Speech commands classification with recurrent neural networks

Paulina Pacyna, 290600
Mateusz Wójcik, 290639
Mateusz Szysz, 298911

February 18, 2022

1 Preprocessing

First, we loaded the audio files and converted them to an array representing amplitude of the sound time. Then we computed a melspectogram of this sound wave. A melspectogram is an 2d array, where the x-axis corresponds to time, y-axis corresponds to frequencies and z-axis (values) represents the amplitude. Then we applied a logarithmic transformation. Lastly, we normalized the whole dataset by substracting the mean and dividing by the standard deviation. Below we present a sample from our dataset (Figure 1).

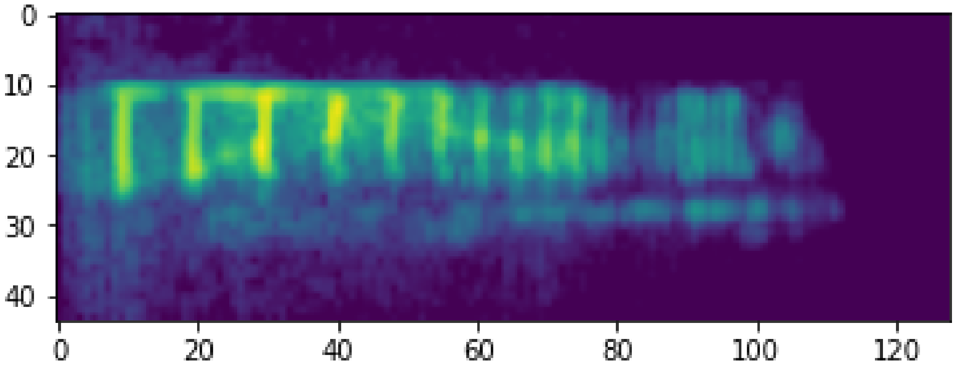


Figure 1: Caption

2 LSTM model

Our first model was based on the LSTM architecture. The model consist of 3 blocks of LSTM and and Dropout layers, preceded with one BatchNormalization layer and followed by Dense layer. The schema is presented below (Figure 8):

