Warsaw University of Technology

FACULTY OF
MATHEMATICS AND INFORMATION SCIENCE

Speech Commands Classification with Recurrent Neural Networks

Adam Majczyk 313420, Szymon Matuszewski 313435

Version 1.0

07.05.2024

Contents

1 Introduction

1.1 Problem Definition

Speech Classification problems have become very popular throughout last years mainly due to the skyrocketing outburst of AI techniques. Some may not notice that many of these techniques' concepts originated decades ago. One of these are the Recurrent Neural Networks (*RNNs*), which were supposedly first introduced by David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams in 1985 [8]. Their popular descendant Long Short Term Memory (*LSTM*) Network appeared in 1997 as breakthrough in storing information over extended time intervals (therefore fixing the vanishing and explodind gradient problems of vanilla RNNs) via recurrent backpropagation and its developers are Sepp Hochreiter and Juergen Schmidhuber [4].

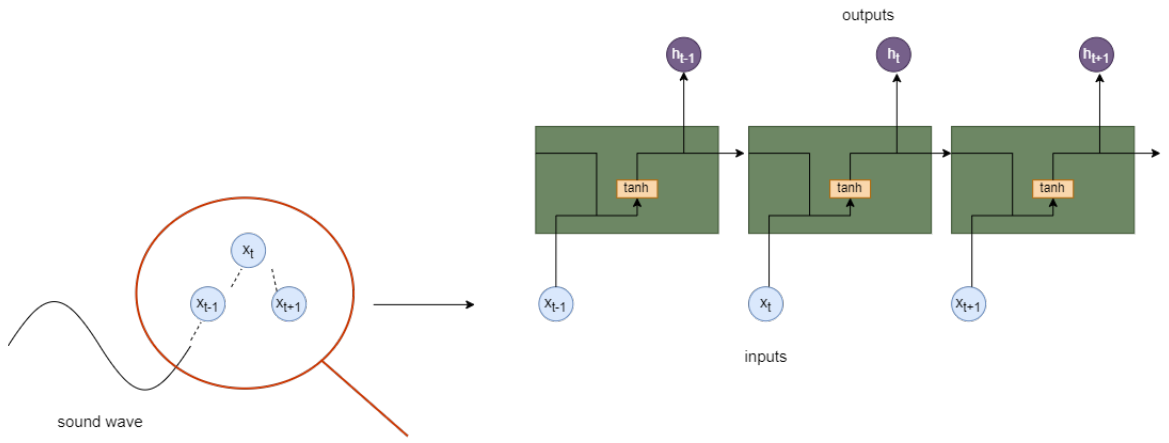


Figure 1: Layers flow in a classic RNN architecture.

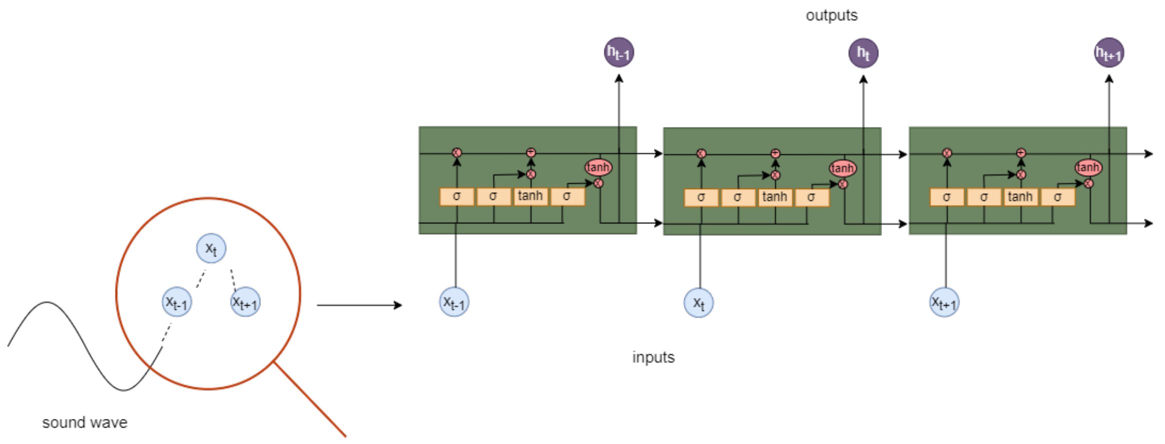


Figure 2: Layers flow in a classic LSTM architecture.

Inspired by biological nature of neurons such a network is combined of multiple connected layers. According to [5], a classic RNN architecture consists of chunks of layers where each chunk have one hyperbolic tangent layer and corresponds to one specific time slot (i. e. last second from now, last but one second from now) and its output (See Fig. 1).

