Deep Learning

Final report

Project2 - Recurrent Neural Networks

Authors Jan Wojtas
Mikolaj Zalewski

24th April 2023

Table of Contents

1 Project description

The project below is an adaptation of the TensorFlow Speech Recognition Challenge task from the Kaggle platform. The goal of this project is to test out various recurrent neural network architectures and try to achieve the best score possible.

The TensorFlow Speech Recognition Challenge aimed to develop a machine learning model capable of accurately recognizing spoken words in short audio clips. This task is challenging due to the complex and variable nature of speech signals, which contain a wide range of acoustic features that can vary greatly depending on the speaker, environment, and context.

Participants in the challenge were required to design and train deep learning models that could effectively extract the relevant features from the audio signals and use them to classify the spoken words. The objective was to create a system that could generalize well to new audio samples and accurately recognize words spoken by a diverse range of speakers in various environments.

The ultimate goal of the challenge was to advance the state of the art in speech recognition and enable new applications and use cases, such as virtual assistants, voice-controlled devices, and automatic transcription.

2 Theoretical introduction

Recurrent Neural Networks (RNNs) are a type of neural network architecture that is designed to work with sequential data, such as time-series or natural language. Unlike traditional feedforward neural networks, RNNs have the ability to maintain a memory of past inputs and use this memory to inform the processing of future inputs. This memory is maintained through a loop in the network, where the output of each time step is fed back into the network as an input to the next time step.

However, traditional RNNs are prone to the problem of vanishing or exploding gradients, which can make training difficult. This occurs because the gradient signal from the output of the network has to pass through multiple layers and time steps, and it can either become too small (vanish) or too large (explode) to be useful for training. To address this issue, several types of RNNs have been developed, including Long Short-Term Memory networks (LSTMs), Gated Recurrent Units (GRUs), and Transformers.

LSTMs and GRUs are both designed to address the vanishing gradient problem by introducing gating mechanisms that allow the network to remember or forget information from previous time steps. LSTMs achieve this through three gates: an input gate, an output gate, and a forget gate. GRUs simplify this by using just two gates: a reset gate and an update gate.

Transformers, on the other hand, do not use a recurrent loop like LSTMs and GRUs, but instead use an attention mechanism to capture dependencies between different parts of the input sequence and focus on relevant parts on input.

In the project, we explore all three of these architectures and measure their performance on the Speech Command Dataset.

3 Project environment

1. Chosen programming language: Python.

2. This project (unlike the first one) is based on Tensorflow-powerful Python framework for Deep Learning.

3. We use GitHub repository for file hosting and versioning our project. Link to therepository can be found here: https://github.com/mikolajzalewski/Deep_Learning. The repository is di-

vided into subdirectories, each devoted to separate Deep Learning project. More information about the project's modules and their utility can be found in the ReadMe (in root directory, as well as in **RNN** directory).

4 Speech commands dataset

4.1 Overview

The training dataset consists of over 60,000 audio files of spoken English words, each of which is one second in length. The files are organized into 30 different one-word classes. Words belonging to ['yes', 'no', 'up', 'down', 'left', 'right', 'on', 'off', 'stop', 'go'] are considered as known labels. The rest of labels is supposed to be represented as 'unknown'. The testing dataset consists of around 150,000 clips and, apart from the labels, contains clips of 'silence', which also should be detected.

4.2 Data preprocessing

Prior to training various models on the data, we decided to perform initial pre-processing, suggested in [4]. The pre-processing part consisted of 3 stages: voice activity detection, padding and resampling.

4.2.1 Voice activity detection (VAD)

The goal of VAD is to identify the regions of an audio signal that contain speech and separate them from regions that contain background noise or silence. VAD algorithms typically work by analyzing the audio signal and looking for patterns that are characteristic of speech, such as changes in the frequency or amplitude of the signal.

We used VAD to extract only relevant part of each clip. Furthermore, it allowed us to detect clips containing only silence, as in case of such clips the VAD detector would not return any audio frames on the output.

4.2.2 Padding

After performing VAD, most of the videos were shortened. Therefore, we add padding with zeros up to 1 second, to ensure that each audio clip has the same length.

4.2.3 Resampling

At last, as suggested in [4], we perform resampling of the audio clips. The sampling rate of an audio signal refers to the number of samples that are taken per second to represent the signal. We reduce the sampling rate from 16 kHz to 8 kHz of each file and therefore reduce the dimensionality and complexity of data, without significant data loss.

4.3 Feature extraction

After data preprocessing, it is time to extract relevant features out of each audio clip. As suggested in [2], we extract 13 MFCCs, 13 delta coefficients and 13 delta-delta coefficients.

