Speech Commands Recognition using Recursive Neural Networks

Martyna Majchrzak, Władysław Olejnik

25.04.2023

1 Introduction

The aim of this raport is to present the results of a project conducted as part of the Deep Learning course at Warsaw University of Technology. The project included testing multiple Recursive Neural Network architectures on Speech Commands dataset **??**, as a part of the Tensorflow Speech Recognition Challange.

2 Theoretical introduction

Recurrent Neural Networks (RNNs) are a type of neural network designed to process sequential data. They are widely used in natural language processing, speech recognition, time series analysis, and other applications that involve sequences of data. Unlike feedforward neural networks, which process fixed-length inputs, RNNs are able to handle variable-length sequences by maintaining an internal state that is updated as each new element in the sequence is processed.

The basic building block of an RNN is the simple RNN cell. This cell takes as input the current element in the sequence, as well as the internal state from the previous time step, and produces an output and a new internal state. The output is typically fed into a classification or regression layer to produce the final prediction.

Despite their simplicity, simple RNNs can suffer from the vanishing gradient problem, which can make it difficult to train them to process long sequences. To address this issue, several variants of the RNN architecture have been proposed, including Gated Recurrent Units (GRUs) and Long Short-Term Memory (LSTM) networks.

GRUs are similar to simple RNNs, but they include an additional gate that controls the flow of information from the previous internal state to the current internal state.

LSTM networks are another type of RNN that include three gates: an input gate, an output gate, and a forget gate. The input gate controls the flow of information from the current input to the current internal state, while the output gate controls the flow of information from the internal state to the output. The forget gate determines how much of the previous internal state to forget.

A bi-directional Recurrent Neural Network (RNN) is a type of RNN that processes data in both forward and backward directions. This architecture allows the network to take into account both past and future inputs when predicting an output.

3 Dataset and problem describtion

The Speech Commands Data Set v0.01 is a set of audio files, each containing a single spoken English word, from a small set of commands spoken by a variety of different speakers. The data set contains 64,727 audio files, released on August 3rd, 2017, and is designed to help train simple machine learning models. The audio files are organized into folders, with each directory name labeling the word that is spoken them. The core words are "Yes", "No", "Up", "Down", "Left","Right", "On", "Off", "Stop", "Go", "Zero", "One", "Two", "Three", "Four","Five", "Six", "Seven", "Eight", and "Nine". To help distinguish unrecognized words, there are also ten auxiliary words, which most speakers only said once.

4 Data Preprocessing

There are 30 classes of words, but in this task we will focus on recognizing 10 of them that are concidered as speech commands. All the other recordings will be treated as 'Uknown'. There are also recordings marked as background noise, which will be labeled 'Silence', resulting in 12 final classes.

4.1 Balanced training and validation dataset

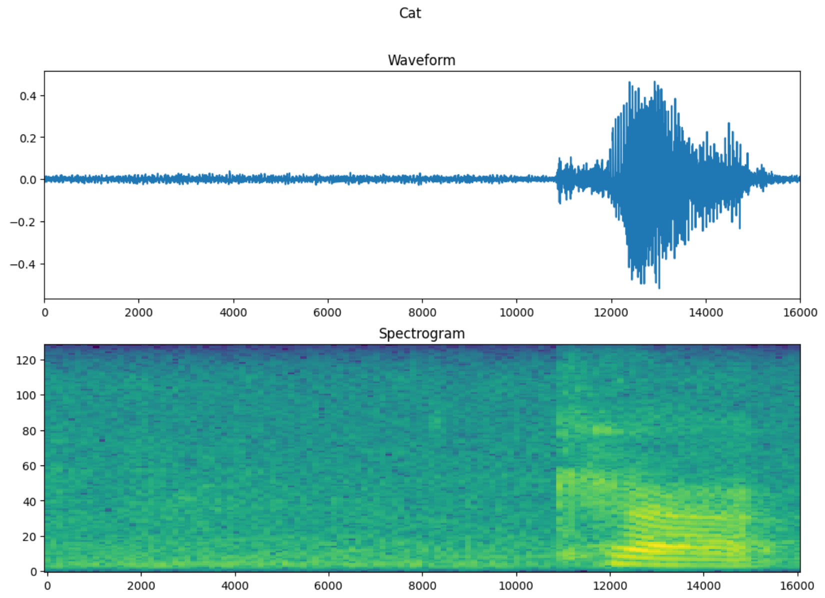
The data provided contained by far the greater number of observations falling into the "unknown" category after processing. To prevent the model from learning to recognize just this class, we tried to create a balanced set in which each of the output 12 classes would be more evenly represented. To do this, we enriched the observations from the target classes with a random selection of 2,000 observations from the "unknown" class and the same number from the "silence" class.