The LSTM network is more advanced RNN solution with additional sigmoid layers within each chunk (See Fig. 2). These layers are responsible for different features:

1. **Sigmoid** - describes how much of each component should be let through. Zero lets nothing through, while one lets everything through.
2. **Tanh** - Decides the direction and magnitude of information. The values range from -1 to 1.

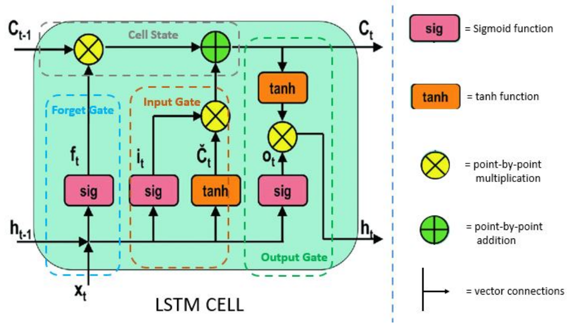


Figure 3: LSTM cell gates, `https://www.pluralsight.com/resources/blog/guides/introduction-to-lstm-units-in-rnn`

The functions are combined to form 3 gates:

- **Forget gate** - decides how much of the information that is currently in the network to use. Uses sigmoid.
- **Input gate** - decides how much and in what direction to use the new input. Uses sigmoid and tanh.
- **Output gate** - updates the hidden state (the lower - h_t and h_{t-1} line in Fig. 3 and the cell state the upper line - c_t and c_{t-1} in Fig. 3). Uses both tanh and sigmoid.

This report explores the task of speech classification using Recurrent Neural Networks on the Speech Commands Dataset [9]. This dataset is a large benchmark dataset composed of recordings of different words. In our case we focused on recordings of 11 classes: (*"down"*): **"go"**, **"left"**, **"no"**, **"off"**, **"on"**, **"right"**, **"stop"**, **"up"**, **"yes"**, **"unknown"**. All recordings are in **.wav** format. Our whole dataset consists of 26050 recordings. The recordings of class **"unknown"** consists of different 20 not distinguished classes from [9] cut to the average length of other classes. That is, there were 23682 files in the 10 classes, so 2368 files from the 20 other classes were randomly selected.

Our study aims to investigate the effectiveness of different LSTM architectures and techniques for speech classification on this dataset. We evaluate different approaches, such as transformers and custom LSTMs.

1.2 Approaches

We attempted to train LSTMs in PyTorch. In our dataset, we have three subsets: train, validation, and test, summarising up to **26,050** recordings. Each train-validation-test split is perform in given order: **95%** of data for training, **2.5%** for validation, **2.5%** data for testing. A seed has always been applied to ensure that the datasets are the same within the 5 training iterations (iterations, not to confuse with epochs).

The test dataset is used exclusively for evaluating the trained model's performance.

For each (of 5) iteration of training the algorithms the new train set is shuffled with deterministic random seeds (each iteration with a different seed).

As a loss function cross-entropy was applied in all training settings.

LSTM

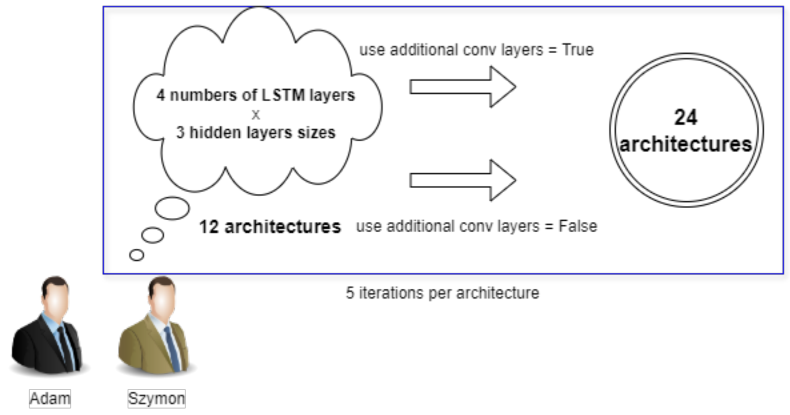

4 numbers of LSTM layers
x
3 hidden layers sizes
12 architectures
use additional conv layers = True
use additional conv layers = False
24 architectures
5 iterations per architecture
Adam
Szymon

Figure 4: Pipeline for training classic LSTMs manually defined with a grid.

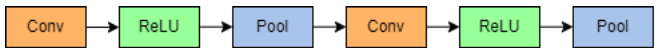


Figure 5: Layers flow in a our CNN architectures for LSTMs.

Considering long time needed for training many architectures of Recurrent Neural Networks and Transformers we decided to define grid configurations for training our LSTM Networks (See Fig. 4). Tested architecture parameters were as follow:

- **num_layers** - a number of LSTM layers in the architecture
- **hidden_sizes** - a number of recurrent LSTM layers
- **use_conv** - a boolean parameter determining if additional convolutional layers in the architecture are used (a good practice) (See Fig. 5)

Each configuration was passed through **5** iterations per **10** epochs for training. Then the best set of architecture configuration was chosen based on the *mean accuracy* on the test dataset (See Fig. 4).

