Deep Learning Project II – Report

Transformers – Speech commands classification with Transformers

Michał Gromadzki and Lukas Joisten

6th July 2025

Contents

1 Introduction

Deep Learning is nowadays heavily used in speech recognition and natural language processing. Transformers turned out to be a suitable deep learning technique to classify speech into spoken words. In this project, we train a convolutional neural network (CNN) and a Transformer and use these models to classify speech commands.

In Section 2, we describe the research problems we are going to investigate in this project. Then, in Section 3, we give some theoretical background on the topic of speech classification and transformers. In Section 4, we describe our experiments and how to reproduce the results we discuss afterwards in Section 6. Finally, in Section 7, we summarise our findings and give hints for future work.

2 Research Problem

The topic of this project is 'Speech commands classification with Transformers'. We use the Speech Commands Dataset from the TensorFlow Speech Recognition Challenge hosted on the kaggle platform [4] as our dataset to train and test our models on. It consists of 30 different classes and includes 65,000 samples in total. The samples are 1-second audio snippets with a sampling rate of 16,000 Hz. Figure 1 shows the distribution of samples per category in the dataset.

We tested and compared different network architectures. We chose using a simple CNN and a transformer network, both adjusted for audio classification tasks. We tested different parameter changes on these network and investigated the influence the changes had on the result.

Finally, we present a confusion matrix on both network architectures.

3 Background

In this section, we first give background information on speech command recognition in general, then we will give details about speech command recognition performed by a CNN and a transformer.

3.1 Speech Command Recognition

A program that takes as an input human speech, interprets it, and transcribes it into text is called 'Speech Recognition Software' [1]. This technique is used heavily in recognising speech commands, for example in IoT devices like Amazon Alexa or Google Home.

Recognising these commands has always been challenging, because of the presence of background noise and the variability of pitch, speed, and other characteristics a natural language has to offer [1].

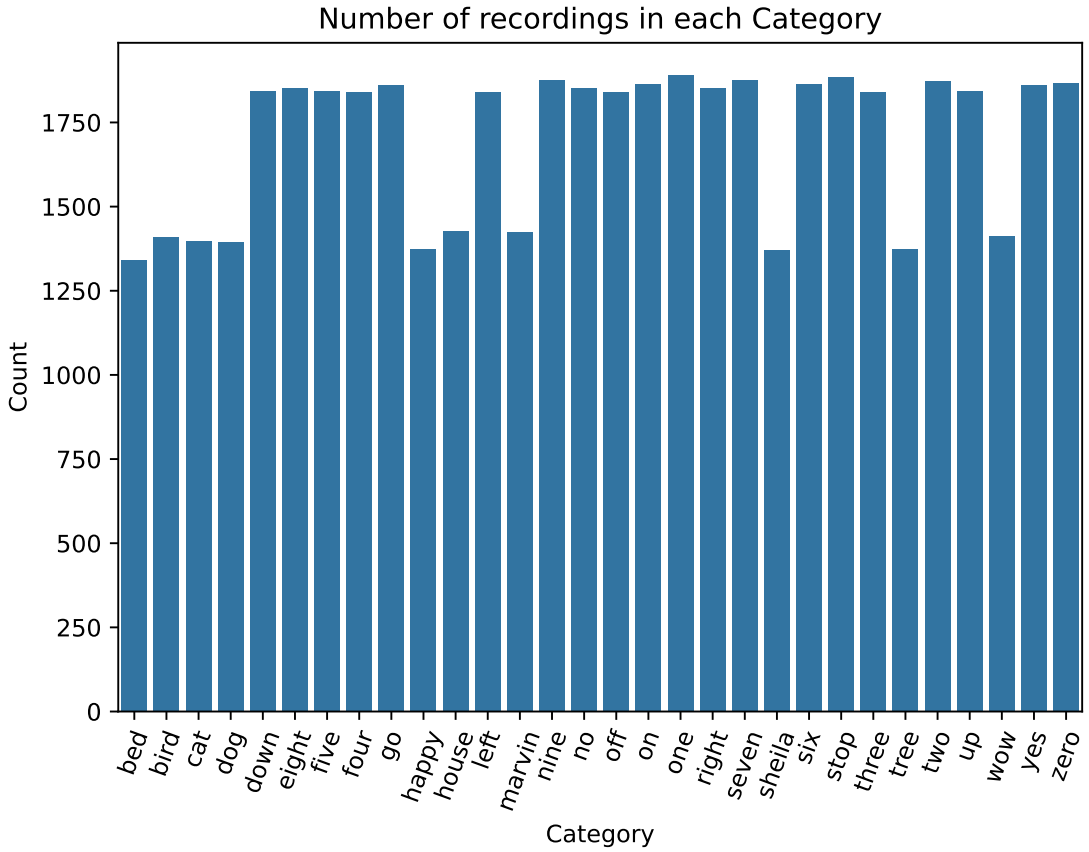

Figure 1: The number of samples for each label in our dataset

3.2 CNNs in speech command recognition

Deep Neural networks can be considered as a suitable technique for speech recognition [1]. One can train them for a large amount of data which then results in a high accuracy in recognising human speech and isolating spoken commands.

Ayache et al. [1] presented a CNN to recognise speech commands intended for the use on robots controllable by these speech commands. They gain an accuracy of 92-95% on their validation, train and test sets. Nevertheless their training time took a minimum of 5 minutes and 15 seconds on GPUs.