When loading the data, in order to save resources, we were limited to a sample rate of 8kHz. The human voice contains frequencies reaching as high as 20kHz, but the vast majority of the spectrum is occupied by frequencies up to 4kHz. According to Nyquist's theorem, the sampling rate should be at least twice the frequencies we want to capture.

4.2 Creating spectrograms

The raw data are read from the .wav files as waveforms, that are represented in the time domain. We transofrm the waveforms form the time-domain signals into the time-frequency-domain using short-time Fourier transform (STFT) using the libROSA [3] package, convering them to spectrograms. Those spectrograms can be represented as 2D images, which will be fed to the neural networks in the training process.

Figure 1: Waveform and spectrogram representation for the word 'cat'



5 Architectures

During the experiment we tested three recurrent layer types:

1. Simple Recurent Neural Network (SimpleRNN)
2. Gated Recurrent Unit (GRU)
3. Long Short-Term Memory (LSTM)

Each layer type was combined with a couple dense layers at the end of the architecture with increasing level of complexity. First architecture had 2 recurrent layers, followed by 1 Dense layer. In the second architecture, one recurrent and one Dense Layer was added, and in the third one another recurrent layer was added.

The schema of number of layers of each type in the architectures is provided in the tables 1, 2 and 3 below.

architecture	RNN layers	Dense layers
SimpleRNN1	2	1
SimpleRNN2	3	2
SimpleRNN3	4	2

Table 1: SimpleRNN architectures scheme

architecture	GRU layers	Dense layers
GRU1	2	1
GRU2	3	2
GRU3	4	2

Table 2: GRU architectures scheme

architecture	LSTM layers	Dense layers
LSTM1	2	1
LSTM2	3	2
LSTM3	4	2

Table 3: LSTM architectures scheme

The last architecture combines 2 bidirectional LSTM layers with 3 Convolutional layers in an attempt to recognize patterns in the spectrogram images before feeding them to the recurrent neural network. The schema of number of layers in CNN_LSTM1 is provided in the table 4 below.

architecture	CNN layers	Bidirectional LSTM layers	Dense layers
CNN_LSTM1	3	2	3

Table 4: CNN_LSTM1 architecture scheme

All the architectures use batch normalization, dropout after each hidden layer and a softmax activation function.

6 Evaluation Methodology

All models and experiments were created using Tensorflow [1] and the Weights and Biases [2] platform was used to store the parameters of trained models. To train each model, a following base configuration was used:

1. batch_size: 128
2. n_epochs: 50
3. loss: space_categorical_crossentropy
4. optimizer: adam
5. metrics: accuracy, sparse_categorical_accuracy
6. early_stopper
    - monitor: val_sparse_categorical_accuracy,
    - min_delta: 0.001,
    - patience: 4,
    - start_from_epoch: 10,

For each architecture, a combination of the following parameters was checked:

- **seed**: 0,1,2
- **dropout**: 0.3, 0.5

7 Results

Below we present the plots of training loss 2 and training accuracy 3.