4.3.1 MFCCs

Mel Frequency Cepstral Coefficients (MFCCs) are a representation of the short-term power spectrum of an audio signal, which can be used to capture important characteristics of the signal, such as the pitch and the timbre. The MFCCs extraction involves a series of steps, including windowing, discrete Fourier transform, Mel-scale filtering, logarithm, discrete cosine transform and normalization. MFCCs are often used as features for speech recognition and other audio processing tasks as they are relatively robust to noise and can capture important aspects of the signal.

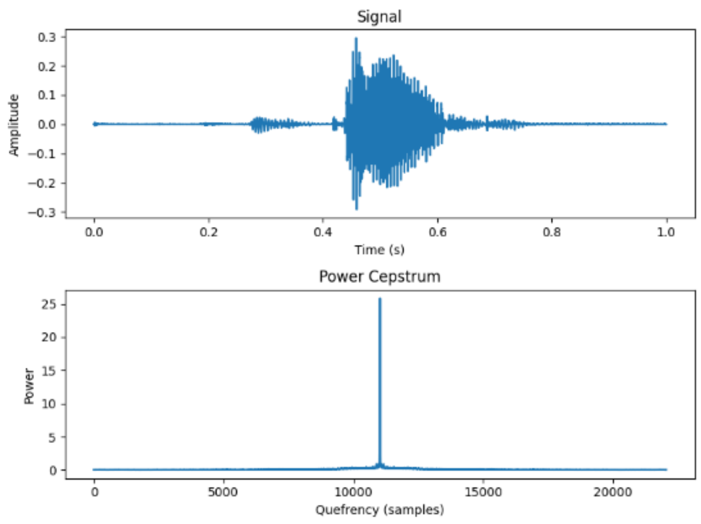


Figure 1: Signal and power cepstrum of spoken word "BED". MFCCs are later derived from power cepstrum

4.3.2 Delta and delta-delta coefficients

Delta and delta-delta coefficients are features derived from the MFCCs.

Delta coefficients represent the rate of change of the MFCCs over time, while delta-delta coefficients represent the acceleration of the MFCCs over time (in some sense they are equivalents of the 1st and 2nd derivatives of MFCCs). They are used to capture the dynamic variations in the speech signal that are not captured by the static MFCCs.

5 Models

In this section we review the architecture of 3 models, which are analysed in this project: LSTM, GRU, and Transformer.

5.1 LSTM

Long Short-Term Memory (LSTM) network is a type of recurrent neural network (RNN) that is designed to handle sequential data. LSTMs are commonly used in natural language processing (NLP), speech recognition, and other applications that involve processing sequences of data.

The structure of our specific LSTM architecture consists of several layers that are designed to process sequential data in a specific way. The first layer is a **Bidirectional** LSTM layer that processes the input sequences in both directions, allowing the model to capture context from both past and future inputs. The next few layers are **BatchNormalization** and **Dropout** layers, which help to prevent overfitting and improve the generalization of the model.

The model then contains another Bidirectional LSTM layer followed by BatchNormalization and Dropout layers. This is followed by a single LSTM layer that returns sequences, and several **Time Distributed Dense** layers that apply a dense layer to each time step of the sequence. The model ends with another LSTM layer, a BatchNormalization layer, a Dropout layer, and a Dense layer with a softmax activation function that produces the final output.

5.2 GRU

Gated Recurrent Unit (GRU) is another type of recurrent neural network (RNN) layer that can handle sequential data with long-term dependencies. It is similar to LSTM layer, where we observe slightly different gating mechanism.

Our model architecture starts with a **1D convolutional** layer with 128 filters and a kernel size of 3, followed by **batch normalization** and **max pooling** layers. Then, there are three **GRU** layers, the first two of which are bidirectional and the last one is unidirectional. Each GRU layer is followed by batch normalization and dropout layers. The output of the last GRU layer is passed through three **time-distributed dense** layers, each followed by batch normalization and dropout layers. Then, there is a global max pooling layer and three fully connected dense layers, each followed by batch normalization and dropout layers. The output layer has a **softmax** activation function and predicts one of the classes.

5.3 Transformer

The last architecture is a type of neural network model called a Transformer, which is commonly used for sequence-to-sequence tasks such as machine translation and text generation.

The model consists of a series of **Transformer blocks**, which are a specific type of layer that includes self-attention and feedforward neural network layers. Each Transformer block takes as input a sequence of embeddings and applies self-attention to compute a weighted sum of the embeddings, followed by a feedforward neural network layer. The outputs of the Transformer blocks are then combined and passed through several **dense** layers with **batch normalization** and **dropout** for classification.