3.3 Transformers

The idea behind the transformer model has been introduced by Vaswani et al. [7] in 2017. It is intended for the use in natural language processing and generating machine translations. A transformer can be also used in speech recognition scenarios and in those it performs much faster than a recurrent neural network (RNN) [5].

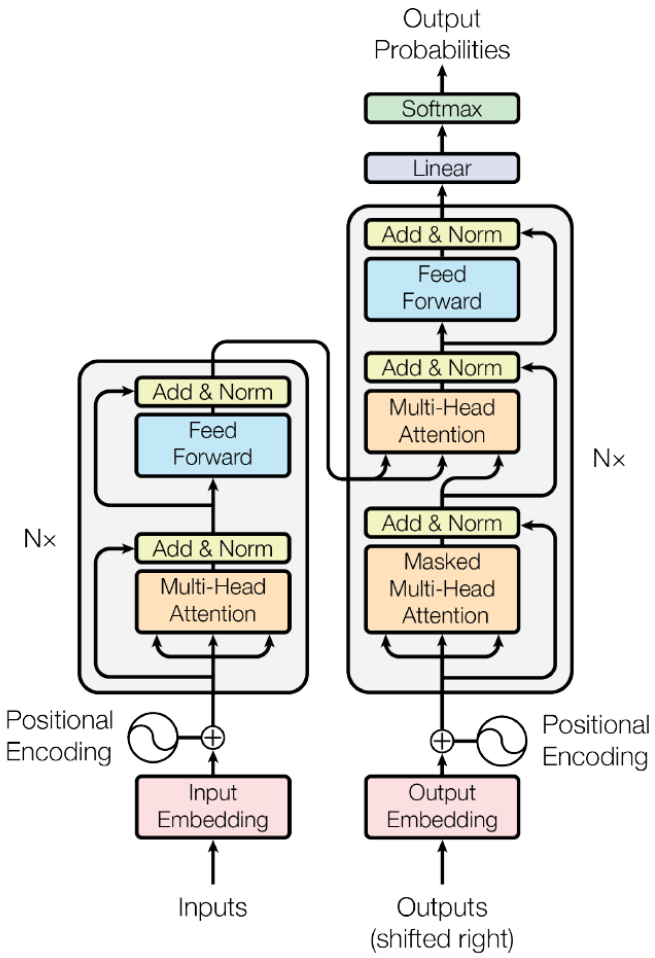

Figure 2: The Transformer model architecture from [7]

Figure 2 depicts the model's architecture. The model follows the encoder-decoder structure [7]. The encoder maps an input sequence of symbol representations to another sequence of continuous representations. From these representations, the decoder generates an output sequence of symbols one element at a time. In each step, the model consumes the previously generated symbol as an additional input when generating an output text.

The core of the model are the self-attention sublayers [7]. These allow the model to assign weights to the importance of different words. Attention is a function that maps a query and a set of key-value pairs to an output, where all 3 parameters are all vectors. The output is computed as the weighted sum of query and key-value pairs. The transformer model uses a so-called 'Scaled Dot-Product Attention' to calculate the attention scores between each pair of tokens. That is calculated by the following formula: $\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$, where d_k is the dimension of the queries and

keys.

In order to jointly attend to information from different projections to different dimensions of query, keys and values, the transformer uses so-called 'Multi-Head Attention' [7]. The Scaled Dot-Product Attention is applied multiple times in parallel, each with its own set of projections of query, keys and values. After calculating these scores for every attention head, these scores are concatenated and transformed linerly to get the get the multi-head score.

The positional encoding makes sure that positional information about the tokens is incorporated into the model [7]. During training, a positional encoding in form of vectors, is learned to encode information about the absolute and relative position of the tokens.

Additionally to the self-attention sublayers, each layer in encoder and decoder includes a fully connected feed-forward network (FFN) [7]. It consists of two linear transformations with ReLU as activation function. The FFN is applied to each to each token's position separately and identically which allows to capture more complex relationships between tokens.

With this architecure the transformer model is very much suitable for natural language processing and speech recognition tasks and is replacing RNNs in that field of research rapidly [5].

4 Methodology

4.1 Models

We have tested 3 different model architectures. In this section, we present them and give insights into the training of these models.

4.1.1 CNN

First model is a convolutional neural network (CNN). We obtained it from the tensorflow tutorial. The architecture begins with an input layer followed by a resizing layer, which downsamples the input to the size of $(32, 32)$. Normalization is then applied to standardize the input data. Subsequently, two convolutional layers with increasing filter sizes (32 and 64) and ReLU activation functions extract hierarchical features from the input. Max-pooling reduces spatial dimensions. Dropout layers with rates of 25% and 50% are employed to regularize the network, mitigating overfitting by randomly deactivating neurons during training. The flattened output is then fed into dense layers for further feature extraction and classification, culminating in an output layer with the number of units corresponding to the number of labels for prediction. Figure 3 presents the described architecture.

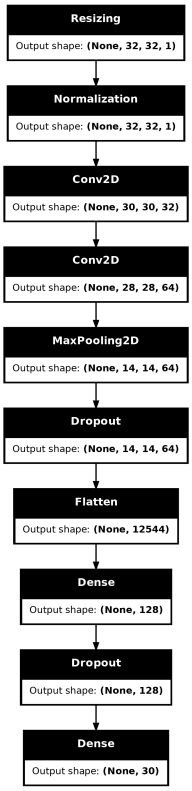

Figure 3: Architecture of CNN model

4.1.2 Transformer

Second architecture is a Transformer-based model tailored for audio classification tasks. The model architecture consists of several key components. First, the Positional Embedding layer incorporates positional information into the input sequence using convolutional layers and positional encodings. Next, the SelfAttention mechanism computes attention weights based on the input sequence itself, facilitating capturing relevant context. The FeedForward layer processes the output of the self-attention module through two fully connected neural networks with ReLU activation and dropout. The DecoderLayer combines self-attention and feedforward networks, forming the fundamental building block of the Transformer. The Decoder class stacks multiple DecoderLayers. Finally, the Transformer class orchestrates the entire model, integrating the decoder and a final dense layer for classification, enabling the model to make predictions based on the learned representations.