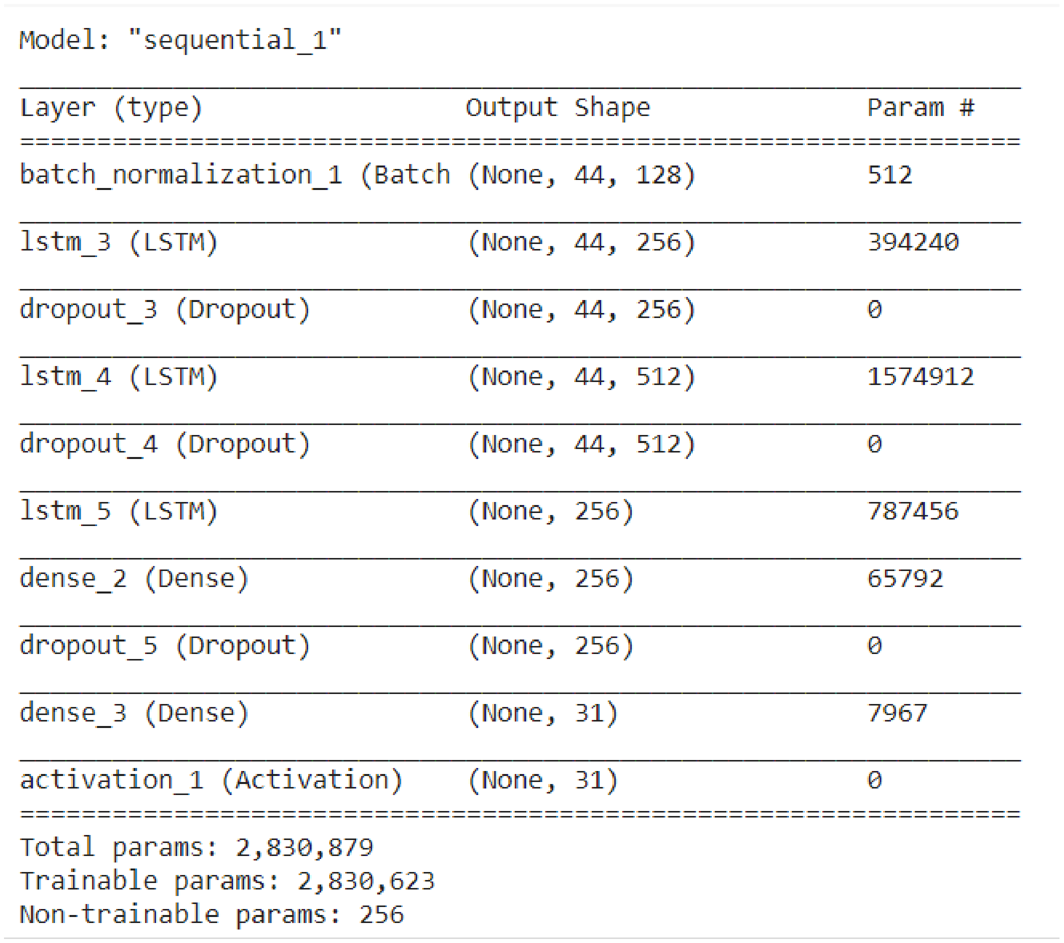

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
batch_normalization_1 (Batch (None, 44, 128)           512
_________________________________________________________________
lstm_3 (LSTM)                (None, 44, 256)           394240
_________________________________________________________________
dropout_3 (Dropout)          (None, 44, 256)           0
_________________________________________________________________
lstm_4 (LSTM)                (None, 44, 512)           1574912
_________________________________________________________________
dropout_4 (Dropout)          (None, 44, 512)           0
_________________________________________________________________
lstm_5 (LSTM)                (None, 256)               787456
_________________________________________________________________
dense_2 (Dense)              (None, 256)               65792
_________________________________________________________________
dropout_5 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 31)                7967
_________________________________________________________________
activation_1 (Activation)    (None, 31)                0
=================================================================
Total params: 2,830,879
Trainable params: 2,830,623
Non-trainable params: 256
```

Figure 2: Summary of our model

This simple model resulted in about 92% accuracy on the test set generated from training, and 60% accuracy on kaggle.
We have generated a confusion matrix. The raw matrix is quite big, so it can't be attached here, please find it attached in the 'LSTM.ipynb' notebook. Below (Figure 8) we present a heatmap which more readable:

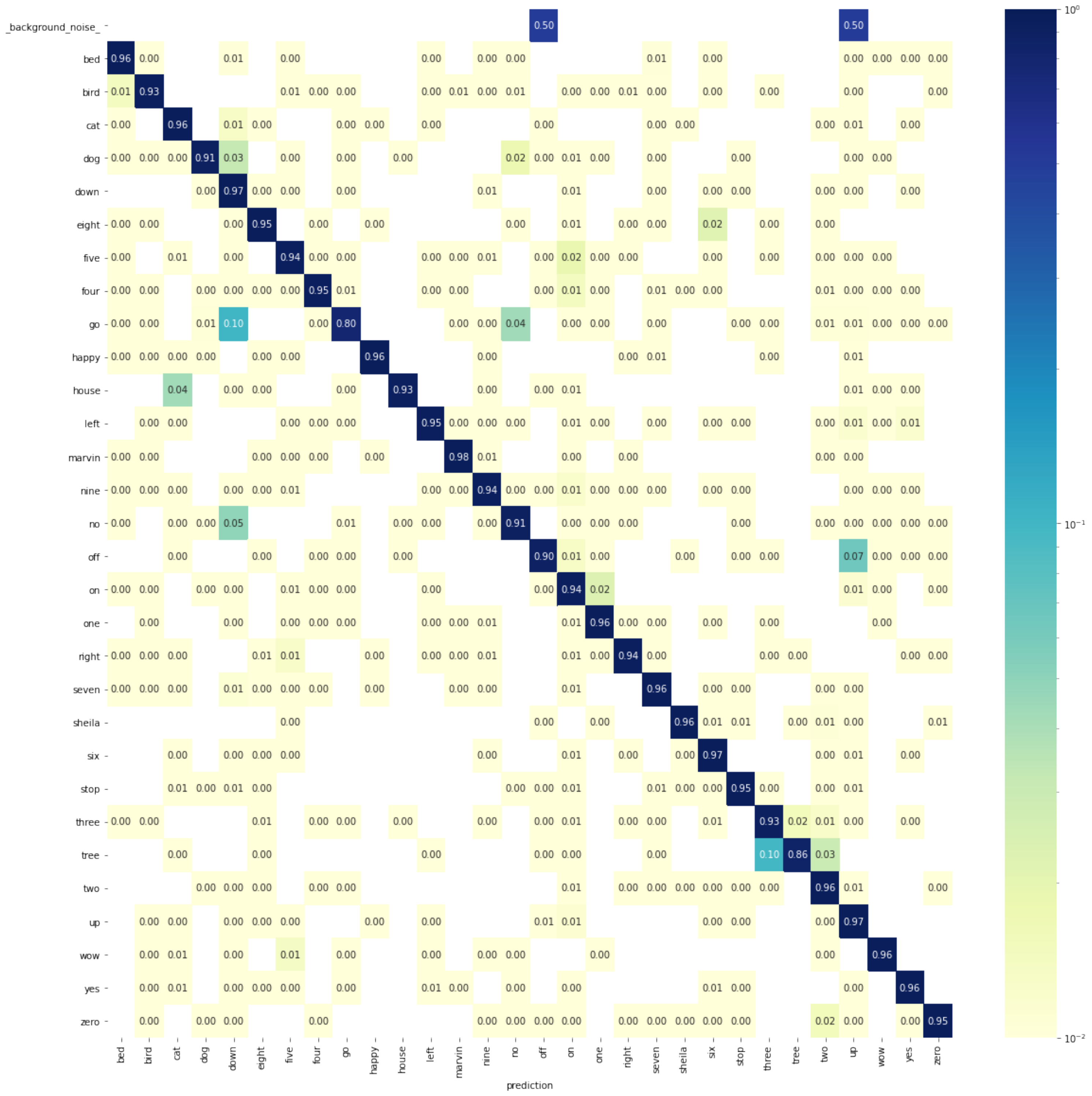


Figure 3: The heatmap for the confusion matrix based on the LSTM model. Blank fields indicates 100% accuracy for this pair.

3 Testing the architecture

We checked how manipulating different parameters impacts the accuracy of the model on the test set generated from the training one.
The model below is our basic architecture similar to the one described in the previous section. In reality it's a little simpler because it has smaller number of units in different layers. However we decided to work on this one due to the fact that time needed to train this model is much smaller. Even though it is simpler, it has very similar accuracy which amounts to 0.915