During training we also used the optimisation algorithm and learning rate. These were chosen manually, as an optimiser: **ADAM**, as a learning rate: **0.001**. In the later research it suffices as these settings provide satisfactory results.

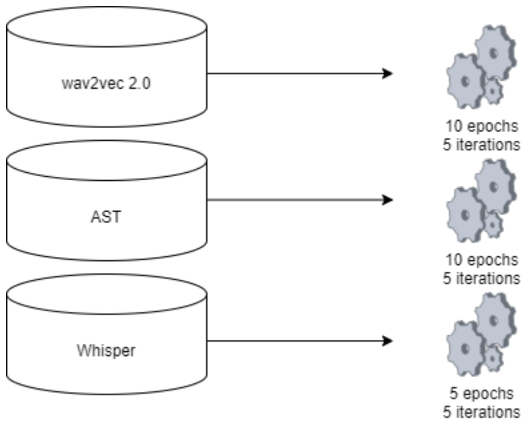


Figure 6: Pipeline for training transformer architectures.

Transformers

We managed to test 3 transformer architectures (See Fig. 6) so as to validate LSTMs scores with state-of-the-art solutions for the speech classification problem. The tested transformers are:

1. **wav2vec 2.0** [1]
2. **Audio Spectrogram Transformer** [3]
3. **Whisper** [7]

NOTE: Each but one transformer (**Whisper**) was passed through 5 iterations per 10 epoch of training. **Whisper** was passed through 5 iterations per 5 epochs due to time consuming training (1 epoch approx. 1h10min). Additionally, **wav2vec 2.0** was initially optimized with Grid Search for a learning rate and a optimiser. Other transformers' training time did not allow us to apply Grid Search.

2 LSTM

2.1 Architectures

Tested parameters in our LSTM architectures:

1. **num_layers** - [2, 4, 6, 10]
2. **hidden_sizes** - [32, 64, 128]
3. **use_conv** - True or False

That summarises up to **24** tested architectures. Each architecture was passed through 5 iterations per 10 epochs.

2.2 Feature Extractors

To enhance LSTM performance the models were boosted with **MFCC** *Mel Frequency Cepstral Coefficient* feature extractor with settings as follow:

- sample_rate = 16000
- n_mfcc = 13
- n_fft = 400
- hop_lenght = 160
- n_mels = 23

2.3 Experiments

As the Table 1 indicates the best performing configuration of architectures it the one with: **4** numbers of LSTM layers, hidden sizes **64** and **with** additional convolutional layers. The mean test accuracy **0.855** is reasonably good and uncovers the potential of such architectures.

Best LSTM model - the model from best iteration from the best performing configuration of architecture (**4** numbers of LSTM layers, hidden sizes **64** and **with** additional convolutional layers).

2.4 Results

Not surprisingly, additional convolutional layers significantly improve the test accuracy for small and medium number of LSTM layers. It is worth noticing that there is a number of LSTM layers when the models collapse (See Fig. 7). Hence, it is recommended to use between 2 and 6 LSTM layers. In our task, **4** layers achieved the best results.

When it comes to hidden sizes of LSTM layers the **64** size outperforms the rest (See Fig. 8). Additionally, it is proved once again that additional convolutional layers are recommended in such models.

Number of LSTM Layers	Hidden Sizes	Additional Conv Layers	test accuracy mean	test accuracy std
4	64	True	0.855	0.012
4	128	True	0.849	0.014
2	128	True	0.844	0.024
6	64	True	0.838	0.034
2	64	True	0.825	0.012
4	32	True	0.816	0.037
2	32	True	0.790	0.048
6	32	True	0.762	0.056
4	128	False	0.754	0.020
4	64	False	0.718	0.009
2	128	False	0.688	0.022
2	64	False	0.655	0.014
4	32	False	0.639	0.016
6	128	False	0.614	0.297
6	32	False	0.575	0.080
2	32	False	0.571	0.024
6	64	False	0.515	0.253
6	128	True	0.506	0.380
10	32	False	0.171	0.096
10	64	True	0.097	0.012
10	32	True	0.091	0.011
10	128	False	0.089	0.010
10	64	False	0.088	0.013
10	128	True	0.084	0.009

Table 1: Mean accuracy performance of LSTM model grouped by number of LSTM layers and sizes of hidden layers.