The specific layers used in this model are:

1. Input: a Keras Input layer that defines the shape of the input tensor.
2. Conv1D: a 1D convolutional layer that applies a set of filters to the input sequence to extract higher-level features.
3. BatchNormalization: a layer that normalizes the output of the previous layer to help with training stability.
4. MaxPooling1D: a pooling layer that reduces the size of the output tensor by taking the maximum value in each window of a given size.
5. Dropout: a regularization layer that randomly sets a fraction of the inputs to zero during training to prevent overfitting.
6. TransformerBlock: a custom layer that includes a Multi-Head Attention layer and a feedforward neural network layer.
7. GlobalMaxPooling1D: a pooling layer that reduces the output tensor to a single vector by taking the maximum value across the entire sequence.
8. Dense: a fully connected layer that applies a linear transformation to the input vector followed by an activation function (ReLU or softmax).
9. Softmax: an activation function that normalizes the output of the previous layer to produce a probability distribution over the possible classes.

Overall, all 3 models seem to be well-suited for processing sequential data such as audio signals, as they can effectively capture context and dependencies between input sequences. The various layers used in the models are carefully chosen to prevent overfitting and improve the generalization, which is crucial for achieving high accuracy on complex tasks like speech recognition.

6 Methodology and approach

The TensorFlow Speech Recognition Challenge required the creation of a model to classify audio recordings into one of ten known classes, as well as a label for any unknown sounds. Additionally, the test set contained both known and unknown labels, as well as words not seen in the training or validation sets.

Overall, the classification task was difficult and therefore, we tested out different methods and approaches to identify the best classification strategy.

6.1 Single, label detection model

Our first approach was to create a single model that predicted each of the ten known labels as well as the "unknown" label, and evaluate its performance on the validation set. We performed various hyper-parameter searches, including **grid search** and **random search** and achieved about **80-85 % accuracy** on best models.

6.2 Silence + label detection models

Next, we tried a model based on LSTM that would predict silence, allowing us to classify the test set into known labels and unknown sounds. The model was evaluated on validation dataset and gave great results of almost 100% accuracy

We combined both models into one, with the first model predicting if the file contains silence, and the second model predicting the label. However, while this approach showed high accuracy on the validation set, it performed poorly on Kaggle.

6.3 VAD + label detection model

We discovered that the model was incorrectly identifying many files as not containing silence when in fact they did. We suppose that our silence model was overfitted to silence training data (gathered from _background_noise_ directory from train folder), but the test set contained other forms of silence, which could not be detected by our model.

To address this, we changed the approach and utilized Voice Activity Detection (VAD) to detect significant moments of speech (instead of separate model for silence detection), and identify whole files as silence if no significant moments were found.

With this new approach, the silence detection achieved an accuracy of 90%, which formed a good foundation for further improvements.

6.4 VAD + known vs unknown detection + label detection

We continued by building more advanced versions of the model, training it to recognize whether a given label belonged to the known set (['yes', 'no', 'up', 'down', 'left', 'right', 'on', 'off', 'stop', 'go']) or not. This led us to a model with three layers - detecting silence, detecting unknown labels, and assigning known labels to observations.

Finally, we found that our first approach - detecting silence and then the label in comparison with second approach - detecting silence, whether the label is known or not, and then predicting the label, was more effective.

At last, We built an ensemble model based on the best model out of all tested models. We evaluated the ensemble's performance based on **mean** and **max** predict methods. This model achieved the best results overall, and we were satisfied with the final outcome of our project.

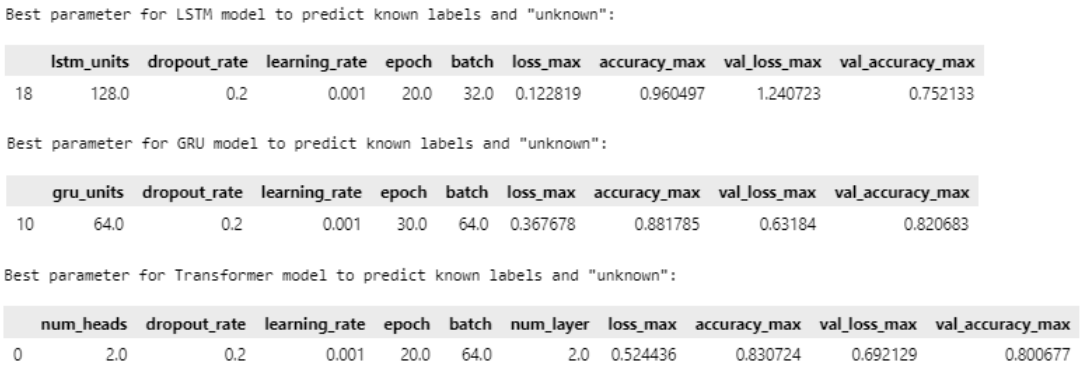
7 Experiment results

In this section discuss results of the previously mentioned experiments.

