. Other research tells us to use only 12-20 coefficients. We want our network to be robust to signal power, thus we exclude the very first coefficient. To reduce complexity we only take 12 coefficients, thus we take coefficients 2-13. The size in feature domain is 12.

4.2 Configurations

We thus have five different input representations we'd like to evaluate. We'd also like to measure the influence of the maximum number of speakers in the training set on performance on the test set. Therefore we create two training sets (discussed in Section 4.4.1): one containing recordings of 1 to 10 concurrent speakers, and one containing recordings of 1 to 20 concurrent speakers. For both sets, we train the network 5 times: once for each feature representation. This gives us 10 training configurations. Each configuration is trained on the same network (Section 3.1), with the following network configuration:

- Train for at most 80 epochs.
- Use a batch size of 128.
- Use the Keras Poisson loss[3] function to minimize.
- Optimize using the Adam optimizer, add Mean Absolute Error as metric.
- Use the following callbacks:
  - Tensorboard, for easy monitoring.
  - EarlyStopping with a patience of 15, measuring validation loss.
  - ReduceLROnPlateau with a patience of 7, measuring validation loss.
  - A custom callback that keeps track of the average time of one epoch.
  - ModelCheckpoint to save the best performing model after training has finished, measuring validation loss.
- We train on the Thunderlane server of the Ponyland server cluster for CLS[4]. This server has 3 NVIDIA Tesla T4's and 250GB of ram. We run Keras using CuDA.

4.3 Data augmentation

Since one of our goals is to make a network that generalizes well, we applied several preprocessing techniques:

1. By means of *volume randomization*, we randomly select a loudness value between 50 and 70 dB for each file and normalize the file to this loudness level. To compute the initial loudness of an input file s, we use Equation 1, where $\|s\|$ is the length of the signal.

$$10 \cdot \log_{10} \frac{\sum s^2}{\|s\| \cdot 4 \cdot 10^{10}} \tag{1}$$

   We thus seek a multiplier m for signal s such that the new loudness level is the randomly selected value $l_2 \in [50, 70]$. After computing the original loudness l_1 and rewriting Equation 1, we multiply s with m to convert it to the new loudness level l_2, where m is given by

$$10^{\frac{l_2 - l_1}{20}}$$

   We apply loudness randomization during the training and testing phase. We hereby ensure that all the data we train and test on is within the desired dB range.
2. With a probability of 20%, we nosify the input data, during training only. We generate noise data $n \in [0, 1]$, and update our input to s be $s + 0.01 \cdot n$.
3. With a probability of 25%, we shift our input data by 1 second to the left or right, during training only.

[3] https://www.tensorflow.org/api_docs/python/tf/keras/losses/poisson
[4] https://ponyland.science.ru.nl/

4. With a probability of 35%, we change the pitch of our sound using the `pitch_shift` of librosa, with a factor $f \in [1, 5]$.
5. With a probability of 40%, we time-stretch our sound using `time_stretch` of librosa, with a factor $f \in [0.75, 1.25]$.

4.4 Dataset preparation

We denote three custom generated sets: C_{tr} is our custom training set, C_{va} is our custom validation set, C_{te} is our custom test set. All sets are generated differently; for each of them the process is stated below. As explained, we generate these sets out of single-speaker recordings of TIMIT. TIMIT is divided into a train set (T_{tr}) containing 4620 recordings, and a test set (T_{te}) with 1680 recordings. C_{te} is generated out of T_{te}, C_{tr} and C_{va} are generated out of T_{tr}.

4.4.1 The training set. Our training set C_{tr} is generated on the fly using a Keras DataGenerator. Before training, we randomly select 80% of T_{tr}, these are used as input l. The parameters for the generator are:

- The list of single-speaker files l.
- The batch size b.
- The feature type f (see Section 4.1).
- The minimum number of speakers per generated file c_{min}.
- The maximum number of speakers per generated file c_{max}.
- The number of times a single-speaker file is used to merge into multi-speaker files n.

The generator generates a shuffled balanced set of labels Y, where each label is $\in [c_{min}, c_{max}]$, such that $sum(Y) = len(l) \cdot n$. This means that on average, each file w in l is used n times. We allow duplications, since the files will be merged at random each time, thus the chance of exact duplicates of merged files is extremely low. Since Y is balanced, the average number of speakers per generated file is $a = \frac{c_{max} - c_{min}}{c_{max}}$. Thus if we wouldn't allow duplicates, and only use each w once, we would only have a set of size $\frac{len(l)}{a}$. This size is now increased to $n \cdot \frac{len(l)}{a}$.