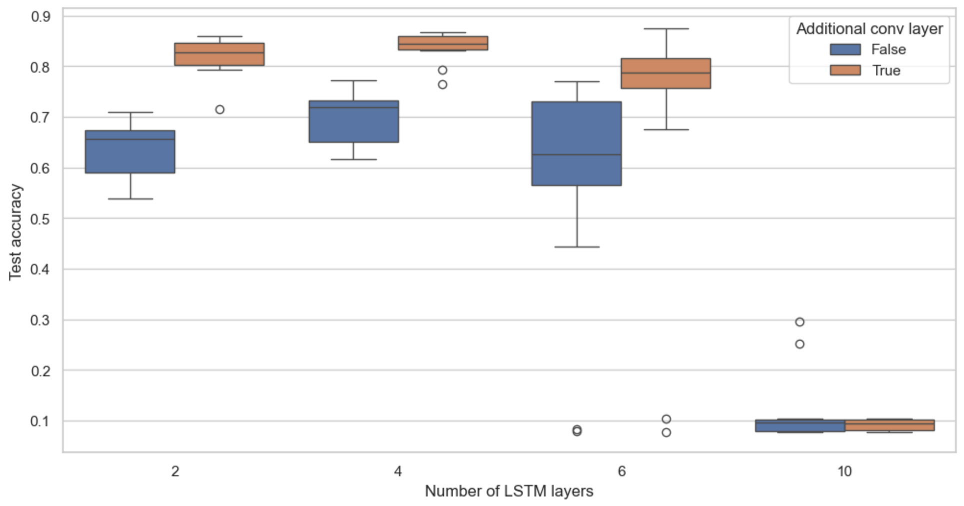


Figure 7: Boxplots showing how number of LSTM layers have an impact on test accuracy.

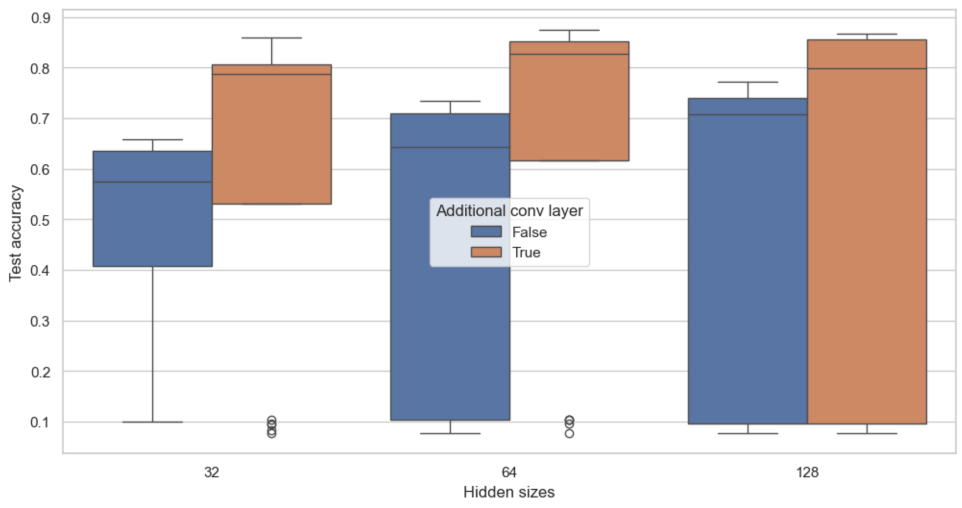
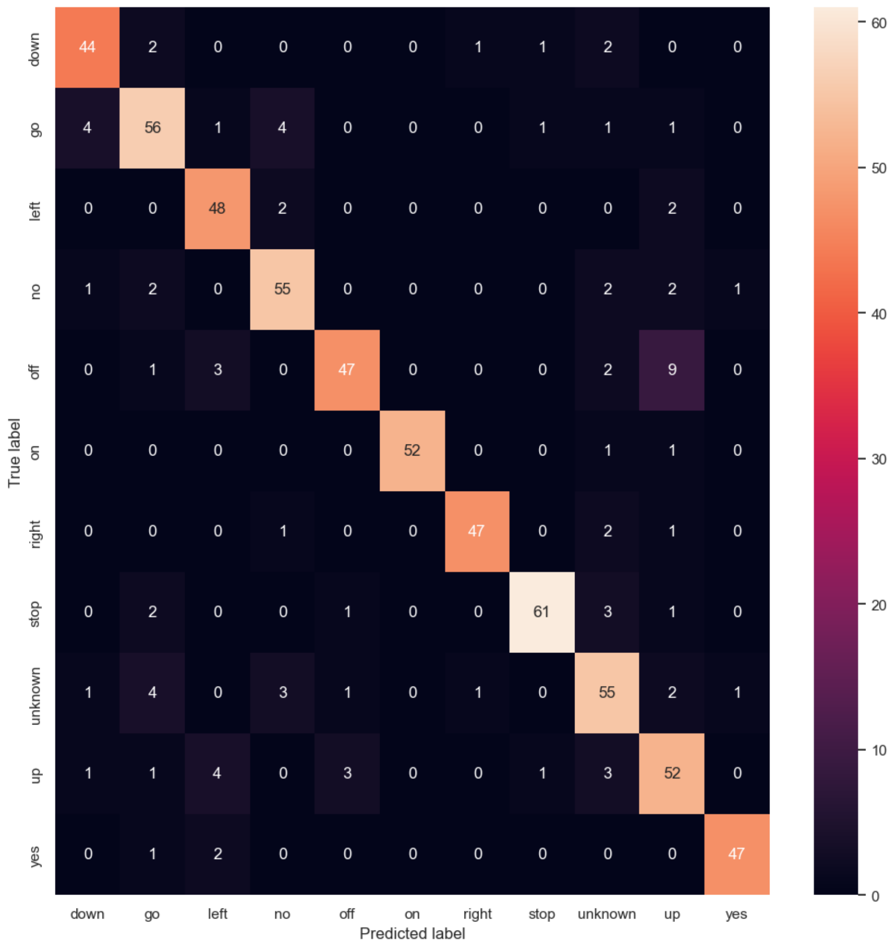