```
model = Sequential()
model.add(BatchNormalization())
model.add(LSTM(
    128,
    input_shape=X.shape[1:],
    return_sequences=True
))
model.add(Dropout(0.5))
model.add(LSTM(128, return_sequences=True))
model.add(Dropout(0.5))
model.add(LSTM(128))
model.add(Dense(128))
model.add(Dropout(0.5))
model.add(Dense(y.shape[1]))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

Listing 1: Basic model for testing

As can be seen, there are three LSTM layers, one fully connected regular layer and the output one. The default activation function for hidden layers is ReLu. Between all the layers, there are dropouts with probabilities 0.5. As the optimizer we chose Adam's method which is currently the most frequently used one.
To maintain time performance of conducting many tests we chose 30 as the number of epochs and used early stopping.

3.1 Simple RNN instead of LSTM

At the beginning, we wanted to compare LSTM with Simple RNN. To do that we changed all the LSTM layers from the model above into Simple RNN. The decrease of accuracy was dramatic, because we got results between $0.06-0.07$. It's a little more than the results of random classifier (which would get around 0.032), but it's still very far from satisfying. Increasing number of units or layers had a weak positive impact on the accuracy. Unfortunately, the time needed to train these models increased significantly.
I think it's quite convincing example of an enormous difference between LSTM and simple RNN.

3.2 GRU instead of LSTM

Similarly to the previous subsection, we compared LSTM and GRU by changing all the LSTM layers into GRU ones. Due to the fact that GRU is less complex than LSTM, time needed to train these models were significantly smaller. However it turned out that the performance is very similar, because we got the accuracy around 0.925.

3.3 Preventing overfitting

As we indicated previously, we added dropouts to prevent overfitting. We wanted to check how different probability of dropping units impacts the accuracy.

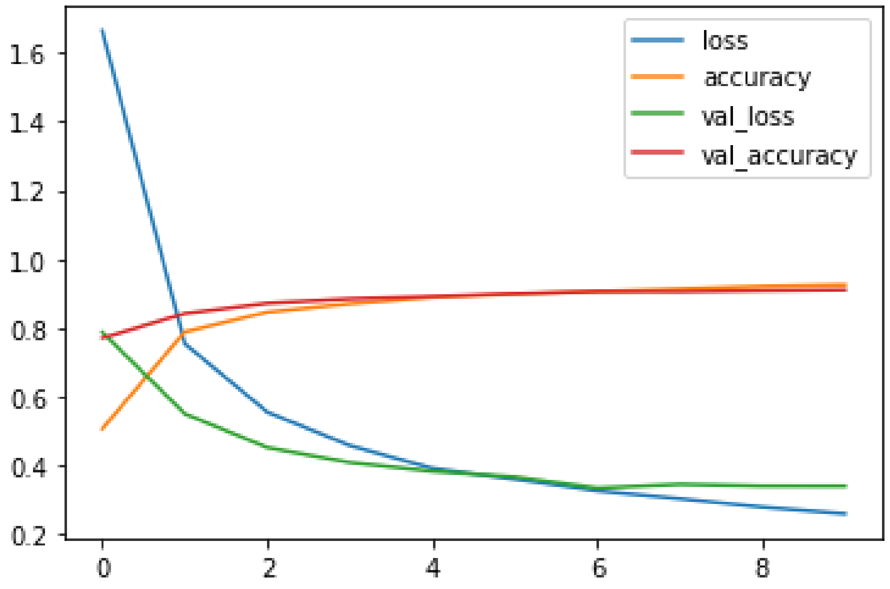


Figure 4: Loss function values and accuracy on the train and validation set for the basic model (with 50% probability of dropout).

As can be seen accuracy on the training and the validation sets are almost identical for majority of time. Similarly, after some epochs loss function values are similar too.
The importance of dropouts are shown in the charts below. For model without dropouts and for the one with dropouts but with smaller probability, the accuracy on the training and validation sets are not so similar as previously.

