Evaluating different network architecures for speech recognition task

Max Henning Höth[†]

Abstract—**Recognition the spoken language is a key field of AI. It enables the computer to recognise the human speech. The goal of speech recognition is to accurately transcribe spoken words into written text. In recent years, there have been significant advancements in speech recognition technology, driven by the rapid development of deep learning techniques. As a result the speech recognition systems are becoming more accurate and widely used in a range of applications. In order to achieve high accuracy, speech recognition systems typically involve several stages, including feature extraction, sound modelling and predicting. The continued development and improvement of speech recognition technology has the potential to greatly impact the way we interact with computers and revolutionise the power of AI.**

Index Terms—**Neural Networks, LSTM, Bidirectional, Speech Recognition, CNN, Key word spotting.**

I. Introduction

This paper introduces different types of architectures and compares them based on their performance and computing time. We designed different networks for speech recognition with the tensorflow library. The designs are based on different papers. [1]–[3] and own ideas. We designed around 15 models and evaluated their performance. The best and most important 4 of them we are going to present and compare in this paper. Beforehand we used the tensorflow speech command dataset [4]. This dataset has around 100000 samples with with 35 different classes. It is commonly used for speech recognition tasks. From there we carried on with a STFT and transformed it into the mel spectrogram since all of the papers suggesting it to increase performance and efficiency of training. After we fed the spectrograms in to our models and evaluated their performance.

II. Related Work

The related paper are all using mel spectrograms as inputs since it increases the performance and the efficiency of the training. Furthermore all using CNN layers followed by their preferred architecture. Some papers are using only CNN layers [2] and receive a better performance than common DNN's if they limit the number of parameters. Other papers [3] are using RNN layers to their networks. Also they are adding different types of noises to the samples with different SNR and compare them. It's shown that the added noises increase the accuracy of the models significantly. But these papers were working on KWS so they evaluated their performance based on the false rejected rate.

Email: `maxhenning.hoeth@studenti.unipd.it` ID: `2055977`

It is also an idea to add attention layers to the networks like in [1]. They add a bidirectional LSTM layer after the convolutional layer and then feed it into an attention layer. Based on this architecture they were able to receive an accuracy of 96.9 % on the small version of tensorflow speech command dataset [4] with 12 classes.

III. Signals & Processing

Beginning with the dataset we build using the tensforflow api which provided a function called audio_dataset_from_directory so after downloading the dataset we were able to directly able to create it. We were using a batch size of 32 since one sample has a quite big amount of data. Also since we were doing multi-class classification we used integers as labels to calculate the cross-entropy later. The audio files are '.wav' files which are one second in time and sampled with a frequency of 16kHz. So a sample consists out of 16000 data points. We then applied the a short time fourier transformation.

It is a widely used method in the field of digital signal processing and audio signal analysis. It is used to analyse a time-varying signal in terms of its frequency over short time intervals. The idea behind STFT is to divide a longer audio signal into smaller overlapping segments, and then compute the fourier transform of each of these segments. The result is a matrix where the rows represent frequency and the columns represent time. This matrix was used to create the spectrogram of the the input signal. As parameters for the STFT we used a window length of 550 and 128 frame steps. Then we applied a transformation to these spectrogram which converted them into the mel scale.

The mel scale is a perceptual scale for frequencies. Since the human ear has a non-linear relationship to it. For example a human can easily tell the difference between 500Hz and 1000Hz but not between 10000Hz ans 10500Hz. So the mel scale is transforming the frequency also to a non-linear scale which is way better to perceive for the human ear. After some testing we decided to take a frequency range from 0 to 3000 to almost all of the frequencies and create 120 bins. We decided to take these parameters since we wanted a smooth and not corrupted mel spectrogram since for same values the spectrograms git corrupted.

IV. Learning Framework

We created around 15 models with different architectures from only CNN layers in various configurations, LSTM layers with or without bidirectional wrapper, interconnected

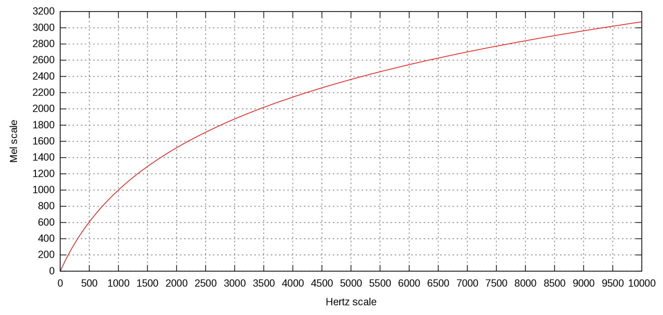

Fig. 1: Mel scale [5]