Figure 8: Boxplots showing how hidden layer sizes have an impact on test accuracy.



	down	go	left	no	off	on	right	stop	unknown	up	yes
down	44	2	0	0	0	0	1	1	2	0	0
go	4	56	1	4	0	0	0	1	1	1	0
left	0	0	48	2	0	0	0	0	0	2	0
no	1	2	0	55	0	0	0	0	2	2	1
off	0	1	3	0	47	0	0	0	2	9	0
on	0	0	0	0	0	52	0	0	1	1	0
right	0	0	0	1	0	0	47	0	2	1	0
stop	0	2	0	0	1	0	0	61	3	1	0
unknown	1	4	0	3	1	0	1	0	55	2	1
up	1	1	4	0	3	0	0	1	3	52	0
yes	0	1	2	0	0	0	0	0	0	0	47

Figure 9: Confusion matrix for the best LSTM model performance based on test accuracy for the best mean accuracy settings.

The confusion matrix of the best model from LSTM architectures shows satisfactory results. The only problem recognised here is the resemblance of **'off'** and **'up'** words. Actually, these words sound very similar in different accents so the mistakes are explainable. Other classes of mistakes vary from 0 to 3.

It is also interesting to see, that setting the number of recurrent layers to 10 effectively destroyed the discriminatory capabilities of the classifier. The cause of that effect however unclear.

3 Transformers

3.1 wav2vec 2.0

Wav2vec 2.0 [1] performs better than many other semi-supervised methods by providing a new way of using embeddings learned from speech data and applying fine-tuning based on transcribed speech. Wav2vec 2.0 exhibits competitive speech recognition performance even with limited labeled data.

3.2 Audio Spectrogram Transformer

The Audio Spectrogram Transformer (AST) [3] was first introduced as a convolution-free, purely concept-based audio segmentation algorithm. AST achieves state-of-the-art improvements in a variety of resolutions, including 0.485 mAP with Audioset [2], 95.6% accuracy with ESC-50 [6] and 98.1% accuracy with Speech Commands V2

3.3 Whisper

Whisper [7] is a speech recognition model, trained on 680,000 hours of supervised multilingual work. Whisper uses a Transformer based architecture, which shows robustness to accents, background noise, and technical language. Not optimized for benchmarks.

3.4 Experiments

optimizer	learning_rate	test_accuracy_mean	test_accuracy_std
adam	0.0001	0.937	0.005
sgd	0.0100	0.929	0.003
sgd	0.0010	0.921	0.008
adam	0.0010	0.907	0.008
sgd	0.0001	0.575	0.096
adam	0.0100	0.236	0.351