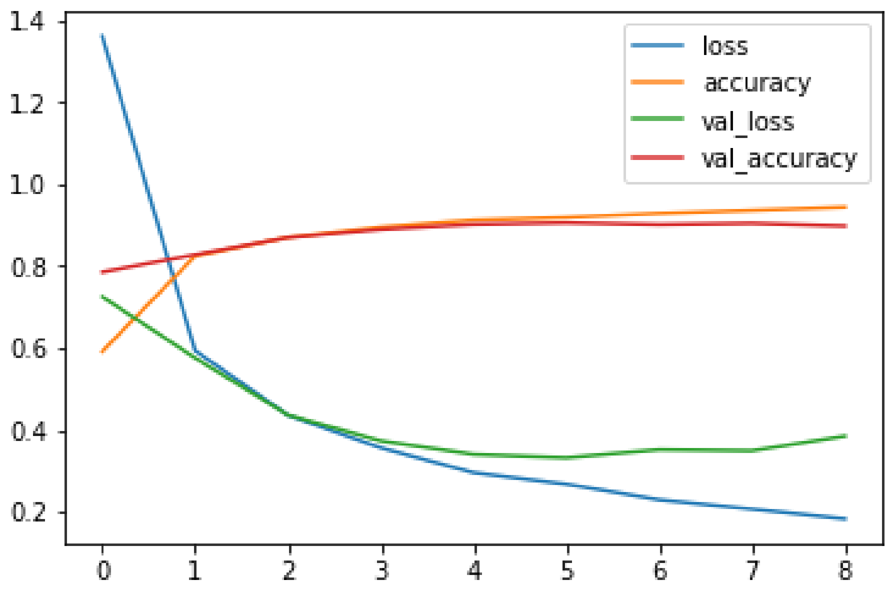


Figure 5: Loss function values and accuracy on the train and validation set on the model without dropouts.

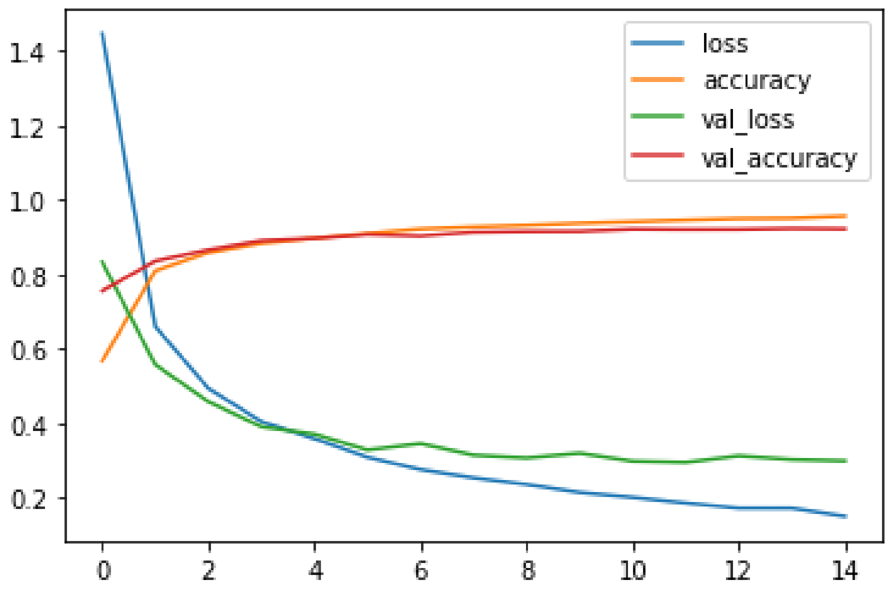


Figure 6: Loss function values and accuracy on the train and validation set on the model with 30% probabilities in dropouts.

In case of the model without dropouts we got the accuracy that amounted to 0.89 and for the one with smaller probability of dropping out units, we got 0.924. The first number indicates that the model lost some accuracy due to overfitting. The other number suggests that using 50% probability may make the model not enough flexible.
Below there is an example where, apart from using dropouts, we added r1-regularization.

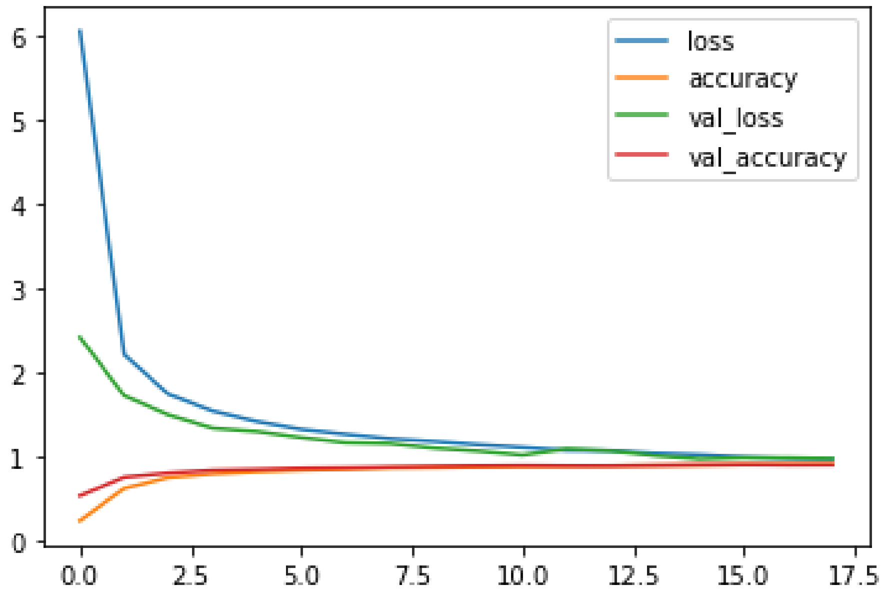


Figure 7: Loss function values and accuracy on the train and validation set for the model with dropouts and r1-regularization.