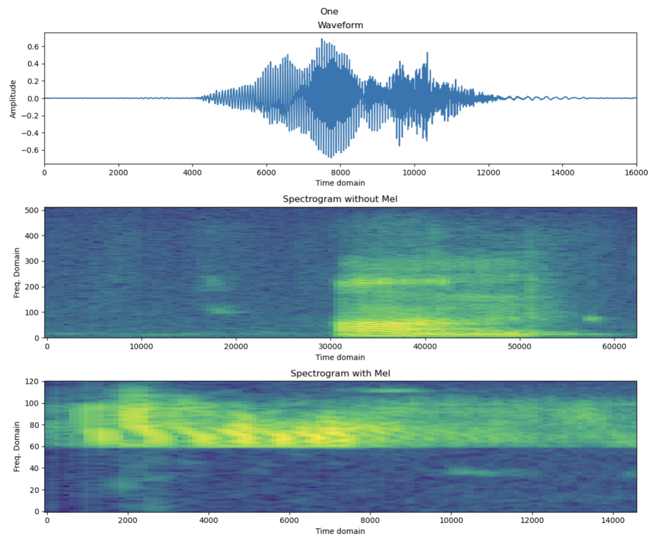

Fig. 2: Audio Signal, STFT and Mel of the word "One" from the top

Conv1DLSTM layer to attention layer. The most interesting and best performing models were:

- **CNN Bi3ConvLSTMPooling**
- **CNN Bi2ConvLSTMPooling**
- **CNN Bi2ConvLSTM**
- **CNN Bi2ConvLSTM˙AT**

In general we always started to resized the input to a (64, 64) grid because otherwise the training would've taken too long. Then we always started with a CNN layer with kernel 9, 9, (5, 1), (5, 1) respectively. After we always applied Batch-normalization and MaxPooling. Then the whole block was repeated with slightly different kernels 3, 3, (2, 1), (2, 1). That was the basic building block for all networks. From there the experimenting started with different architectures. The **CNN Bi3ConvLSTMPooling** used three layers of Conv1DLSTM layers which is an normal LSTM layer but the input and recurrent transformation are convolutional with kernels 5, 2, 1. These kernels were wrapped by a bidirectional layer. It means the LSTM cell receives the time-dependent input backwards and forwards. After this we applied again a MaxPooling layer and then two dense layers with softmax as output.
The **CNN Bi2ConvLSTMPooling** had the same structure but we only used two Conv1DLSTM layers on this one. For the third model **CNN Bi2ConvLSTM** we removed the MaxPooling after the Conv1DLSTM layers to see which affect it had and for the last model **CNN Bi2ConvLSTM˙AT** we had the same architecture but added a attention layer after the Conv1DLSTM since that's a similar architecture to [1] which had a very good accuracy.
Also for every model was fed with the same training data for 15 Epochs which equals around 40000 steps. Besides we decided to take a learning rate with a polynomial decay and monitored the training with tensorboard.

Model	Tot. param.	trainable	non-trainable
CNN Bi3ConvLSTMPooling	124,425	124,382	43
CNN Bi2ConvLSTMPooling	121,945	121,902	43
CNN Bi2ConvLSTM	470,105	470,062	43
CNN Bi2ConvLSTM˙AT	21,068	21,025	43

TABLE 1: Parameters of given models

V. Results

As discussed above we not only evaluate their performance but also the computation time since some models had a considerably longer training time. Notable things are that the model with attention which was supposed to be good had a worse accuracy than the models without the attention even if doesn't have the worst loss. This can be due to the training time since attention models may need a longer time to converge. Also we tried to use the same implementation as [1] so it is probably not the fault of the implementation.