4.1.3 Whisper-tiny

The last tested architecture was based on Whisper-tiny[6], a general-purpose speech recognition model, developed by OpenAI. Whisper is also a transformer-based model, however, it is capable of completing much more speech processing tasks than just classification.

The model, we developed, uses Whisper-tiny as a feature extractor. The features are then passed on to two dense layers, which serve the purpose of classification head.

4.2 Data Augmentation

Audio samples are augmented by adding background noise with a certain probability p. The function takes, as input, a waveform tensor representing the audio sample, a list of tensors representing background noises waveforms, and the probability p of applying augmentation. Inside the function, it first checks whether augmentation should be applied based on the probability p. If augmentation is to be applied, it randomly selects a background noise waveform from the provided list, aligns it with the length of the input waveform by selecting a random starting point, and mixes it with the original waveform using a random mixing coefficient from alpha distribution. The resulting waveform, now augmented with background noise, is returned. We used probability $p = 0.7$. Figures 4 and 5 present a waveform before and after augmentation.

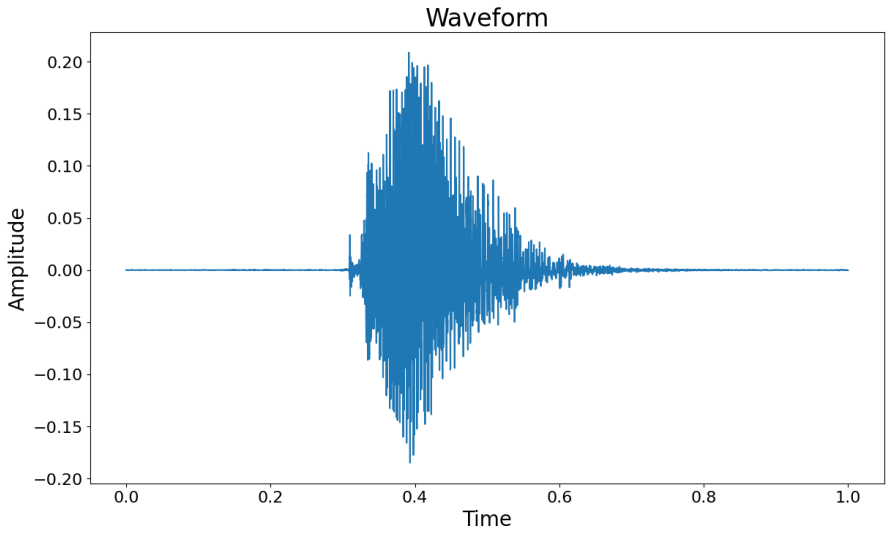

Figure 4: Sample waveform

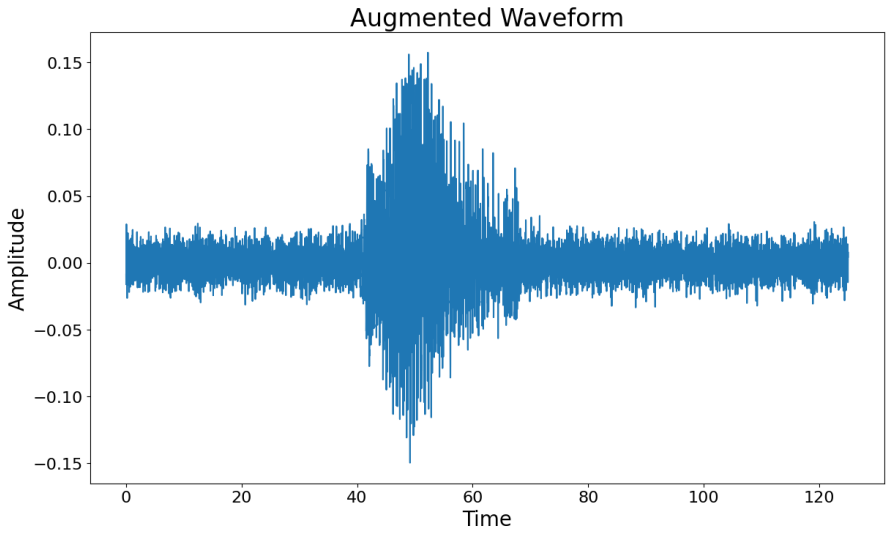

Figure 5: Augmented waveform

4.3 Training process

All models were trained for 100 epochs or until there was no improvement in the best validation accuracy for 4 consecutive epochs. All models, apart from Whisper-tiny were trained with Adam optimizer. During training we have tested two variations of learning rates and spectrograms.

Learning rates could either be stable for the whole training process or change according

to a schedule, based on [8]. The formula for the changing learning rate is:

$$lrate = d_{model}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5})$$

We used $warmup\_steps = 4000$. Figure 6 presents the changing learning rate.