When a batch is retrieved, the generator retrieves b labels of Y and creates a list of merged files X, where each $x \in X$ is a merge of $y \in Y$ wav files. We merge by summing the values of each file. The length of the merge file is equal to the length of the longest original file; shorter files are zero-padded. After each epoch, we shuffle l and Y.

Each $x \in X$ is preprocessed:

- Randomize the loudness to have a value $\in [50, 70]$.
- Augment x, see Section 4.3.
- Pad and cut off x at 5 seconds. The padded values will be masked out in the network (Section 3.1).
- Normalize x to $[0, 1]$.
- Extract the features of x as defined by f.

4.4.2 Validation set. C_{va} is generated using the same generator as C_{tr}, with the following modifications:

- We do not augment x. We want to measure performance on un-augmented data, as we won't augment our test set either.
- We do not generate a new X and Y on every epoch. Instead, we generate them once before each configuration and use the same set after each epoch. We do this such that we evaluate on the same validation set each epoch, hence we have a more consistent view of the performance increase on the validation set during training. This also makes the used callbacks (Section 4.2) more effective.

4.4.3 Test set. We generate a test set only once, and use the same set for each configuration. Therefor we use a separate `DataLoader` for the test set. The dataloader consumes the list of files l in T_{te}, and parameters c_{min}, c_{max}. The test is balanced in Y, and creates $n = \frac{len(l)}{c_{max}}$ merged files per $c \in [c_{min}, c_{max}]$. So for each c, we make n merged files out of c randomly selected files (without replacement) of T_{te}. The files are stored in the file system, such that we can easily re-use them along different configurations.

5 Results

We have defined three goals in Section 4. We perform experiments to answer these questions and elaborate on the results, supported by tables and figures. The experiments are run on two different training sets, we call them C_{tr10}, C_{tr20}. The sets are generated from the same source set T_{tr}, as explained in Section 4.4.1. The first set contains files generated by merging up to at most 10 files, the second set has merged up to at most 20 files. Since we use the same set T_{tr} using the same DataLoader, our first set naturally contains more data points. This is reflected in the average training times in Table 1.

5.1 Best performing feature representation

We have evaluated five different feature representations. For each of these representations, we have trained our network twice: once for each set. During training, we keep track of performance using callbacks. The lowest losses, MAE's and LR's are shown in Table 1. We also provide the average time per epoch, and the number of epochs before early stopping was applied. For each result, we show them for both training sets. The lowest value of each column is printed in **bold**.

Table 1. Performance comparison for all features types for two data sets. The minimum value for each column is printed in **bold**. LOG_STFT performs best on a dataset with at most 10 concurrent speakers; MFCC performs best on a dataset with at most 20 concurrent speakers.

	MAE C_{tr}		MAE C_{va}		Loss$_{tr}$		Loss$_{va}$		LR		s / Epoch[5]		#Epochs	
Dataset	C_{tr10}	C_{tr20}	C_{tr10}	C_{tr20}	C_{tr10}	C_{tr20}	C_{tr10}	C_{tr20}	C_{tr10}	C_{tr20}	C_{tr10}	C_{tr20}	C_{tr10}	C_{tr20}
STFT	1.361	2.925	1.753	3.454	-4.429	**-15.294**	-4.099	-14.771	1.600e-4	1.600e-4	**160.030**	**91.462**	**22**	**33**
LOG_STFT	**1.345**	3.347	**0.942**	2.844	**-4.441**	-15.124	**-4.383**	-15.053	6.400e-5	1.600e-4	160.847	91.618	61	47
MEL20	2.055	3.642	1.839	3.611	-4.092	-14.938	-4.046	-14.634	1.600e-4	1.600e-4	163.490	92.702	33	80
MEL40	1.843	4.080	1.557	3.539	-4.218	-14.716	-4.211	-14.720	**2.560e-5**	**6.400e-5**	163.897	93.786	55	56
MFCC	1.732	**2.900**	1.564	**2.433**	-4.264	-15.285	-4.152	**-15.197**	6.400e-5	1.600e-4	176.045	105.286	50	69

The first thing we notice is a drastic decrease in performance in C_{tr20} with respect to C_{tr10}. We did expect a decrease in performance, but not so drastically. The second noteworthy observation is that two different feature representations show to be effective on the different datasets. For C_{tr10}, LOG_STFT shows the best results whereas for C_{tr20} this is MFCC. We also see that the loss is not always in line with the MAE: The MAE$_{va}$ for LOG_STFT is much lower than we would expect when looking at LOSS$_{va}$. Another observation is that the number of features does not significantly reduce training time: s / Epoch of MEL40 and MEL20 are comparable. Lastly, we note that STFT converges much faster than the other feature types. Since s / Epoch is of the same magnitude as the rest, this means STFT trains a lot faster. This is surprising, especially with respect to LOG_STFT, which is a very similar feature representation.