It's clearly seen that this model is the least flexible. However its performance on the training and the validation sets are almost identical all the time. This time we got very similar accuracy which equals 0.91.
We also tested r2-regularization. The performance was a little worse, but still around 0.89.

3.4 Different sizes of models

We also tested how adding or removing LSTM layers or changing number of units in each of them impacts the performance.
Without surprise it occured that models with smaller number of units in each layers had a little smaller performance which were equal to around 0.89. Adding units had positive impact on the accuracy. In one of the model, we got 0.929.
What is interesting decreasing or increasing number of LSTM layers by one had only an impact on time efficiency of learning but did not impact the accuracy which were equal to 0.92.

3.5 Bidirectional LSTM

Next, we tested bidirectional LSTM. It's an improvement that allows information to flow in two directions - from the past to the future and from the future to the past. It makes the model more complex, but also more flexible. Initially, we changed all the LSTM layers into bidirectional LSTM ones. The learning curves for this model are presented below.

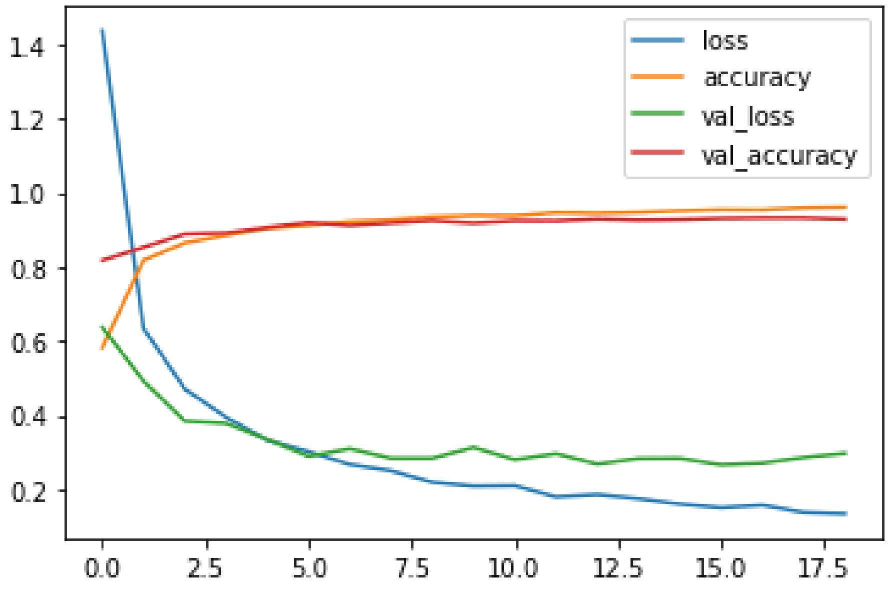


Figure 8: Loss function values and accuracy on the train and validation set for the model bidirectional LSTM.

It can be clearly seen that the curves for the training and the validation sets are not so similar as in the basic model. It's due to the higher flexibility of the model.
This model was the only one in which we got the accuracy slightly above 0.93. Also the learning was much faster than in the case of more complex models with regular LSTM with greater number of layers or units.
Apart from the modifications described in this section, we also added some more and included them in the files attached.

3.6 CNN and Attention Influence

Finally, we tested five different architectures to investigate the influence of adding convolutional layers and using attention to prepare the model. This approach was mostly based on the article *A neural attention model for speech command recognition* by Douglas Coimbra de Andradea, Sabato Leob, Martin Loesener Da Silva Vianac, and Christoph Bernkopfc.

In these cases, we again used mel-scale spectrogram computed using 80-band mel scale, 1024 discrete Fourier transform points and hop size of 128 points. After that, a set of convolutions is applied to the spectrogram in the time dimension to extract local and short-term relations in the audio file. Then, we applied a set of bidirectional LSTM or GRU units to capture two-way long term dependancies. Next, one of the output vectors of the last LSTM layer is extracted and projected using a dense layer to identify which part is the most relevant. To do so, we choose the middle vector of the LSTM output, because the voice command was asssumed to be centered in the files. Finally, we calculate the weighted average of the LSTM output and feed it into fully connected layers for classification.

Attention gives the possibility to explain the results and get the intuition about the model. For example, plotting the attention weights allows to visualise which parts of the audio were most relevant for the model and its classification.