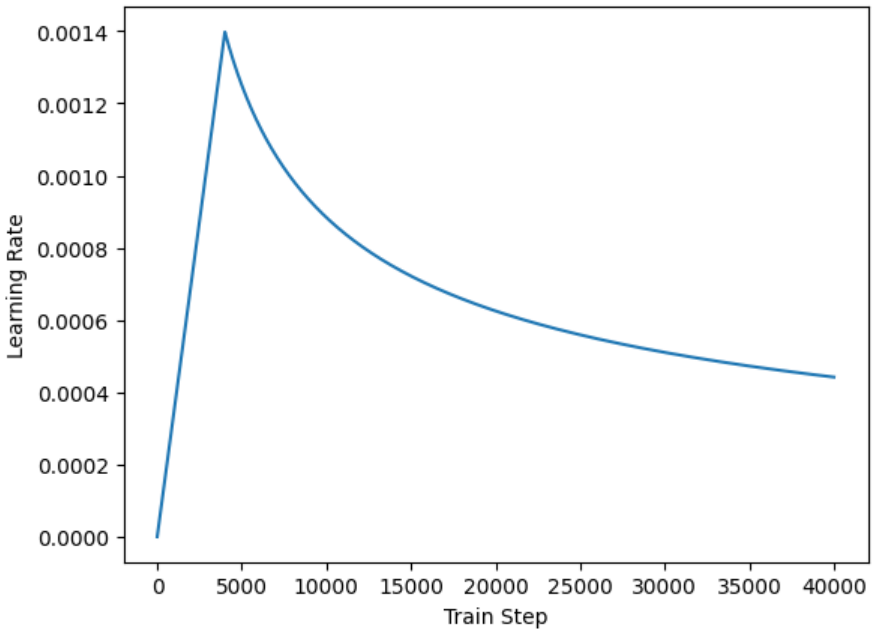

Figure 6: Learning rate based on train step

A spectrogram is a visual representation of the spectrum of frequencies and their intensity of a signal as it varies with time [3]. Figure 7 shows the waveform and the spectrogram of a sample of the word 'yes'. We also used a log-mel spectrogram. That is a variation of the spectrogram that incorporates some preprocessing steps to better represent human audio perception. While spectrograms represent the direct magnitude of frequencies over time, log-mel spectrograms include mel-scaling and logarithmic compression to better align with human audio perception. Because a log-mel spectrogram captures more meaningful features, it is commonly used in speech recognition tasks.

4.3.1 Whisper-tiny

Whisper-tiny uses a large spectrogram size of (None, 80, 3000), hence we needed to make several steps to make training possible on available hardware. We decided to use a much less popular optimizer - Lion[2]. Lion is an optimizer known for its small memory usage. Using this optimizer, we achieved a batch size of 2. Using Adam optimizer we couldn't start the training process, even with the batch size of 1. Based on the authors' suggestions we used a learning rate about 10 times smaller than with Adam optimizer - $1e-5$. We trained the model for 3 epochs which is equal to about 75 000 training steps.

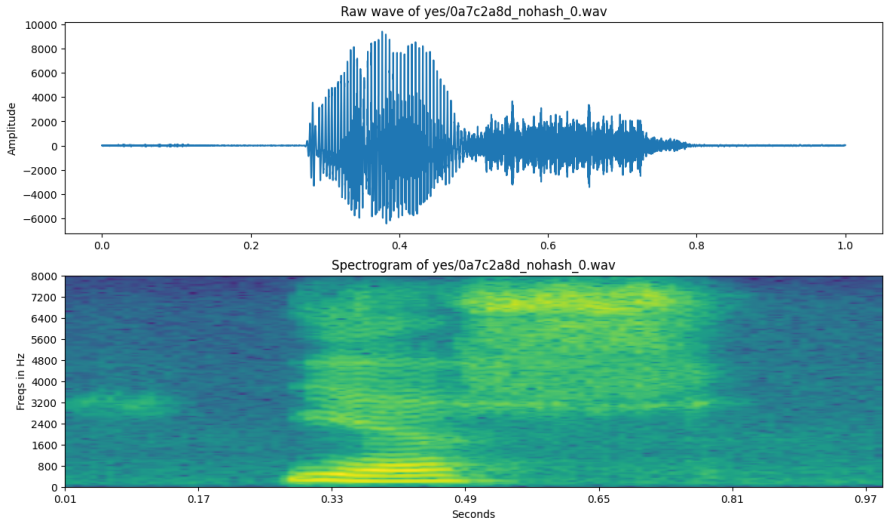

Figure 7: Waveform and spectrogram of the word-sample 'yes'

4.4 Table of all trained models

Table 1 summarises all models we trained during our experiments.

Model	**Batch size**	**Learning rate**	d_{model}	**nl**	**nh**	**Spectrogram**	**DA**
CNN	32	0.001	None	None	None	Normal	No
Transformer	256	0.001	256	2	2	Normal	No
Transformer	256	0.001	256	4	2	Normal	No
Transformer	128	0.0001	256	4	4	Normal	No
Transformer	128	0.0001	512	4	4	Normal	No
Transformer	256	0.001	256	2	2	Log-Mel	No
Transformer	256	0.001	256	4	2	Log-Mel	No
Transformer	128	0.0001	256	4	4	Log-Mel	No
Transformer	128	0.0001	512	4	4	Log-Mel	No
Transformer	256	Schedule	256	2	2	Log-Mel	No
Transformer	256	Schedule	256	4	2	Log-Mel	No
Transformer	128	Schedule	256	4	4	Log-Mel	No
Transformer	128	Schedule	512	4	4	Log-Mel	No
Transformer	256	0.001	256	2	2	Log-Mel	Yes
Transformer	256	0.001	256	4	2	Log-Mel	Yes
Transformer	128	0.0001	256	4	4	Log-Mel	Yes
Transformer	128	0.0001	512	4	4	Log-Mel	Yes
Whisper-tiny	2	0.00001	384	4	6	Log-Mel	No

Table 1: Table of all trained models

5 Results

Performance of the models was evaluated based on loss and accuracy achieved on the test dataset. Table 2 includes metrics on testing datasets of all trained models.