7.1 Training results

First of all we review the results of hyperparameter tuning and discovering optimal neural network architectures. we show the results for single **label** detection model, **silence** detection model and **known vs unknown + label** detection model.

7.1.1 Single, label detection model - Grid search



Best parameter for LSTM model to predict known labels and "unknown":

	lstm_units	dropout_rate	learning_rate	epoch	batch	loss_max	accuracy_max	val_loss_max	val_accuracy_max
18	128.0	0.2	0.001	20.0	32.0	0.122819	0.960497	1.240723	0.752133

Best parameter for GRU model to predict known labels and "unknown":

	gru_units	dropout_rate	learning_rate	epoch	batch	loss_max	accuracy_max	val_loss_max	val_accuracy_max
10	64.0	0.2	0.001	30.0	64.0	0.367678	0.881785	0.63184	0.820683

Best parameter for Transformer model to predict known labels and "unknown":

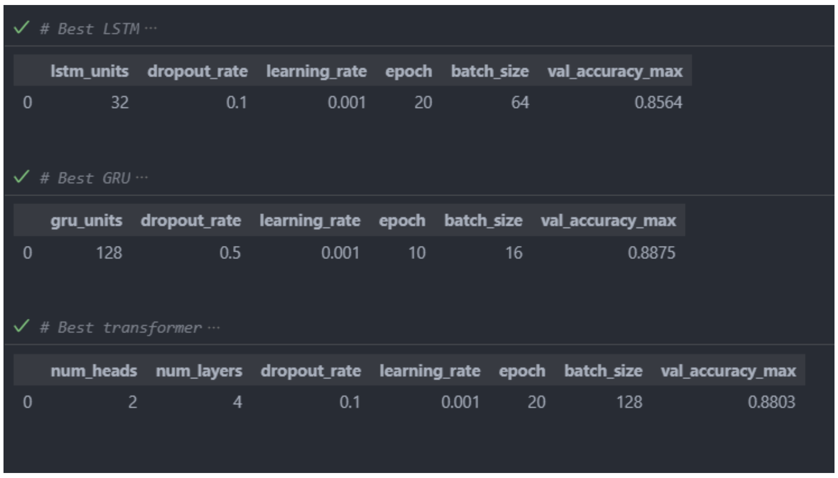
	num_heads	dropout_rate	learning_rate	epoch	batch	num_layer	loss_max	accuracy_max	val_loss_max	val_accuracy_max
0	2.0	0.2	0.001	20.0	64.0	2.0	0.524436	0.830724	0.692129	0.800677

Figure 2: Chosen parameters for specific models

1. **lstm units** - hyperparameter defining the number of units in the LSTM layer of a neural network model. It determines the size of the hidden state of each LSTM unit, which controls the capacity and complexity of the model.
2. **gru units** - similairly defines the number of units in the GRU layer of a neural network model.
3. **num heads** - refers to the number of parallel self-attention heads in the multi-head self-attention sublayer of the TransformerBlock. Increasing the number of heads can increase the model's capacity to model complex relationships within the input sequence.
4. **num layers** - refers to the number of TransformerBlock layers in the network. Increasing the number of layers can also increase the model's capacity to learn complex relationships within the input sequence, but at the cost of increased computational complexity and the potential for overfitting on the training data.

7.1.2 Single, label detection model - Random Search

We also tried to perform Random Search on the space of hyperparameters to find the best model for label vs unknown detection.



```
# Best LSTM ···
```

	lstm_units	dropout_rate	learning_rate	epoch	batch_size	val_accuracy_max
0	32	0.1	0.001	20	64	0.8564

```
# Best GRU ···
```

	gru_units	dropout_rate	learning_rate	epoch	batch_size	val_accuracy_max
0	128	0.5	0.001	10	16	0.8875

```
# Best transformer ···
```

	num_heads	num_layers	dropout_rate	learning_rate	epoch	batch_size	val_accuracy_max
0	2	4	0.1	0.001	20	128	0.8803

Figure 3: Best models from random search

On the figures below we present training plots for best performing models (with regard to validation accuracy). In most cases the training process is smooth and does not lead to major overfitting on training dataset:

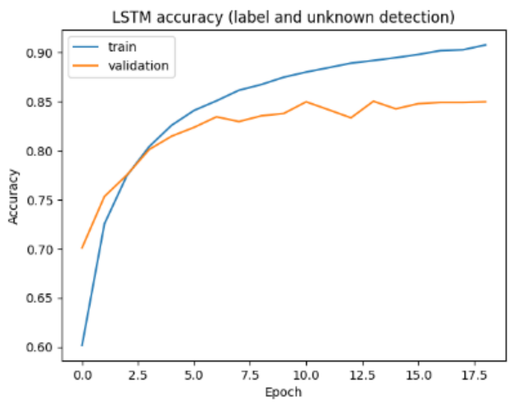


Figure 4: LSTM Accuracy

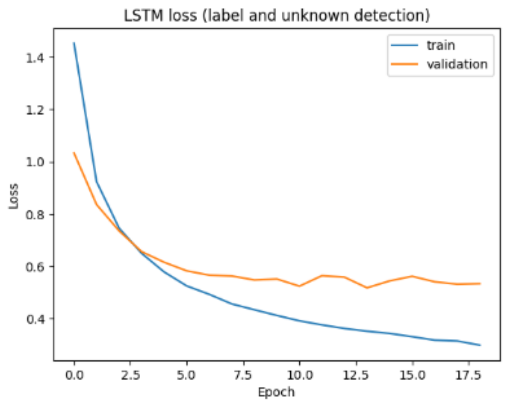


Figure 5: LSTM Loss

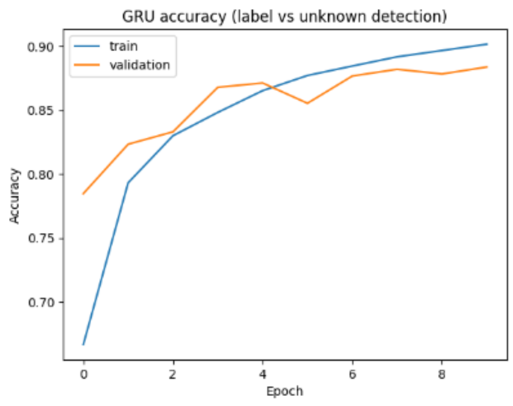


Figure 6: GRU Accuracy

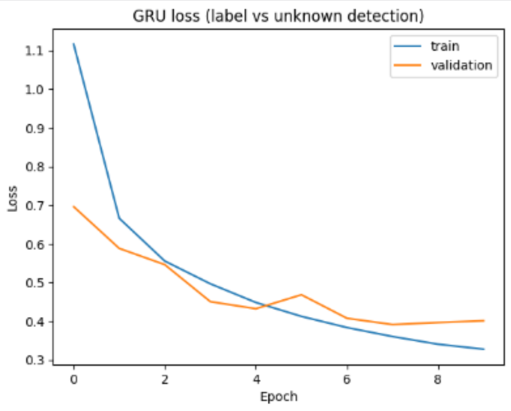


Figure 7: GRU Loss

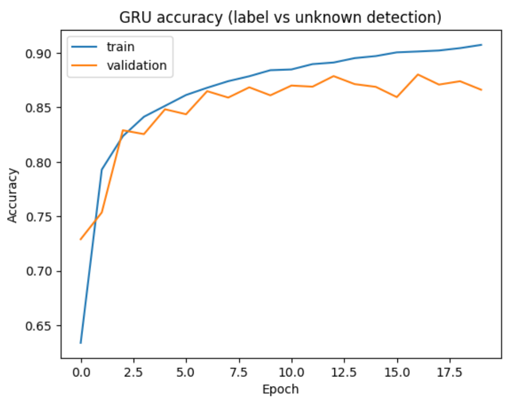


Figure 8: Transformer Accuracy

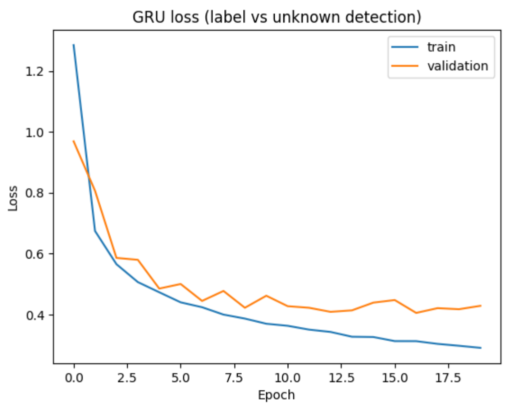
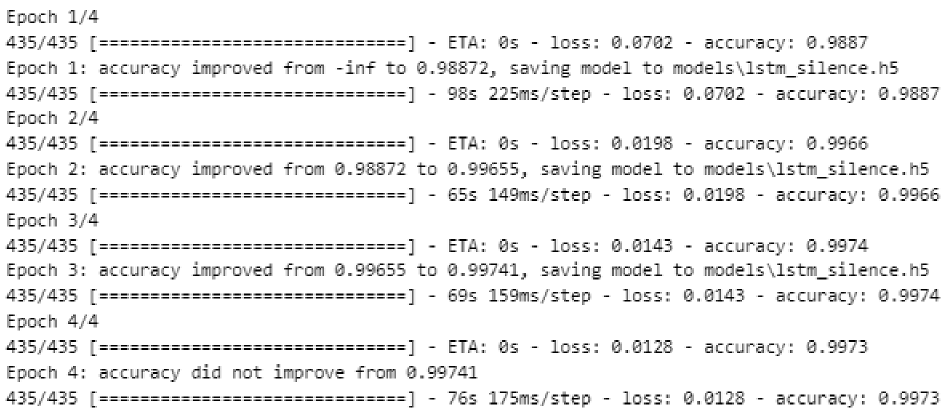