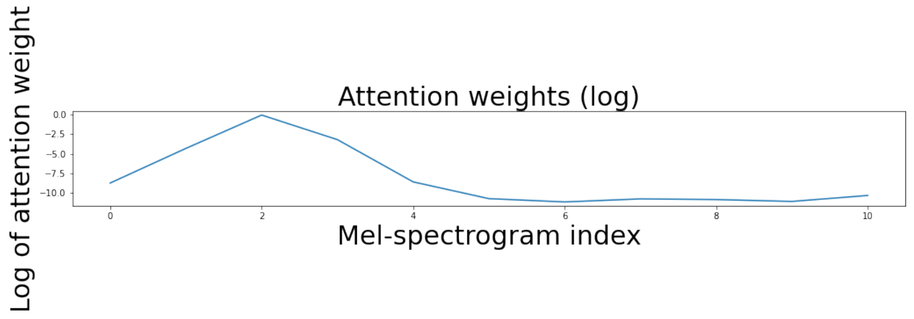


Figure 9: Example Attention Weights

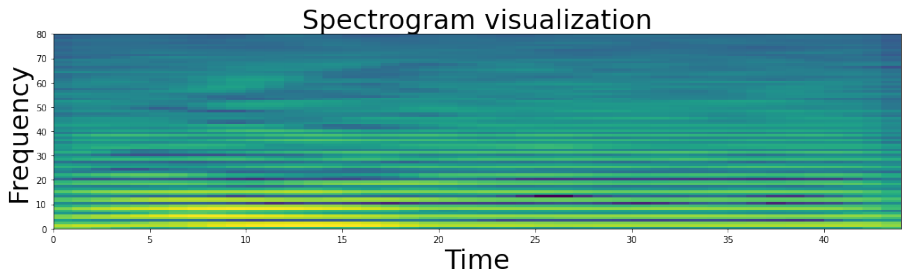


Figure 10: Example Spectrogram Visualisation

As we mentioned before, five different architectures were considered in the tests:

- Model 1 - consists of the most advanced CNN part: 3 layers with 20, 40, and 1 filters and (5, 1) and (3, 3) kernels. Then, two bidirectional LSTM layers with 64 units and finally, the attention part.
- Model 2 - more advanced LSTM part: 2 convolutional layers with 10 and 1 filters and (5, 1) kernel, 4 bidirectional LSTM layers with (128, 128, 64, 64) units, attention part.
- Model 3 - the same as Model 2, except for the lack of the attention part.
- Model 4 - the same as Model 3, except for the switching from LSTM to GRU units.
- Model 5 - Model 4 with attention part.

For all of the above models we used Adam optimizer, two metrics: *loss* and *categorical sparse accuracy*. As a learning rate scheduler, we used simple drop-based learning scheduler with initial learning of 0.001 and drop 0.4 (with the power based on the epoch). We calculated at most 50 epochs with the batch size of 128.

The accuracies on test and training data sets (based on the split of Kaggle training data) are presented as follows.

Model	Measure	
	Train Accuracy	Test Accuracy
1	0.946	0.871
2	0.961	**0.957**
3	0.958	0.891
4	0.958	**0.960**
5	0.970	0.918

Table 1: Training and Test Accuracies Based On Split Kaggle Training Set (80-20)

We can see that CNN+LSTM model with attention (Model 2) mechanism and CNN+GRU model without attention (Model 4) obtained the highest scores on test set.