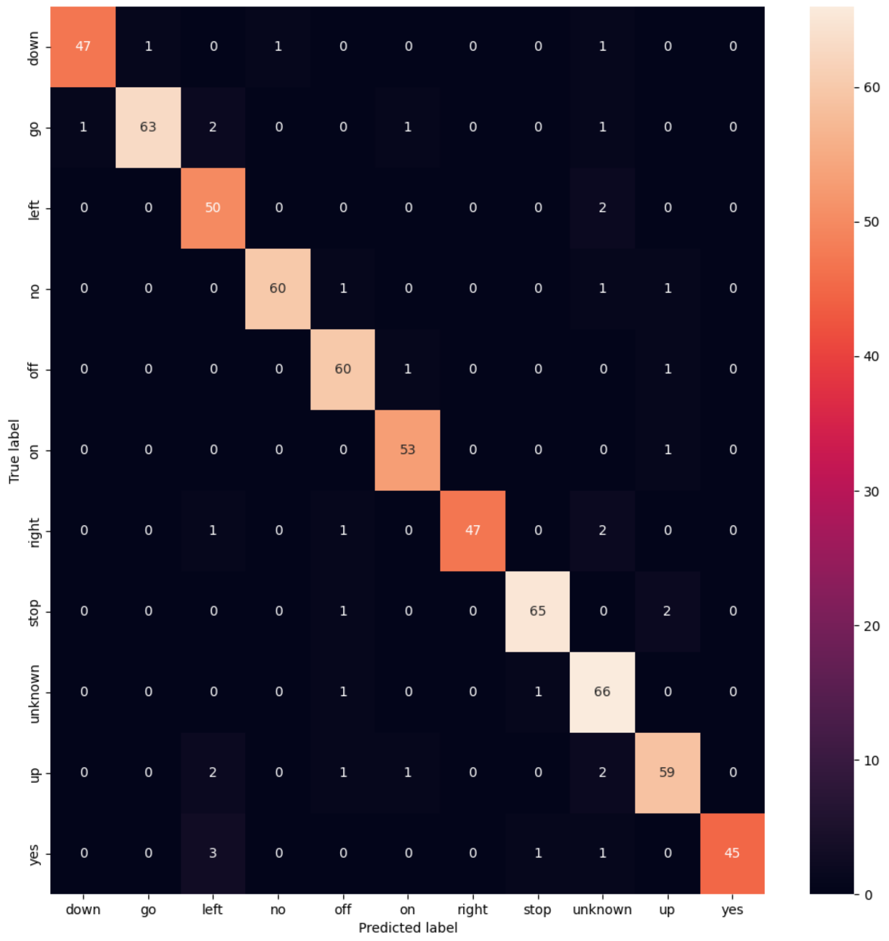
Table 2: Tested hyperparameters on the wav2vec 2.0 transformer. Tested optimizers: ADAM and SGD, tested learning rates: 0.0001, 0.001, 0.01. This was the only transformer to have hyperparameters tested due to time consuming training.

In the experiment described in Table 2 it is proved that **ADAM** optimiser and learning rate **0.0001** achieves the best results with **wav2vec 2.0** transformer. It was the only transformer to have these hyperparameters tested. Thus we decided to set learning rate to **0.001** in all transformers to compare them equally and stick to default value of learning rate as we recognise this experiment as not sufficient to determine the real optimum for all transformers' learning rates.

The table 3 shows the training parameters for all the transformers.

transformer	optimizer	learning rate	criterion	iterations	epochs
wav2vec 2.0	ADAM	0.001	crossentropy	5	10
AST	ADAM	0.001	crossentropy	5	10
Whisper	ADAM	0.001	crossentropy	5	5

Table 3: Transformers' parameters for training.



	down	go	left	no	off	on	right	stop	unknown	up	yes
down	47	1	0	1	0	0	0	0	1	0	0
go	1	63	2	0	0	1	0	0	1	0	0
left	0	0	50	0	0	0	0	0	2	0	0
no	0	0	0	60	1	0	0	0	1	1	0
off	0	0	0	0	60	1	0	0	0	1	0
on	0	0	0	0	0	53	0	0	0	1	0
right	0	0	1	0	1	0	47	0	2	0	0
stop	0	0	0	0	1	0	0	65	0	2	0
unknown	0	0	0	0	1	0	0	1	66	0	0
up	0	0	2	0	1	1	0	0	2	59	0
yes	0	0	3	0	0	0	0	1	1	0	45

Figure 10: Confusion matrix for the best wav2vec 2.0 transformer performance based on test accuracy.

3.5 Results

The confusion matrix of the best iteration from **wav2vec 2.0** underline how incredibly accurate this transformer can be. All mistakes here varies from 0 to 3 per class. Undeniably, this accuracy is close to the human accuracy. There are no classes where a problem occur.