Figure 9: Transformer Loss

7.1.3 Silence model

Next, let's focus on the silence prediction model. We started with the LSTM, but the results were so good that we decided to stop at first. Probably due to well-conducted preprocessing, detecting silence was a fairly easy task on the validation set. Unfortunately, it did not yield good results on test dataset and we decided to switch to silence detection with VAD.

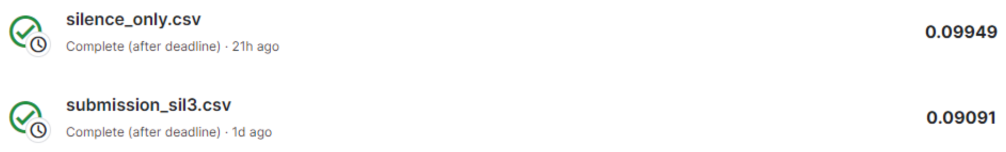


```
Epoch 1/4
435/435 [==============================] - ETA: 0s - loss: 0.0702 - accuracy: 0.9887
Epoch 1: accuracy improved from -inf to 0.98872, saving model to models\lstm_silence.h5
435/435 [==============================] - 98s 225ms/step - loss: 0.0702 - accuracy: 0.9887
Epoch 2/4
435/435 [==============================] - ETA: 0s - loss: 0.0198 - accuracy: 0.9966
Epoch 2: accuracy improved from 0.98872 to 0.99655, saving model to models\lstm_silence.h5
435/435 [==============================] - 65s 149ms/step - loss: 0.0198 - accuracy: 0.9966
Epoch 3/4
435/435 [==============================] - ETA: 0s - loss: 0.0143 - accuracy: 0.9974
Epoch 3: accuracy improved from 0.99655 to 0.99741, saving model to models\lstm_silence.h5
435/435 [==============================] - 69s 159ms/step - loss: 0.0143 - accuracy: 0.9974
Epoch 4/4
435/435 [==============================] - ETA: 0s - loss: 0.0128 - accuracy: 0.9973
Epoch 4: accuracy did not improve from 0.99741
435/435 [==============================] - 76s 175ms/step - loss: 0.0128 - accuracy: 0.9973
```

Figure 10: Chosen parameters for specific models

7.1.4 VAD

To verify if the VAD correctly classifies the silence samples, we created submission for Kaggle with only 'silence' labels, and submission with VAD classification. The detector yielded over 90% accuracy.



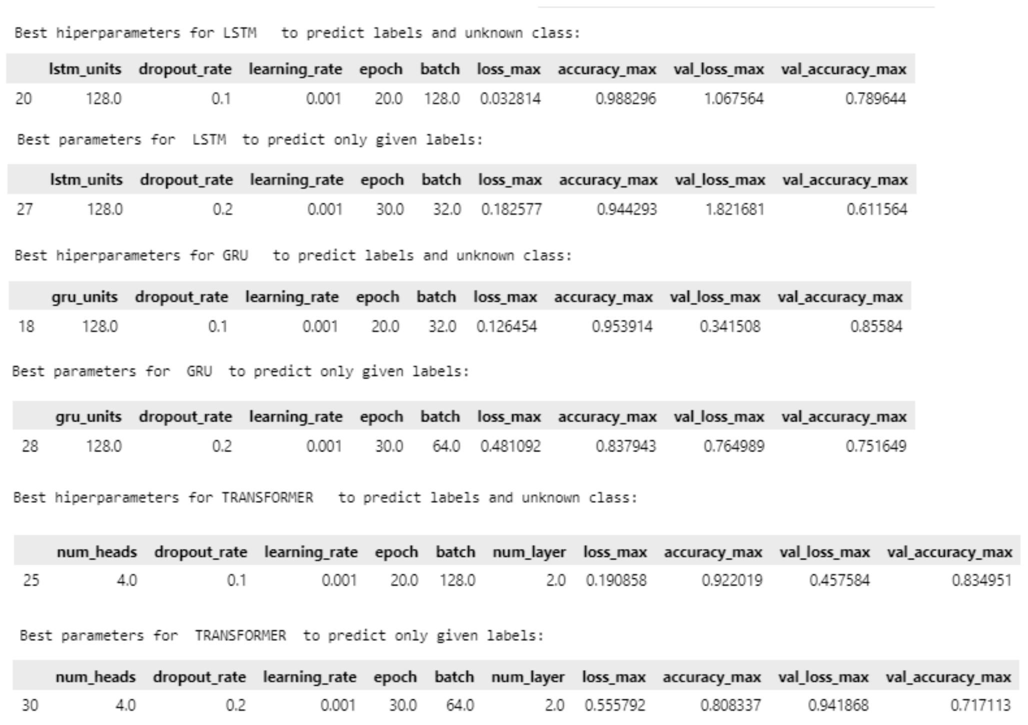
silence_only.csv
Complete (after deadline) · 21h ago
0.09949