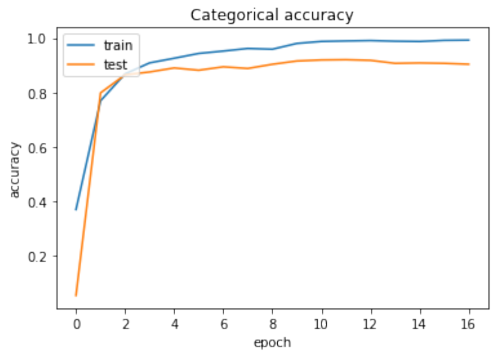


Figure 11: Model 2 - Accuracy on Training and Validation Set

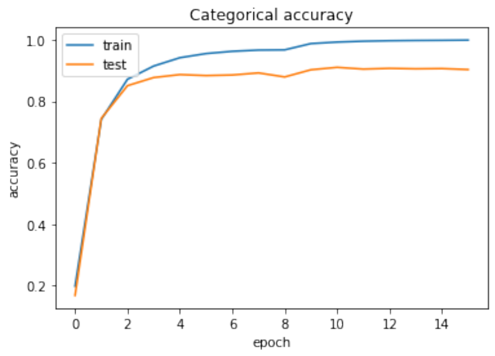


Figure 12: Model 4 - Accuracy on Training and Validation Set

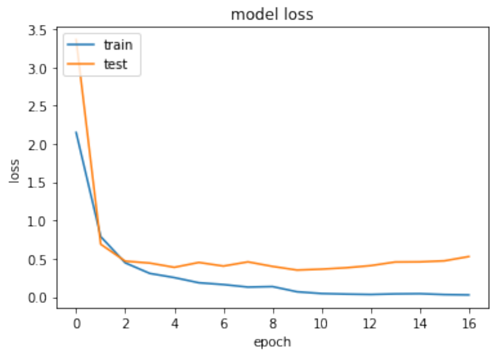


Figure 13: Model 2 - Crossentropy Loss on Training and Validation Set

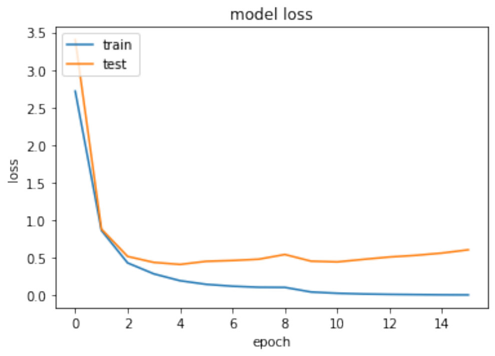
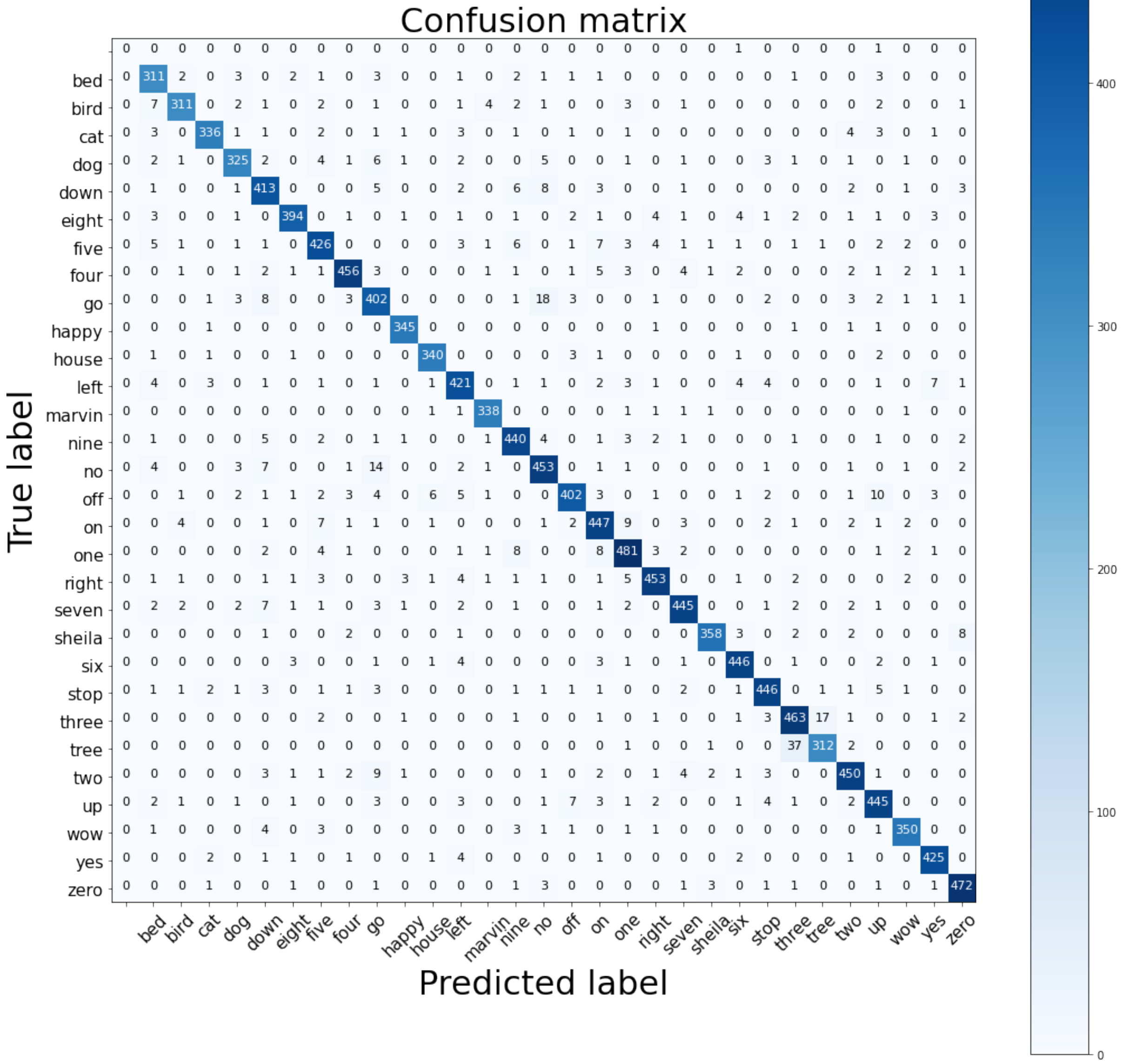


Figure 14: Model 4 - Crossentropy Loss on Training and Validation Set

We also prepared confusion matrix for all of these five models. Let's have a look at the confusion matrix of Model 1 (more advanced CNN + LSTM + attention mechnism).