Figure 2: Training loss

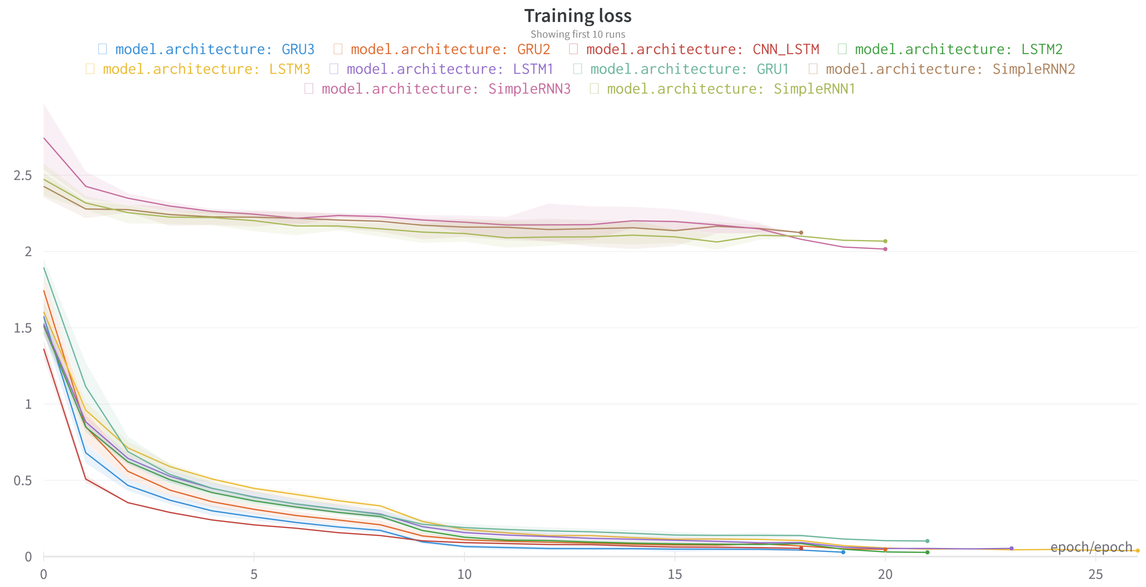
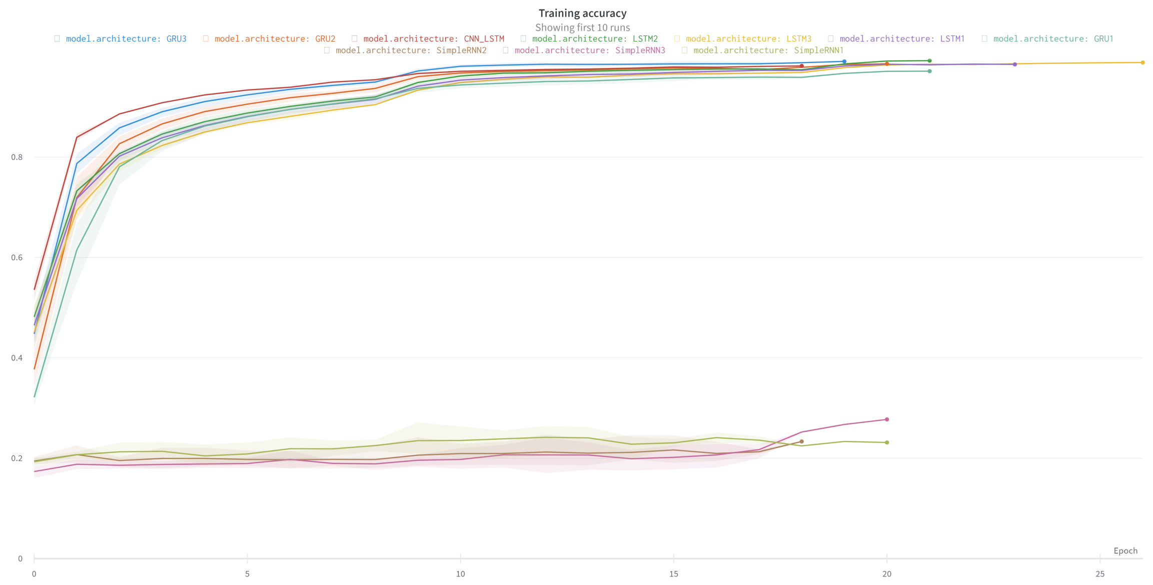


Figure 3: Training accuracy



Among all the configurations, you can see that the models are divided into two groups. Models converging to 80% accuracy quite quickly and models that did not exceed 20% accuracy. It turns out that the weaker models are the SimpleRNN models, while all other GRU, LSTM and CNN+LSTM architects fared much better.

The above can be confirmed by the plots of validation loss 4 and validation accuracy 5.