submission_sil3.csv
Complete (after deadline) · 1d ago
0.09091

Figure 11: Silence classification check. 'submission sil3' - VAD prediction. 'silence only' - csv with 'silence' label in each row

7.1.5 known vs unknown detection + label detection

Below you can see the results for models from 'VAD + known vs unknown detection + label detection' concept.



```
Best hiperparameters for LSTM   to predict labels and unknown class:
```

	lstm_units	dropout_rate	learning_rate	epoch	batch	loss_max	accuracy_max	val_loss_max	val_accuracy_max
20	128.0	0.1	0.001	20.0	128.0	0.032814	0.988296	1.067564	0.789644

```
Best parameters for  LSTM  to predict only given labels:
```

	lstm_units	dropout_rate	learning_rate	epoch	batch	loss_max	accuracy_max	val_loss_max	val_accuracy_max
27	128.0	0.2	0.001	30.0	32.0	0.182577	0.944293	1.821681	0.611564

```
Best hiperparameters for GRU   to predict labels and unknown class:
```

	gru_units	dropout_rate	learning_rate	epoch	batch	loss_max	accuracy_max	val_loss_max	val_accuracy_max
18	128.0	0.1	0.001	20.0	32.0	0.126454	0.953914	0.341508	0.85584

```
Best parameters for  GRU  to predict only given labels:
```

	gru_units	dropout_rate	learning_rate	epoch	batch	loss_max	accuracy_max	val_loss_max	val_accuracy_max
28	128.0	0.2	0.001	30.0	64.0	0.481092	0.837943	0.764989	0.751649

```
Best hiperparameters for TRANSFORMER   to predict labels and unknown class:
```

	num_heads	dropout_rate	learning_rate	epoch	batch	num_layer	loss_max	accuracy_max	val_loss_max	val_accuracy_max
25	4.0	0.1	0.001	20.0	128.0	2.0	0.190858	0.922019	0.457584	0.834951

```
Best parameters for  TRANSFORMER  to predict only given labels:
```

	num_heads	dropout_rate	learning_rate	epoch	batch	num_layer	loss_max	accuracy_max	val_loss_max	val_accuracy_max
30	4.0	0.2	0.001	30.0	64.0	2.0	0.555792	0.808337	0.941868	0.717113

Figure 12: Chosen parameters for specific models

7.2 Confusion matrices

The confusion matrices presented below represent the performance of various recurrent neural network models trained on a labeled training set and tested on a validation set.

The results for three different types of models are presented in this section:

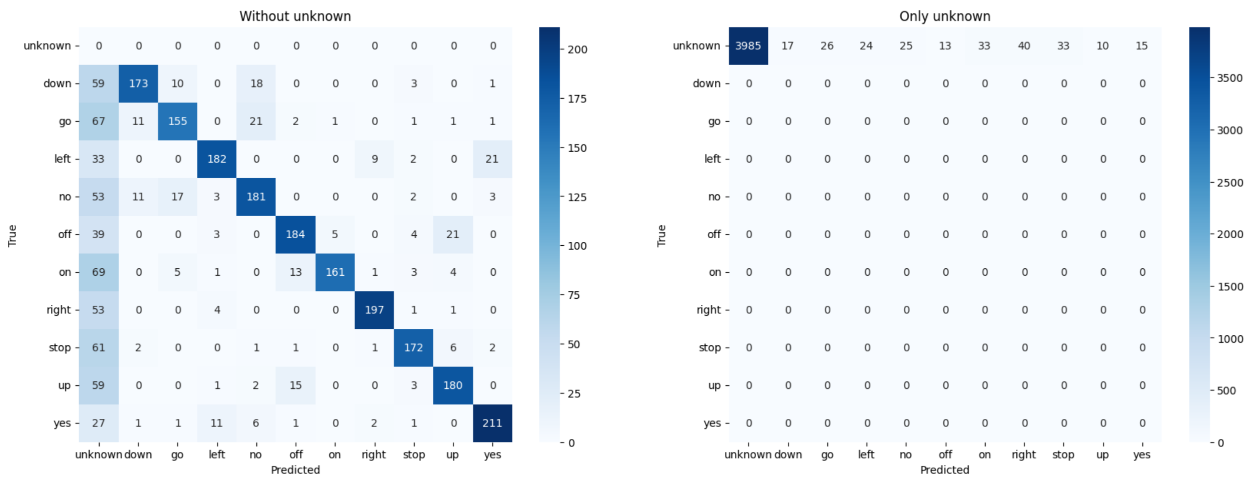
1. The first type of model is designed to predict specific labels and mark the rest as unknown.
2. The second type of model is designed to predict whether a given label is known or not.
3. The third type of model is designed to predict labels only from the known set of names.