Model	d_{model}	**nl**	**nh**	**n_params**	**Spectrogram**	**DA**	**Loss**	**Acc**
CNN	None	None	None	1 628 446	Normal	No	0.553	0.857
Transformer	256	2	2	2 4251 18	Normal	No	0.415	0.881
Transformer	256	4	2	4 530 462	Normal	No	0.337	0.915
Transformer	256	4	4	6 633 758	Normal	No	0.391	0.912
Transformer	512	4	4	26 243 614	Normal	No	0.359	0.917
Transformer	256	2	2	2 425 118	Log-Mel	No	0.394	0.892
Transformer	256	4	2	4 530 462	Log-Mel	No	0.311	0.915
Transformer	256	4	4	6 633 758	Log-Mel	No	0.336	0.912
Transformer	512	4	4	26 243 614	Log-Mel	No	0.309	0.937
Transformer	256	2	2	2 425 118	Log-Mel	No	0.338	0.908
Transformer	256	4	2	4 530 462	Log-Mel	No	0.326	0.906
Transformer	256	4	4	6 633 758	Log-Mel	No	0.414	0.881
Transformer	512	4	4	26 243 614	Log-Mel	No	0.391	0.893
Transformer	256	2	2	2 425 118	Log-Mel	Yes	0.457	0.873
Transformer	256	4	2	2 425 118	Log-Mel	Yes	0.430	0.877
Transformer	256	4	4	6 633 758	Log-Mel	Yes	0.329	0.930
Transformer	512	4	4	26 243 614	Log-Mel	Yes	0.362	0.922
Whisper-tiny	384	4	6	$\sim$ 30 000 000	Log-Mel	No	**0.278**	**0.945**

Table 2: Metrics of all trained models

6 Discussion

Firstly, let's focus on performance of model in relationship with number of parameters. Plot 8 depicts a scatter plot presenting this relationship.

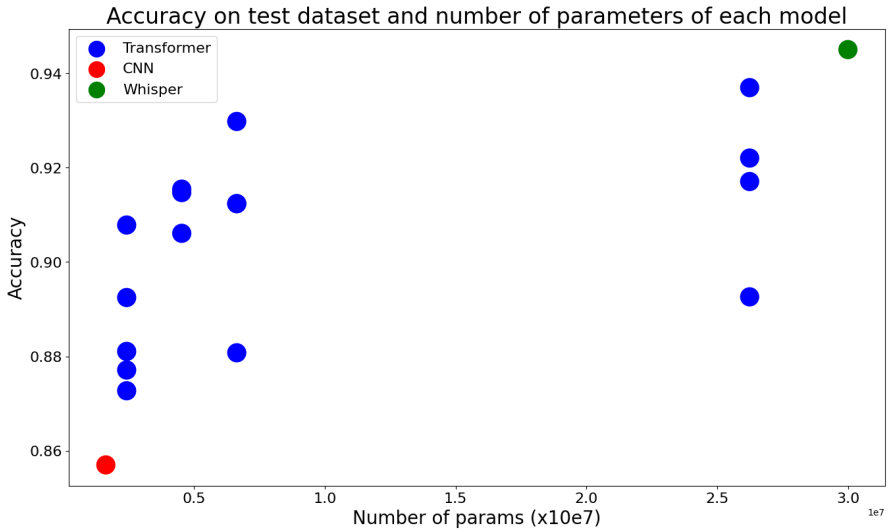

Figure 8: Augmented waveform

Based on the plot 8 we can say that the best performing model is Whisper-tiny. Performance of the transformer models gradually increase with their size. The worst performing model is CNN.

Secondly, we examine the impact of increasing the 3 main transformer parameters - d_{model}, num_layers and num_heads. Figure 9 depicts change in performance based on values of those parameters.

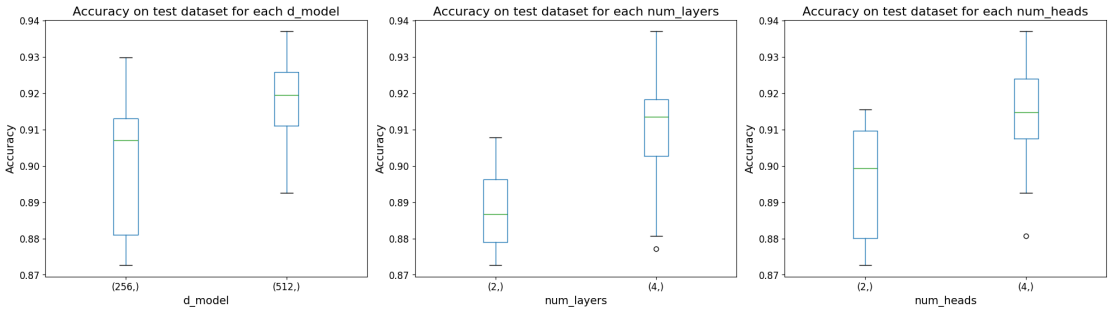

Figure 9: Augmented waveform

In all cases increase in those parameters causes improvement in performance. The biggest improvement was caused by increase the num_layers parameter.

During training we have tested 3 different learning rates, Plot 10 presents the mean accuracy based on each learning rate.