Figure 4: Validation loss

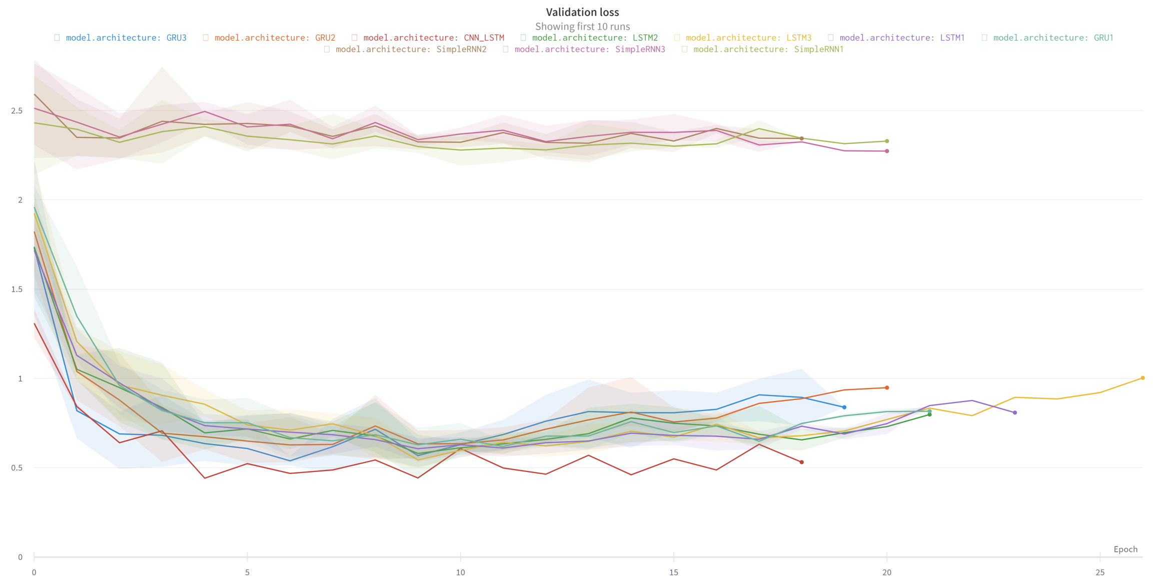
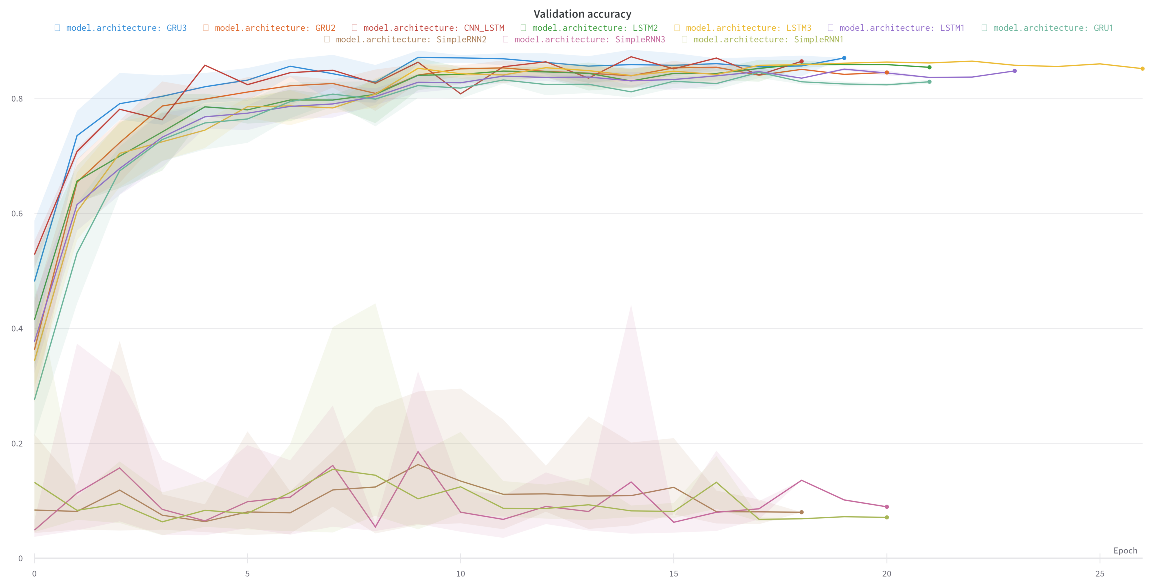