Model	Accuracy	Loss
CNN Bi3ConvLSTMPooling	89.65 %	0.4927
CNN Bi2ConvLSTMPooling	88.77 %	0.5405
CNN Bi2ConvLSTM	88.82 %	0.7116
CNN Bi2ConvLSTM˙AT	81.51 %	0.6180

TABLE 2: Evaluated model with the validation set

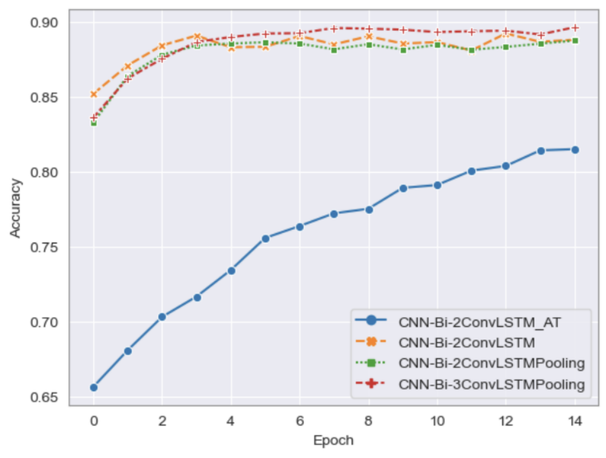

Fig. 3: Validation Loss per Epoch

As we can see here the attention model converges much slower and is not even fully converged after 15 epoch or

40000 steps. The other models in comparison are really fast and could've stopped after 4-5 epochs. Here we have the

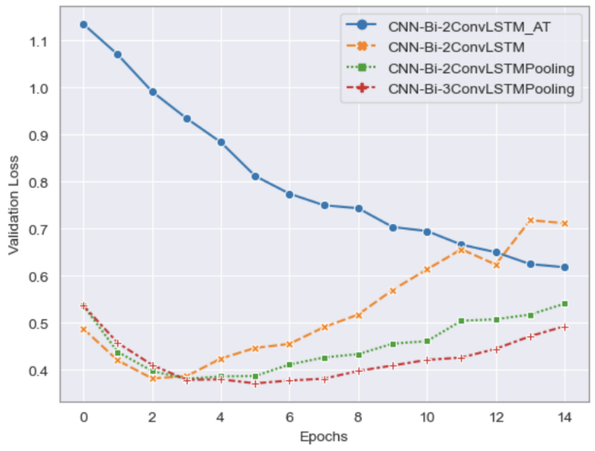

Fig. 4: Validation Accuracy per Epoch

same phenomena and it's even easier to see that it would've been useful to stop the training of the other models after 4 epochs but again the attention model is not converged yet. Also we can observe the weird behaviour of the model **CNN Bi2ConvLSTM** where it is still more accurate but the loss raises and even crosses the one of the attention model. In 5

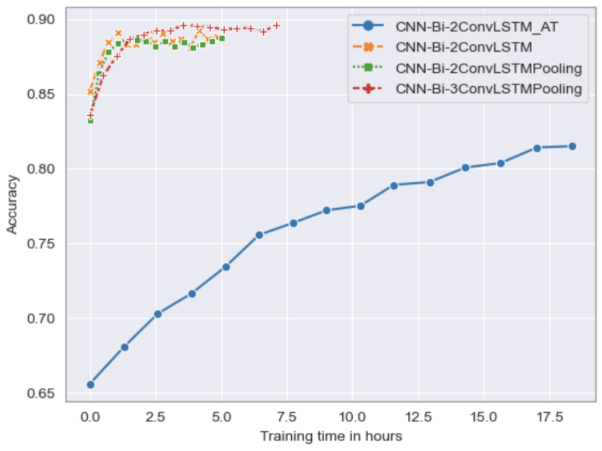

Fig. 5: Validation Accuracy over time

we can see how slow the attention model really is and how slowly it converges. Again all the models were trained on the same data and the attention model took almost three times the time to train as the other model just to receive an even worse accuracy. All the other models would've been ready after around 2.5 hours on the other hand the attention model took up to 18 hours for the same data. That's around 620 % faster in training. For converging the model would probably take same amount of time again.
Also we tested the time for predicting an input which had similar results. Also 3 shows that even the time for predicting an input is almost four times higher for the attention model.

Model	Time	Time/Batch
	s	ms/batch
CNN Bi3ConvLSTMPooling	14	41
CNN Bi2ConvLSTMPooling	14	41
CNN Bi2ConvLSTM	15	43
CNN Bi2ConvLSTM˙AT	55	165

TABLE 3: Time of predicting validation set

Nevertheless we also created the confusion matrix for all of the models and as we can see in 6 the majority of the predicted labels are still correct.