We conclude our observations regarding this topic by stating that LOG_STFT performs best when trained on C_{tr10}, and MFCC is the best feature representation when training on C_{tr20}. Therefore we will use these two feature representations for the analyses in the rest of this section.

5.2 Generalization performance on a new test set

The second question we address is the generalization performance of our network to a new dataset. This new dataset is LibriCount, we will call it L_{te10}. It is an evenly distributed set with 5720 records with labels $\in [0, 10]$. Since we have trained on sets with labels $\in [1, 20]$, we drop the datapoints with label 0. We now have $|L_{te10}| = 5200$. We generate C_{te10} from the source set T_{te} as described in Section 4.4.3. This set is evenly distributed as well, with $|C_{te10}| = 1690$. We test the network with feature representation LOG_STFT and the network with feature representation MFCC on both the test sets. We plot the MAE for each label in Figures 2 and 3.

The first thing we notice is that the networks trained on C_{tr20} score significantly worse than those trained on C_{tr10}, however, in this section, we focus on the difference in performance on the test sets; this behavior will be discussed in the next section. As we expected, we see that performance

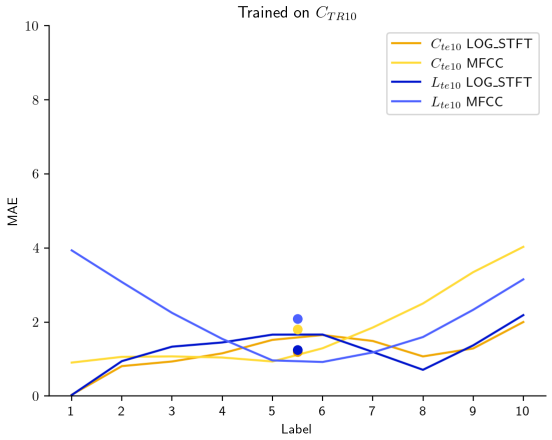


Figure 2. MAE per label on test set C_{te10} and L_{te10} for two feature types trained on C_{tr10}. The average MAE are also shown.

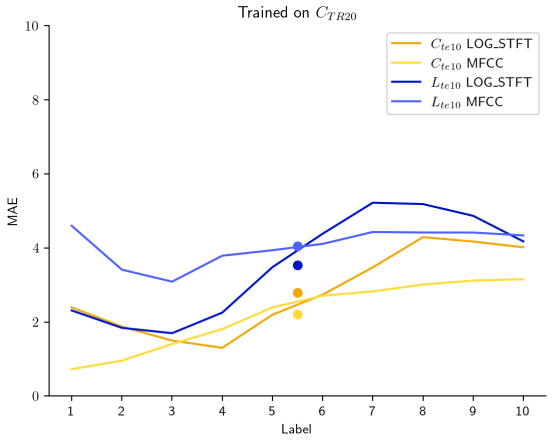


Figure 3. MAE per label on test set C_{te10} and L_{te10} for two feature types trained on C_{tr20}. The average MAE are also shown.

on L_{10} is worse than performance on C_{te10}, however, the scores are quite close. Thus we see that although the scores are in general not remarkably good, they are quite alike over different test sets. The shape of the scores of MFCC trained on C_{tr10} is also interesting. We see that performance decreases up until $y = 5$, after which it increases. Careful investigation in the classified labels tells us that the network classifies the input data close to 5 in many cases. Thus as the label increases, the error decreases up until label = 5, after which the error increases again. Unfortunately, we were not able to pinpoint what causes this behavior, nor were we able to fix it. This pattern does not seem to occur when trained on C_{tr20}. In general, we see that performance decreases as the number of concurrent speakers increases, which is in line with our expectations: differentiating more people is harder than only a few. The last observation we make is visible in Figure 2. It looks like the error linearly increases for all features with $y > 8$. It looks like all configurations classify to at most 8, after which the error increases by one for each new speaker. We take a look at this statement for $y > 10$ in the next section.