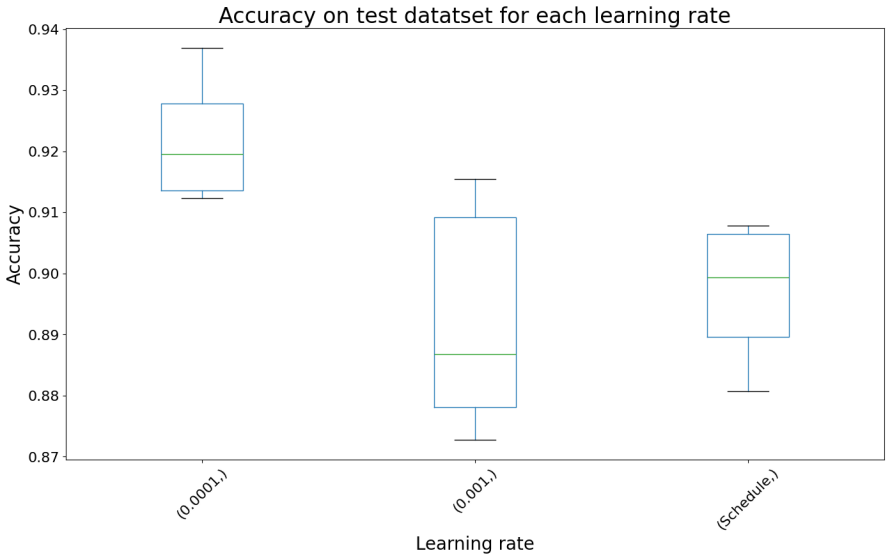

Figure 10: Augmented waveform

Surprisingly scheduled learning rate, which is proposed by the authors of original transformer[8], performs the worst. The best performing learning rate is 0.0001, however, it is due to the fact that is was used for training bigger models.

During training also tested different types of spectrograms, notably the default version and log-mel spectrogram. Figure 11 presents difference in models' performance based on the selected spectrogram type.

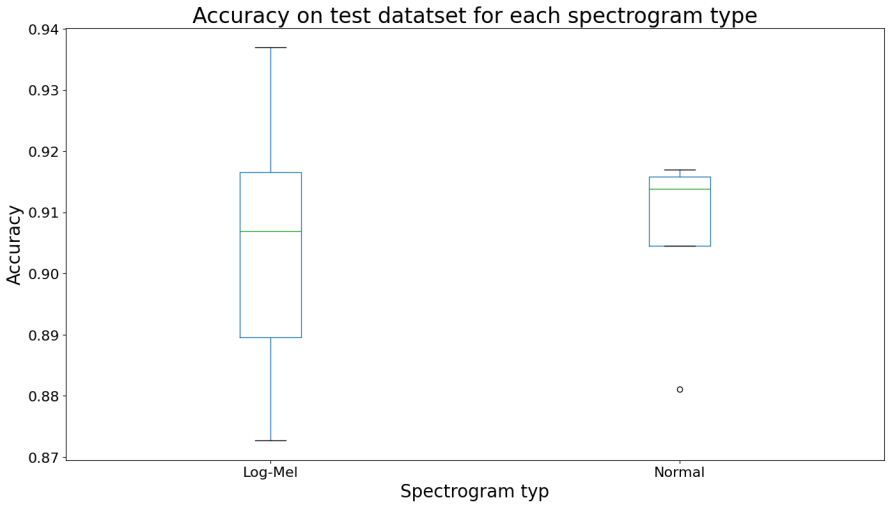

Figure 11: Augmented waveform

Both types perform similarly. However it is worth noting that in a direct comparison, exact 4 models, while the only difference is the used spectrogram, the log-mel spectrogram performed better. Thus, we decided to only use log-mel spectrograms in further tests.

Lastly, we experimented with data augmentation. Plot 12 depicts performance of models with and without data augmentation.

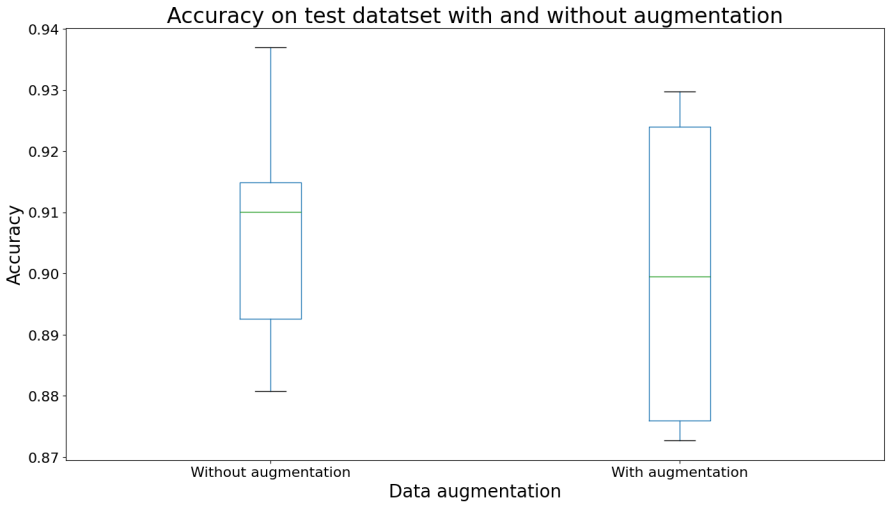

Figure 12: Augmented waveform

Surprisingly, models without any data augmentation performed better.