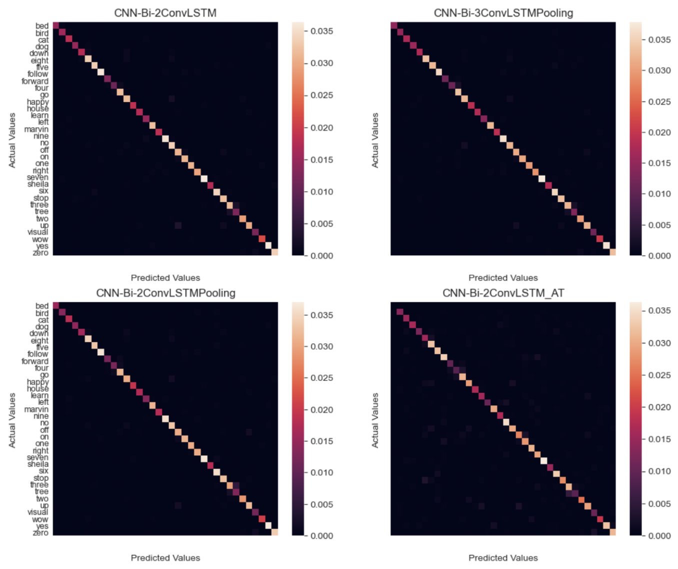

Fig. 6: Validation Accuracy over time

VI. Concluding Remarks

We re-implemented different state-of-the-art speech recognition models and evaluated their performance. It was shown that the attention model, which received an incredible high accuracy in different papers [1], needs a long time to converge in comparison to models without attention layers. Hence in our training scenario the model had a worse performance than the other models also the computation time of this model was significantly higher. Besides this the best one was a model based on convolution layers and bidirectional LSTM layers. These types of models were converging after around 10000 steps and reached accuracy of around 90% on the 35-word speech_command dataset from tensorflow [4].

References

[1] D. Coimbra de Andrade, S. Leo, M. Loesener Da Silva Viana, and C. Bernkopf, "A neural attention model for speech command recognition," *ArXiv e-prints*, Aug. 2018.
[2] T. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," in *Interspeech*, 2015.
[3] S. Arik, M. Kliegl, R. Child, J. Hestness, A. Gibiansky, C. Fougner, R. Prenger, and A. Coates, "Convolutional Recurrent Neural Networks for Small-Footprint Keyword Spotting," pp. 1606–1610, 08 2017.
[4] P. Warden, "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition," *ArXiv e-prints*, Apr. 2018.
[5] K. Vedala, "https://commons.wikimedia.org/w/index.php?curid=3775197,"