5.3 Generalization performance on new labels

The last topic to be discussed is the generalization performance to a dataset with new labels. The generalization of a neural network to a new dataset is not trivial[20][13]. Therefore we do not expect our network to show good performance on new labels, however, we do include an analysis here, since it's an interesting topic. For this analysis, we have created yet another test set called C_{te20}, generated in the same fashion as C_{te10}, with $|C_{te20}| = 1700$ and labels $\in [1, 20]$. We test the network with feature representation LOG_STFT and the network with feature representation MFCC on test sets C_{te10} and C_{te20}, and plot the MAE for each label in Figures 4 and 5.

The second figure confirms our statement of the previous section for C_{tr10}: For $y > 8$, the error linearly increases. Manually checking the predictions indeed shows us that the network does not predict any input to a label > 8. Unfortunately, we could not find out why this happens. We see that for more concurrent speakers, the networks trained on C_{tr20} perform much better. The curve of Figure 5 seems quite weird, as we would expect it to keep increasing with the label value. But we see a concave structure up until $y = 16$. However, the average MAE over all labels is quite good: about 3. This average MAE is lower than we had expected, but the generalization when trained on C_{tr10} is worse than we expected: there is no generalization skill at all, for either feature type. The last observation to mention is that MFCC performs better when trained on C_{tr20}, while it is outperformed by LOG_STFT when trained on C_{tr10}. This behavior is in line with the observation we made in Section 5.1 when looking at Table 1.

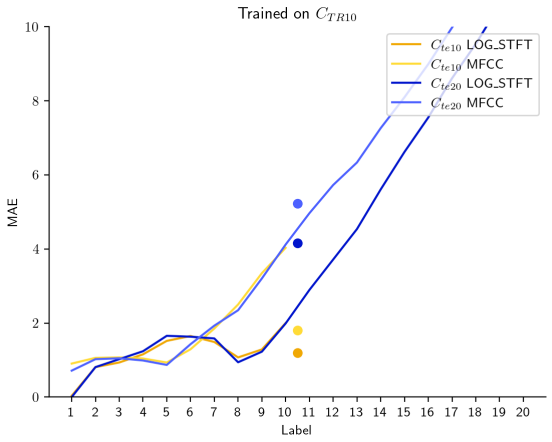


Figure 4. MAE per label on test set C_{te10} and C_{te20} for two feature types trained on C_{tr10}. The average MAE are also shown, over classes 1-10 and 1-20 for C_{te10}, C_{te20} respectively.

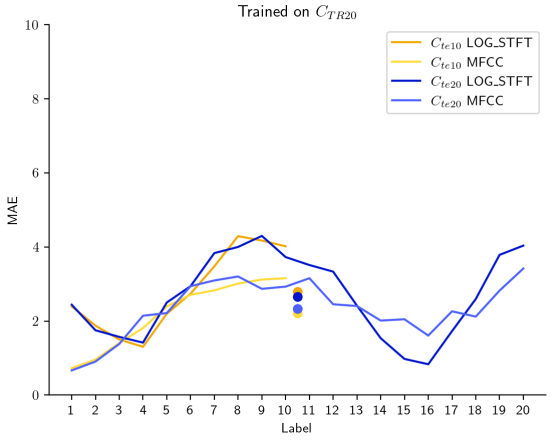


Figure 5. MAE per label on test set C_{te10} and C_{te20} for two feature types trained on C_{tr20}. The average MAE are also shown, over classes 1-10 and 1-20 for C_{te10}, C_{te20} respectively.

6 Discussion

We have successfully investigated the research questions proposed in Section 4. Regarding different feature representations, we see clear differences for the different training sets. We had expected MFCC to be the best representation overall since it's so widely used in literature and practice. It happened to score best when trained on C_{tr20}, but not as much as we expected. Overall this experiment shows us that our system works quite well,

[5]Adjusted for outliers due to waiting time on the server.

although definitely not perfect when trained on C_{tr10}, and it's not very good when trained on C_{tr20}.

The generalization to a new test set showed better performance than expected. In general, neural networks are not very capable of generalization to new sets, but our decrease in performance on a new set was very limited. This might be due to the similarities in how the sets are generated. The generation of LibriCount is elaborated upon in Section 4.1 of [18]; the generation of our own training set was discussed in Section 4.4.1. Regardless of the creation procedure, we were positively surprised by the results.