6.1 Confusion Matrix

Figures 13 and 14 present confusion matrix for CNN and Whisper-tiny.

For the CNN we can observe that the classification results are pretty good after for words like ‘eight’ or ‘zero’, but for other words, like ‘three’ and ‘tree’ still get confused by the model. This can be explained with these words sound similar even for us humans. But also words like ‘no’ and ‘go’ get mixed up by the model’s classification slightly. This could be from the bad quality of some samples or similar sounds of some speakers.

For our whisper-tiny model, we get overall better classification results. We’ve got very few wrong predictions, but nevertheless ‘three’ and ‘tree’ still get wrongly classified more often than other words, but less intense than in our CNN model. The reason for the better performance might be the more elaborateness of the tiny-whisper model compared to the CNN we used.

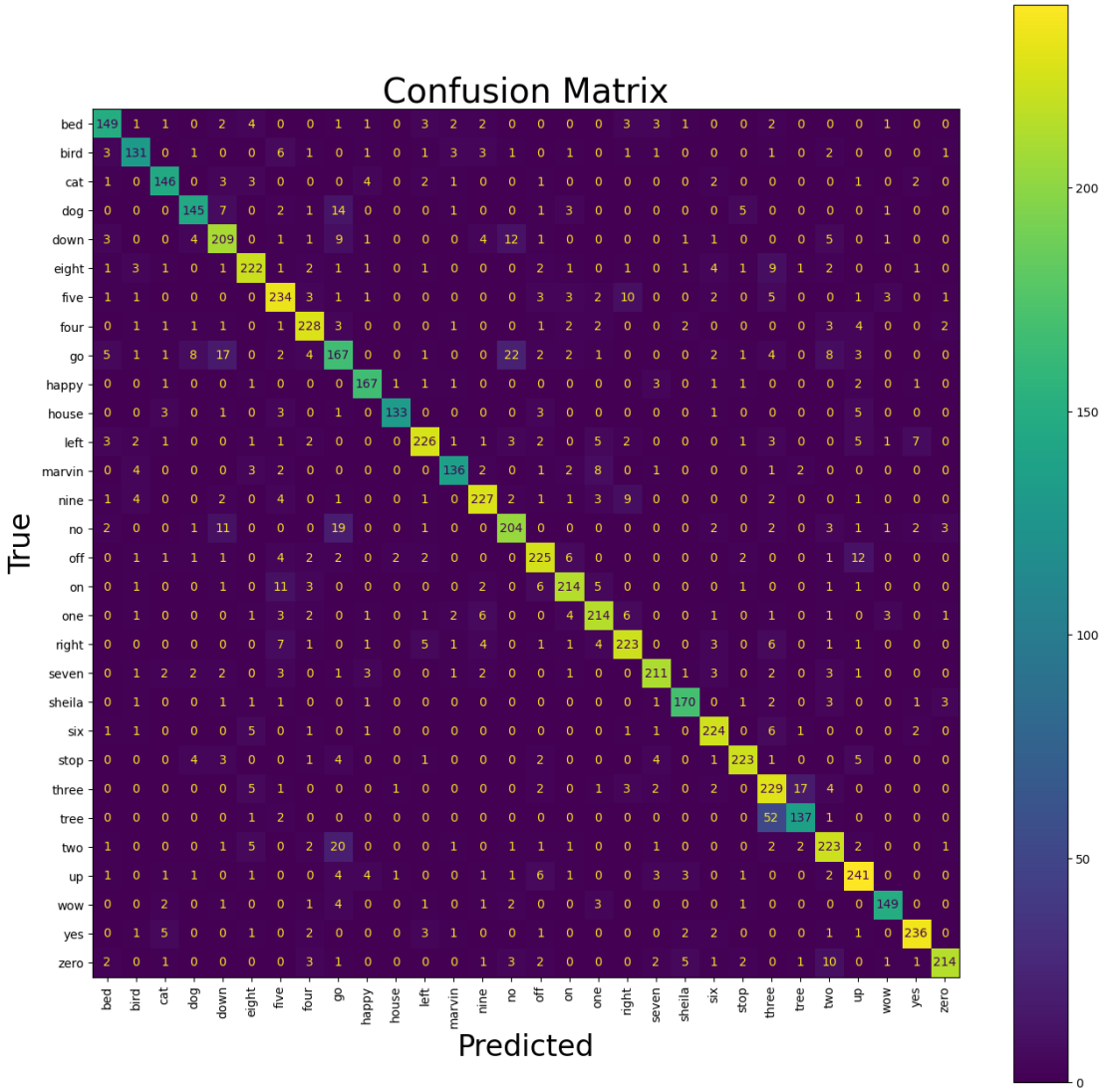

Figure 13: The confusion matrix of our CNN

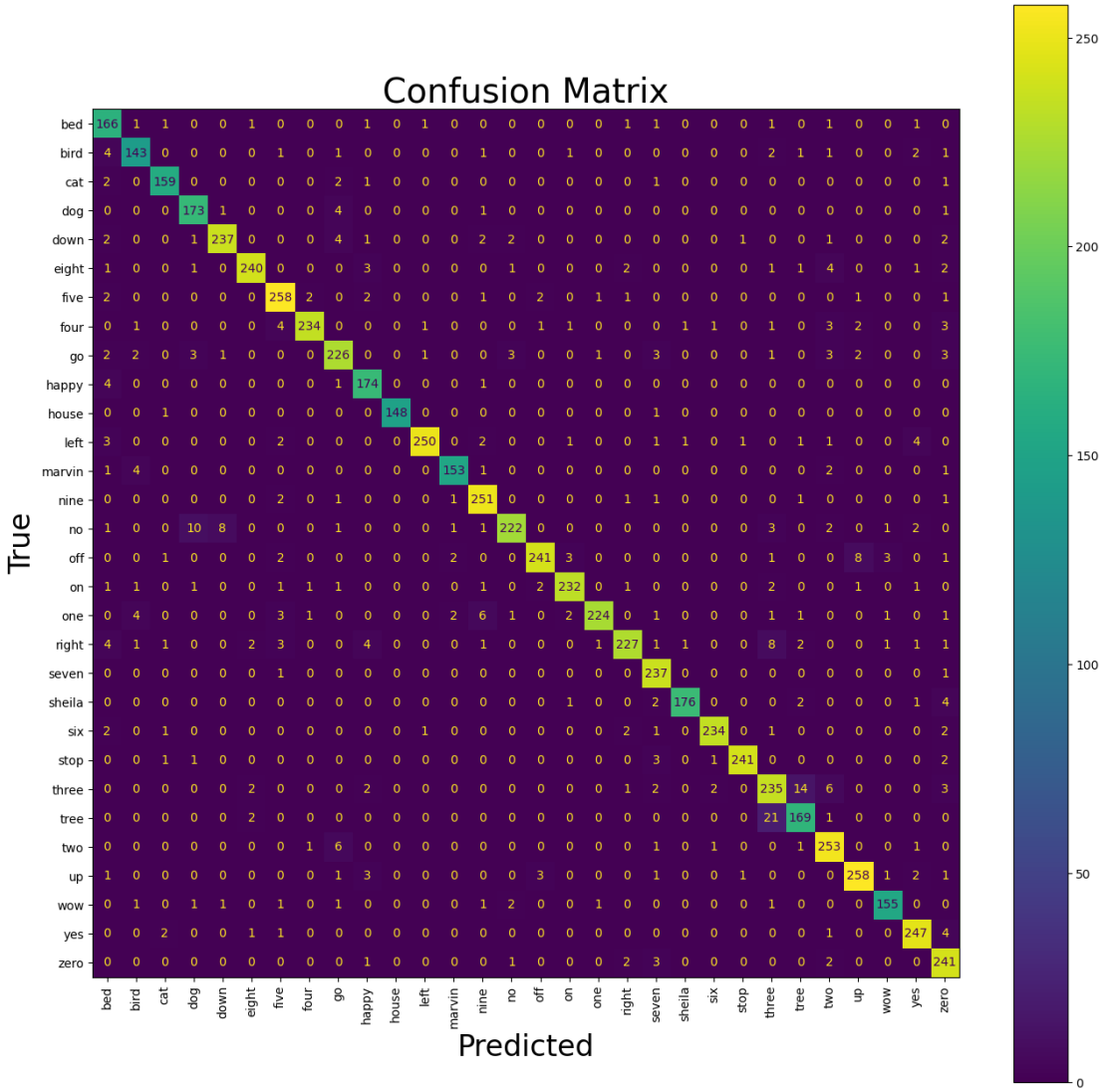

Figure 14: The confusion matrix of Whisper-tiny

7 Conclusion

We classified audio samples according by the use of a CNN, a transformer and the tiny-whisper model. We gave some background information how the transformer model works behind the scenes and we also gave an insight on how we created the models and how we trained them.

We gain pretty good accuracy by using the whisper-tiny model, but also the transformer does not perform bad. The CNN on the other hand still has some space for

improvements. We observed that increasing the d_{model}, num_layers and num_heads improves the transformer's performance. For our use case, the best performing learning rate is 0.0001.

In the confusion matrices, we saw that the CNN does more wrong classifications than the whisper-tiny model.

Future Work