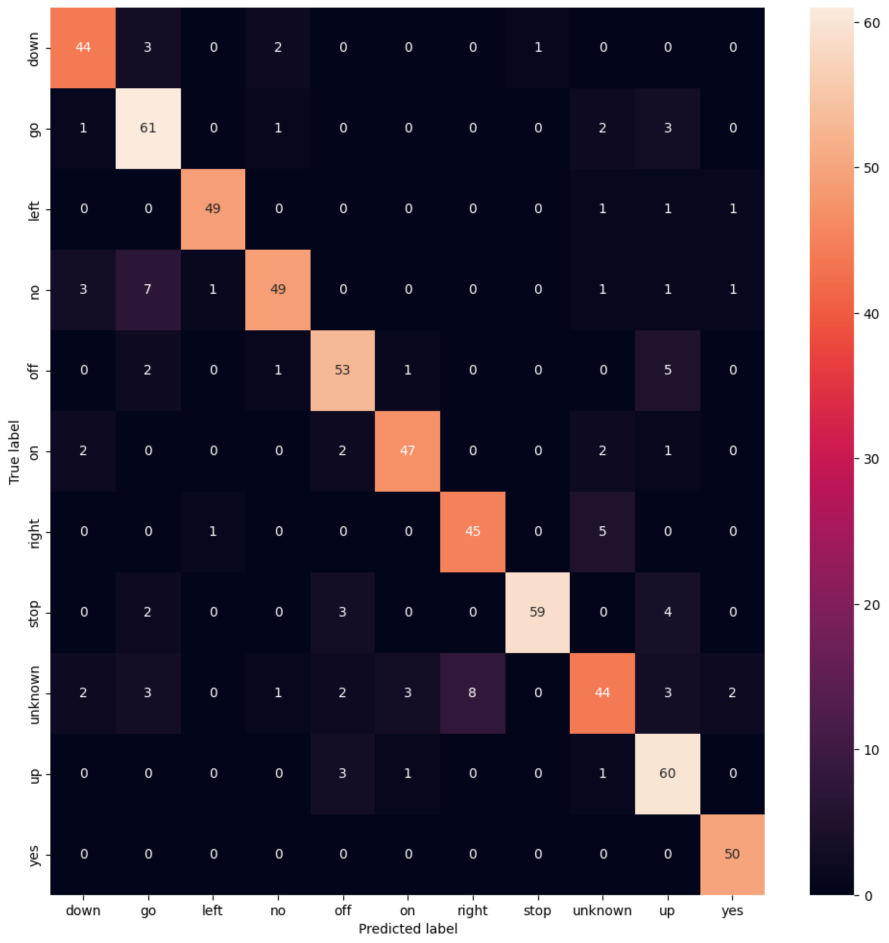


Figure 11: Confusion matrix for the best AST transformer performance based on test accuracy.

The **AST** transformer has its own problems which previous models (LSTMs and wav2vec 2.0) have not had 11. There is a problematic class **'no' - 'go'** which can be easily explainable. The two words are short in lasting and end in the same way. Once again, the **'up' - 'off'** similarity is the obstacle for the algorithm.

The **Whisper** transformer outperforms the AST but is worse than wav2vec 2.0 repeating the **'no' - 'go'** mistakes. It seems that it is slightly overfitted for predicting **'no'** class where words are short (i.e. 'up' or some words classified as 'unknown')

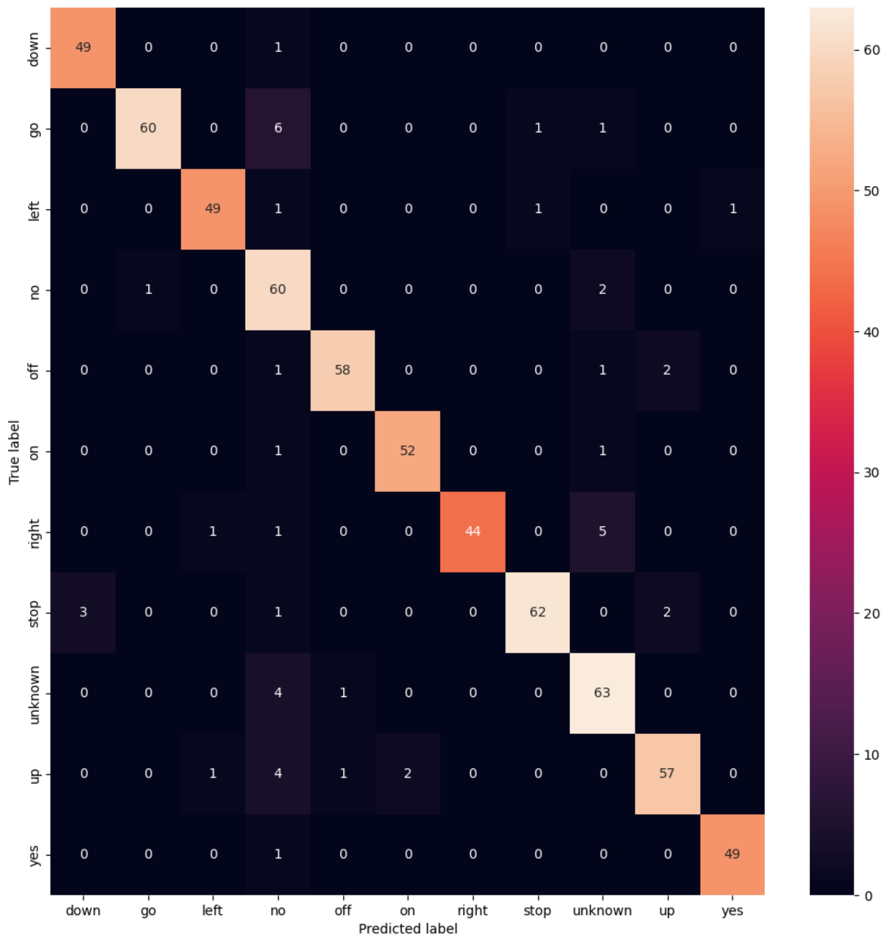


Figure 12: Confusion matrix for the best Whisper transformer performance based on test accuracy.