In line with our expectations, generalization to new labels turned out to be too difficult. When our network was trained on C_{tr10}, it was only able to predict labels of at most 10. For recordings with a label > 10, the error increases linearly with the number of concurrent speakers. However, when we trained our network on labels $\in [1, 20]$, the score was quite decent for more concurrent speakers as well. This was a little surprising because the results of Stöter et al decrease drastically with the true speaker count. This strong decrease in performance is less visible in Figure 5, although the overall trend looks very different too, so the results might not be very comparable.

7 Conclusion

We have experimented with a Recurrent Neural Network trained on an artificially created dataset for speaker count estimation. Our goal was to verify whether such an artificially created dataset would suffice to build a network that can predict the maximum number of concurrent speakers in a multi-speaker audio file with decent accuracy.

For this task, we have defined three subtopics. Firstly, we have investigated five different feature representations and concluded that 2 of those score best on two different training sets. The second task consisted of an experiment where we actually tested the network on a new, unseen dataset. The generalization performance turned out better than expected. Lastly, we tested whether the network was also able to generalize to a new set that contains new labels; containing recordings with more concurrent speakers. It turned out that since the network was not trained for this, it wasn't able to predict the number of concurrent speakers well.

In conclusion, we are somewhat happy with our results. Our system shows that it understands the task, but its performance is not good enough to be usable in practice. As for future work, we might apply more sophisticated data augmentation, and experiment with different network structures. Looking at the paper of Stöter et al, we should be able to achieve higher performance than we have shown here. Nevertheless, the project was very educational and fun, and it has definitely risen our interest in the topic of automatic speech recognition.

References

[1] A. Adjoudani and C. Benoît. "On the Integration of Auditory and Visual Parameters in an HMM-based ASR". en. In: *Speechreading by Humans and Machines: Models, Systems, and Applications*. Ed. by David G. Stork and Marcus E. Hennecke. NATO ASI Series. Berlin, Heidelberg: Springer, 1996, pp. 461–471. ISBN: 978-3-662-13015-5. DOI: 10.1007/978-3-662-13015-5_35. URL: https://doi.org/10.1007/978-3-662-13015-5_35 (visited on 03/13/2021).

[2] V. Andrei et al. "Counting competing speakers in a timeframe - human versus computer". In: *INTERSPEECH*. 2015.

[3] Valentin Andrei et al. "Estimating competing speaker count for blind speech source separation". In: *2015 International Conference on Speech Technology and Human-Computer Dialogue (SpeD)*. Bucharest, Romania: IEEE, Oct. 2015, pp. 1–8. ISBN: 978-1-4673-7560-3. DOI: 10.1109/SPED.2015.7343081. URL: http://ieeexplore.ieee.org/document/7343081/ (visited on 03/13/2021).

[4] Xavier Anguera Miro et al. "Speaker Diarization: A Review of Recent Research". In: *IEEE Transactions on Audio, Speech, and Language Processing* 20.2 (Feb. 2012), pp. 356–370. ISSN: 1558-7916, 1558-7924. DOI: 10.1109/TASL.2011.2125954. URL: http://ieeexplore.ieee.org/document/6135543/ (visited on 03/13/2021).

[5] A. B. Chan and N. Vasconcelos. "Counting People With Low-Level Features and Bayesian Regression". In: *IEEE Transactions on Image Processing* 21.4 (Apr. 2012), pp. 2160–2177. ISSN: 1057-7149, 1941-0042. DOI: 10.1109/TIP.2011.2172800. URL: http://ieeexplore.ieee.org/document/6054049/ (visited on 04/08/2021).

[6] Ichael Cohen et al. "Combining Neural Networks And Hidden Markov Models For Continuous Speech Recognition". In: (June 1999).

[7] Najim Dehak et al. "Support vector machines versus fast scoring in the low-dimensional total variability space for speaker verification". In: vol. 1. Jan. 2009, pp. 1559–1562.

[8] *ESTIMATION OF SPEAKER COUNT USING FEATURES LEARNT FROM SUPERVISED AND UNSUPERVISED DEEP LEARNING METHODS - PDF Free Download*. URL: https://docplayer.net/200781392-Estimation-of-speaker-count-using-features-learnt-from-supervised-and-unsupervised-deep-learning-methods.html (visited on 03/13/2021).