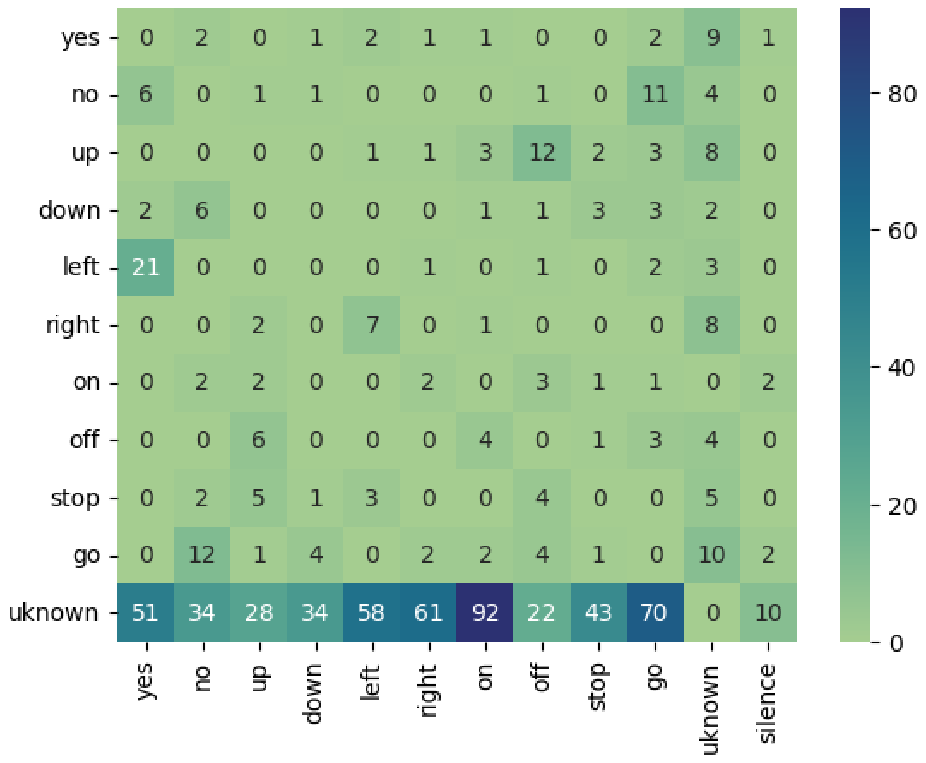
Figure 5: Validation accuracy



Interestingly, you can see that none of the architectures had a specific advantage. Stacking successive recurrent layers did not significantly improve the performance of the models. The dropout value also did not significantly affect the results obtained.

The best architecture turned out to be the GRU3 architecture combined with 0.3 value of dropout. In tests, it achieved an accuracy of over 88%. The confusion matrix is presented in figure 6. However, the values on the diagonal (representing correctly classified observations) were removed to get w clearer view at the words that were classified incorrectly.

Figure 6: Confusion matrix for the GRU3 architecture model with 0.3 dropout



	yes	no	up	down	left	right	on	off	stop	go	uknown	silence
yes	0	2	0	1	2	1	1	0	0	2	9	1
no	6	0	1	1	0	0	0	1	0	11	4	0
up	0	0	0	0	1	1	3	12	2	3	8	0
down	2	6	0	0	0	0	1	1	3	3	2	0
left	21	0	0	0	0	1	0	1	0	2	3	0
right	0	0	2	0	7	0	1	0	0	0	8	0
on	0	2	2	0	0	2	0	3	1	1	0	2
off	0	0	6	0	0	0	4	0	1	3	4	0
stop	0	2	5	1	3	0	0	4	0	0	5	0
go	0	12	1	4	0	2	2	4	1	0	10	2
uknown	51	34	28	34	58	61	92	22	43	70	0	10

Unsuprisingly, the most of those words were in the 'unknown' category, since it is the largest one. Apart from that, the word 'left' got recognized as 'yes' 23 times, the word 'up' got recognized as 'off' 12 times. The words 'go' and 'no' were also commonly confused (11 and 12 times).

7.1 Lack of computing power and further work

From this level, our further goal was to train the best model on the entire training set. The model has already learned the voice commands and extracted as much as possible from the training set constructed by us, as evidenced by over 99% accuracy on the training set. Loading and processing the entire set of over 50,000 files was prevented by the lack of computing power and problems we encountered in the google colab environment.

8 Conclusions

The most important conclusion from this project would be the fact, that stacking multiple recurrent layers does not improve performance significantly - there were no significant difference between types of architectures with the same type of recurrent layer.

However, convolutional layers before recurrent layers may help with proper classification - the CNN+LSTM had a slightly better performance that the LSTM architecture, so it is certainly a subject worth investigating.

It is also worth underlining that this project was the first time working with both Google Colab and TensorFlow for both of the authors. The experience with the latter was especially unpleasant when it came to debugging - we often came across error without clear explanations about what caused then.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com.

[3] Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, volume 8, 2015.