		bed	bird	cat	dog	down	eight	five	four	go	happy	house	left	marvin	nine	no	off	on	one	right	seven	sheila	six	stop	three	tree	two	up	wow	yes	zero
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0
bed	0	311	2	0	3	0	2	1	0	3	0	0	1	0	2	1	1	1	0	0	0	0	0	0	1	0	0	3	0	0	0
bird	0	7	311	0	2	1	0	2	0	1	0	0	1	4	2	1	0	0	3	0	1	0	0	0	0	0	0	2	0	0	1
cat	0	3	0	336	1	1	0	2	0	1	1	0	3	0	1	0	1	0	1	0	0	0	0	0	0	0	4	3	0	1	0
dog	0	2	1	0	325	2	0	4	1	6	1	0	2	0	0	5	0	0	1	0	1	0	0	3	1	0	1	0	1	0	0
down	0	1	0	0	1	413	0	0	0	5	0	0	2	0	6	8	0	3	0	0	1	0	0	0	0	0	2	0	1	0	3
eight	0	3	0	0	1	0	394	0	1	0	1	0	1	0	1	0	2	1	0	4	1	0	4	1	2	0	1	1	0	3	0
five	0	5	1	0	1	1	0	426	0	0	0	0	3	1	6	0	1	7	3	4	1	1	1	0	1	1	0	2	2	0	0
four	0	0	1	0	1	2	1	1	456	3	0	0	0	1	1	0	1	5	3	0	4	1	2	0	0	0	2	1	2	1	1
go	0	0	0	1	3	8	0	0	3	402	0	0	0	0	1	18	3	0	0	1	0	0	0	2	0	0	3	2	1	1	1
happy	0	0	0	1	0	0	0	0	0	0	345	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	1	0	0	0
house	0	1	0	1	0	0	1	0	0	0	0	340	0	0	0	0	3	1	0	0	0	0	1	0	0	0	0	2	0	0	0
left	0	4	0	3	0	1	0	1	0	1	0	1	421	0	1	1	0	2	3	1	0	0	4	4	0	0	0	1	0	7	1
marvin	0	0	0	0	0	0	0	0	0	0	0	1	1	338	0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0	0
nine	0	1	0	0	0	5	0	2	0	1	1	0	0	1	440	4	0	1	3	2	1	0	0	0	1	0	0	1	0	0	2
no	0	4	0	0	3	7	0	0	1	14	0	0	2	1	0	453	0	1	1	0	0	0	0	1	0	0	1	0	1	0	2
off	0	0	1	0	2	1	1	2	3	4	0	6	5	1	0	0	402	3	0	1	0	0	1	2	0	0	1	10	0	3	0
on	0	0	4	0	0	1	0	7	1	1	0	1	0	0	0	1	2	447	9	0	3	0	0	2	1	0	2	1	2	0	0
one	0	0	0	0	0	2	0	4	1	0	0	0	1	1	8	0	0	8	481	3	2	0	0	0	0	0	0	1	2	1	0
right	0	1	1	0	0	1	1	3	0	0	3	1	4	1	1	1	0	1	5	453	0	0	1	0	2	0	0	0	2	0	0
seven	0	2	2	0	2	7	1	1	0	3	1	0	2	0	1	0	0	1	2	0	445	0	0	1	2	0	2	1	0	0	0
sheila	0	0	0	0	0	1	0	0	2	0	0	0	1	0	0	0	0	0	0	0	0	358	3	0	2	0	2	0	0	0	8
six	0	0	0	0	0	0	3	0	0	1	0	1	4	0	0	0	0	3	1	0	1	0	446	0	1	0	0	2	0	1	0
stop	0	1	1	2	1	3	0	1	1	3	0	0	0	0	1	1	1	1	0	0	2	0	1	446	0	1	1	5	1	0	0
three	0	0	0	0	0	0	0	2	0	0	1	0	0	0	1	0	0	1	0	1	0	0	1	3	463	17	1	0	0	1	2
tree	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	37	312	2	0	0	0	0
two	0	0	0	0	0	3	1	1	2	9	1	0	0	0	0	1	0	2	0	1	4	2	1	3	0	0	450	1	0	0	0
up	0	2	1	0	1	0	1	0	0	3	0	0	3	0	0	1	7	3	1	2	0	0	1	4	1	0	2	445	0	0	0
wow	0	1	0	0	0	4	0	3	0	0	0	0	0	0	3	1	1	0	1	1	0	0	0	0	0	0	0	1	350	0	0
yes	0	0	0	2	0	1	1	0	1	0	0	1	4	0	0	0	0	1	0	0	0	0	2	0	0	0	1	0	0	425	0
zero	0	0	0	1	0	0	1	0	0	1	0	0	0	0	1	3	0	0	0	0	1	3	0	1	1	0	0	1	0	1	472

Figure 15: Confusion Matrix on Test Set of Model 1

Note that for Model 1 (and not only Model 1) it was difficult to distinguish between words *tree* and *three*. It is quite reasonable because even for human this task may be difficult depending on the accent for example. There are more pairs that are less likely to be classified properly, for instance: *off* and *up*, *no* and *go*.

To summarise, we have noted that GRU and LSTM units present quite similar results, whereas GRU ones are less computationally expensive, training time was shorter and there were less parameters to optimize. We have also noticed that for LSTM cells attention mechanism increased the accuracy of the model. Nonetheless, for GRU cells it was not the case. Adding CNN layers to the model increased the accuracies, but making them more complicated did not result in better performance.

4 External sources

- www.machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/
- K. Lekshmi, E. Sherley, Automatic Speech Recognition using different Neural Network Architectures
- A neural attention model for speech command recognition