4 Comparison

Model	**test accuracy**
wav2vec 2.0	0.945
Whisper	0.926
Best LSTM	0.866
AST	0.862

Table 4: Ranking of the best iterations per model. As the Best LSTM model we assume model with 4 numbers of LSTM layers of hidden layers' sizes 64.

Overall, the summary indicates that the **wav2vec 2.0** Transformer outperforms other models, with **Whisper** following closely behind. The traditional **LSTM network** performs decently (escpecially with the addition of CNNs) but falls short of the top-performing models, while the **AST** Transformer lags significantly behind in terms of accuracy.

5 Conclusions

1. The LSTM networks works significantly better when CNN layers are added into the architecture. It is accurately shown by the boxplots Figures 7 and 8.
2. Applying too many layers of LSTM can make the model collapse. It should be recommended to apply up to 6 layers in the Speech Recogition tasks with recordings of 1 word per recording.
3. Transformer architectures often outperforms classic LSTM architectures. The state-of-the-art attention mechanism broadens the horizons for Deep Learning tasks. In the case of the Speech Recognition task it was **wav2vec 2.0** transformer which we assume as the best one.
4. If the word is short enough like **'no'** or **'go'** it poses a difficult for either transformer or LSTM network.
5. It is possible to achieve accuracy similar to the human while using transformers in the Speech Recognition tasks.

6 Instruction

All source code is here: `https://github.com/amajczyk/2024L_DeepLearning_P2/tree/main`

6.1 Prerequisites

Install needed dependencies:

```
pip install -r requirements.txt
```

Now, you are able to regenerate the training pipeline.

6.2 Documentation

Files in main directory:

- `README.md`: This file
- `requirements.txt`: Required packages for the project
- `.gitignore`: Files to ignore in the repository
- `AA_ast_torch.ipynb`: Jupyter notebook with the code for AST model
- `AA_ast_torch_summary.ipynb`: Jupyter notebook with the code for AST model's with summary
- `AA_lstm_torch.ipynb`: Jupyter notebook with the code for LSTM model
- `AA_lstm_torch_summary.ipynb`: Jupyter notebook with the code for LSTM model with summary
- `AA_wav2vec_torch.ipynb`: Jupyter notebook with the code for Wav2Vec model

- `AA_wav2vec_torch_summary.ipynb`: Jupyter notebook with the code for Wav2Vec model with summary
- `AA_wav2vec2_torch_hyperparams.ipynb`: Jupyter notebook with the code for Wav2Vec2 model with testing of hyperparameters (use_conv, num_lstm_layers, hidden_size)
- `AA_wav2vec2_torch.ipynb`: Jupyter notebook with the code for Wav2Vec2 model
- `AA_whisper_torch.ipynb`: Jupyter notebook with the code for Whisper model
- `AA_whisper_torch_summary.ipynb`: Jupyter notebook with the code for Whisper model with summary
- `lstm-hidden_sizes.png`: Image with the results of the LSTM model with different hidden sizes
- `reduce_dataset.ipynb`: Jupyter notebook with the code copy 2368 files to unknown folder

Folders in main directory:

- `logs/`: Folder with logs for the models
  - contains folders with logs for each model training and testing
- `PDFs/`: Folder with PDFs (requirements, report)
- `plots/`: Folder with plots for the models
  - contains folders with plots for report
- `src/`: Folder with `utils.py` file with utility functions

References

[1] Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. https://arxiv.org/pdf/2006.11477.

[2] Jort F. Gemmeke, Daniel P. W. Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R. Channing Moore, Manoj Plakal, and Marvin Ritter. Audio set: An ontology and human-labeled dataset for audio events. https://research.google/pubs/audio-set-an-ontology-and-human-labeled-dataset-for-audio-events/, 2017.

[3] Yuan Gong, Yu-An Chung, and James Glass. Ast: Audio spectrogram transformer. https://arxiv.org/pdf/2104.01778.

[4] Sepp Hochreiter and Juergen Schmidhuber. Long short-term memory, 1997.

[5] Christopher Olah. Understanding lstm networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[6] Karol J. Piczak. ESC: Dataset for Environmental Sound Classification. http://dl.acm.org/citation.cfm?doid=2733373.2806390, 2015.

[7] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. https://cdn.openai.com/papers/whisper.pdf.

[8] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. https://stanford.edu/ jlmcc/papers/PDP/Volume

[9] Pete Warden. Speech commands dataset. https://www.kaggle.com/c/tensorflow-speech-recognition-challenge/data, 2017.

List of Tables

List of Figures