[9] Nader Fallah et al. "Nonlinear Poisson regression using neural networks: a simulation study". en. In: *Neural Computing and Applications* 18.8 (Nov. 2009), pp. 939–943. ISSN: 0941-0643, 1433-3058. DOI: 10.1007/s00521-009-0277-8. URL: http://link.springer.com/10.1007/s00521-009-0277-8 (visited on 04/08/2021).

[10] John S. Garofolo et al. *TIMIT Acoustic-Phonetic Continuous Speech Corpus*. type: dataset. DOI: 10.35111/17GK-BN40. URL: https://catalog.ldc.upenn.edu/LDC93S1 (visited on 03/13/2021).

[11] P.-A. Grumiaux et al. "High-Resolution Speaker Counting in Reverberant Rooms Using CRNN with Ambisonics Features". In: *2020 28th European Signal Processing Conference (EUSIPCO)*. ISSN: 2076-1465. Jan. 2021, pp. 71–75. DOI: 10.23919/Eusipco47968.2020.9287637.

[12] H. Hadian et al. "Flat-Start Single-Stage Discriminatively Trained HMM-Based Models for ASR". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 26.11 (Nov. 2018). Conference Name: IEEE/ACM Transactions on Audio, Speech, and Language Processing, pp. 1949–1961. ISSN: 2329-9304. DOI: 10.1109/TASLP.2018.2848701.

[13] Steve Lawrence, C. Lee Giles, and Ah Chung Tsoi. "What Size Neural Network Gives Optimal Generalization? Convergence Properties of Backpropagation". en_US. In: (Oct. 1998). Accepted: 2004-05-31T22:38:33Z. URL: https://drum.lib.umd.edu/handle/1903/809 (visited on 05/07/2021).

[14] *Librispeech: An ASR corpus based on public domain audio books | IEEE Conference Publication | IEEE Xplore*. URL: https://ieeexplore.ieee.org/document/7178964 (visited on 03/17/2021).

[15] J. Pan et al. "Investigation of deep neural networks (DNN) for large vocabulary continuous speech recognition: Why DNN surpasses GMMS in acoustic modeling". In: *2012 8th International Symposium on Chinese Spoken Language Processing*. Dec. 2012, pp. 301–305. DOI: 10.1109/ISCSLP.2012.6423452.

[16] Michael L. Seltzer, Dong Yu, and Yongqiang Wang. "An investigation of deep neural networks for noise robust speech recognition". In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. Vancouver, BC, Canada: IEEE, May 2013, pp. 7398–7402. ISBN: 978-1-4799-0356-6. DOI: 10.1109/ICASSP.2013.6639100. URL: http://ieeexplore.ieee.org/document/6639100/ (visited on 04/09/2021).

[17] Fabian-Robert Stoter et al. "Classification vs. Regression in Supervised Learning for Single Channel Speaker Count Estimation". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Calgary, AB: IEEE, Apr. 2018, pp. 436–440. ISBN: 978-1-5386-4658-8. DOI: 10.1109/ICASSP.2018.8462159. URL: https://ieeexplore.ieee.org/document/8462159/ (visited on 03/11/2021).

[18] Fabian-Robert Stoter et al. "CountNet: Estimating the Number of Concurrent Speakers Using Supervised Learning". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 27.2 (Feb. 2019), pp. 268–282. ISSN: 2329-9290, 2329-9304. DOI: 10.1109/TASLP.2018.2877892. URL: https://ieeexplore.ieee.org/document/8506601/ (visited on 03/11/2021).

[19] Fabian-Robert Stöter et al. *LibriCount, a dataset for speaker count estimation*. type: dataset. Apr. 2018. DOI: 10.5281/zenodo.1216072. URL: https://zenodo.org/record/1216072#.YFHwef4o_Xk (visited on 03/17/2021).

[20] Aarne Talman and Stergios Chatzikyriakidis. "Testing the Generalization Power of Neural Network Models Across NLI Benchmarks". In: *arXiv:1810.09774 [cs]* (May 2019). arXiv: 1810.09774. URL: http://arxiv.org/abs/1810.09774 (visited on 05/07/2021).

[21] Hojatollah Yeganeh, Seyed Mohammad Ahadi, and Ali Ziaei. "A new MFCC improvement method for robust ASR". In: *2008 9th International Conference on Signal Processing*. Beijing, China: IEEE, Oct. 2008, pp. 643–646. ISBN: 978-1-4244-2178-7. DOI: 10.1109/ICOSP.2008.4697213. URL: http://ieeexplore.ieee.org/document/4697213/ (visited on 04/09/2021).