Appendix

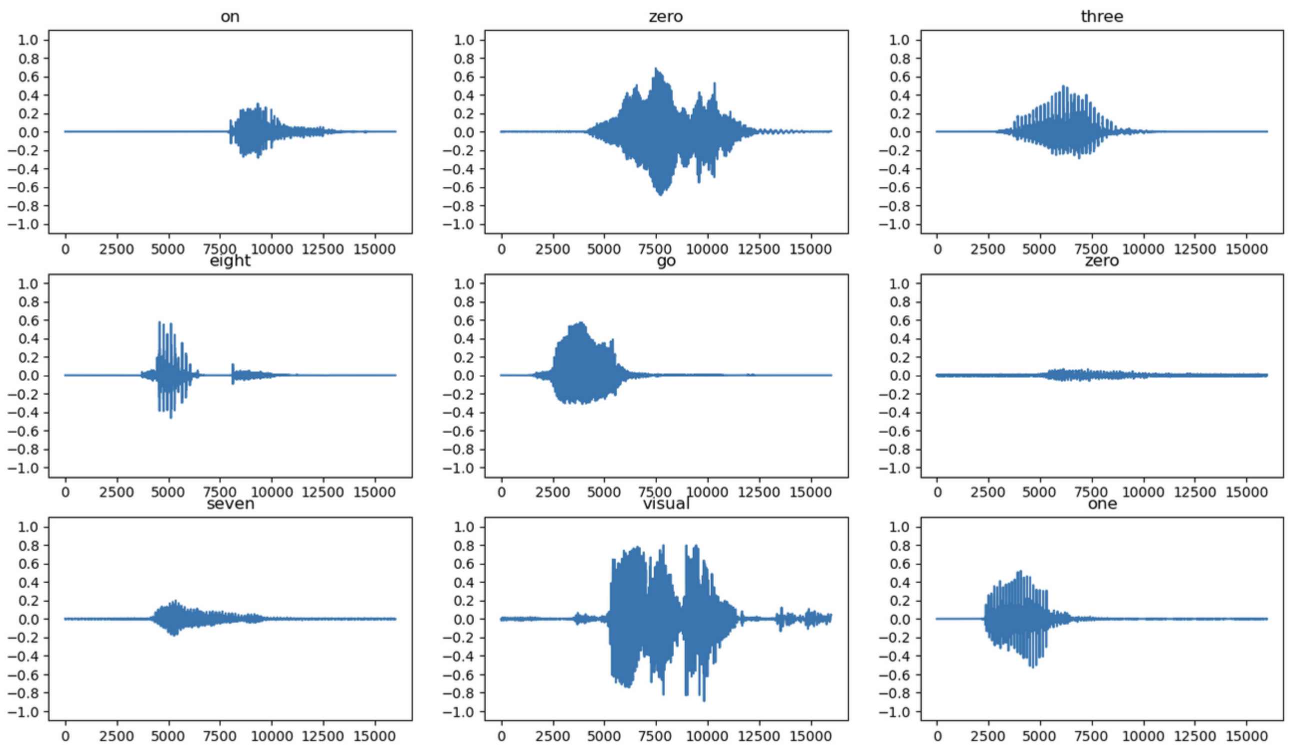

Fig. 7: Example signals from the dataset [4]

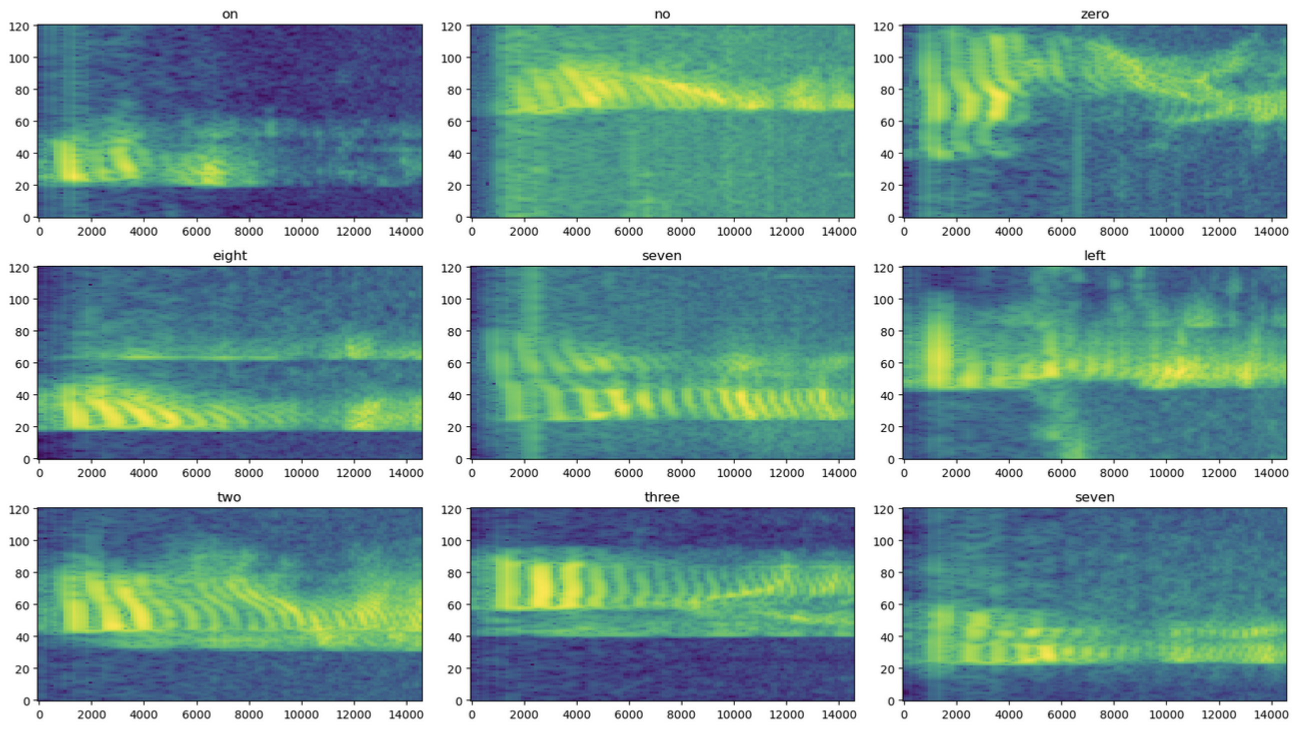

Fig. 8: STFT spectrograms transformed to mel scale