Future work should elaborate on models in different languages than English. There are still many languages all over the world, speech recognition models could be adjusted to.

References

[1] Mohammad Ayache, Hussien Kanaan, Kawthar Kassir and Yasser Kassir. 'Speech Command Recognition Using Deep Learning'. In: *2021 Sixth International Conference on Advances in Biomedical Engineering (ICABME)*. 2021, pp. 24–29. DOI: 10.1109/ICABME53305.2021.9604862.

[2] Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Yao Liu, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu and Quoc V. Le. *Symbolic Discovery of Optimization Algorithms*. 2023. arXiv: 2302.06675 [cs.LG].

[3] Hugging Face. *Introduction to audio data*. accessed on 05-05-2024. URL: https://huggingface.co/learn/audio-course/chapter1/audio_data#spectrogram.

[4] inversion, Julia Elliott, Mark McDonald, Pete Warden and Raziel. *TensorFlow Speech Recognition Challenge*. 2017. URL: https://kaggle.com/competitions/tensorflow-speech-recognition-challenge.

[5] Shigeki Karita, Nanxin Chen, Tomoki Hayashi, Takaaki Hori, Hirofumi Inaguma, Ziyan Jiang, Masao Someki, Nelson Enrique Yalta Soplin, Ryuichi Yamamoto, Xiaofei Wang, Shinji Watanabe, Takenori Yoshimura and Wangyou Zhang. 'A Comparative Study on Transformer vs RNN in Speech Applications'. In: *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. 2019, pp. 449–456. DOI: 10.1109/ASRU46091.2019.9003750.

[6] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey and Ilya Sutskever. *Robust Speech Recognition via Large-Scale Weak Supervision*. 2022. arXiv: 2212.04356 [eess.AS].

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser and Illia Polosukhin. 'Attention is All you Need'. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser and Illia Polosukhin. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].