For each neural network architecture (LSTM, GRU, Transformer) we chose the results of best models (with regard to hyper-parameter tuning process).

Additionally, we present confusion matrices of ensemble, built on five copies of the best model out of all - **GRU** with 128 gru units, 0.1 dropout rate, 0.001 learning rate, 20 epochs and 64 of batch size.

We evaluated ensemble's accuracy in terms of **mean** and **max** predicting methods.

7.2.1 LSTM



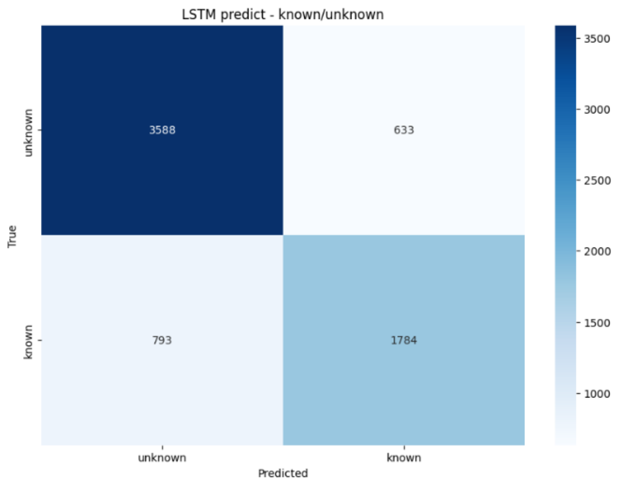
Without unknown

	unknown	down	go	left	no	off	on	right	stop	up	yes
unknown	0	0	0	0	0	0	0	0	0	0	0
down	59	173	10	0	18	0	0	0	3	0	1
go	67	11	155	0	21	2	1	0	1	1	1
left	33	0	0	182	0	0	0	9	2	0	21
no	53	11	17	3	181	0	0	0	2	0	3
off	39	0	0	3	0	184	5	0	4	21	0
on	69	0	5	1	0	13	161	1	3	4	0
right	53	0	0	4	0	0	0	197	1	1	0
stop	61	2	0	0	1	1	0	1	172	6	2
up	59	0	0	1	2	15	0	0	3	180	0
yes	27	1	1	11	6	1	0	2	1	0	211

Only unknown

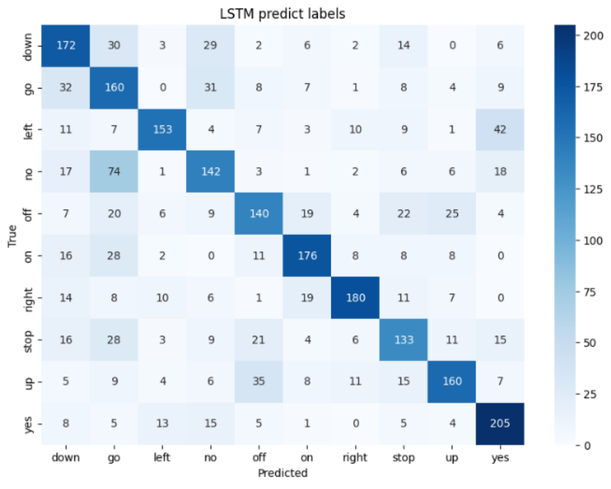
	unknown	down	go	left	no	off	on	right	stop	up	yes
unknown	3985	17	26	24	25	13	33	40	33	10	15
down	0	0	0	0	0	0	0	0	0	0	0
go	0	0	0	0	0	0	0	0	0	0	0
left	0	0	0	0	0	0	0	0	0	0	0
no	0	0	0	0	0	0	0	0	0	0	0
off	0	0	0	0	0	0	0	0	0	0	0
on	0	0	0	0	0	0	0	0	0	0	0
right	0	0	0	0	0	0	0	0	0	0	0
stop	0	0	0	0	0	0	0	0	0	0	0
up	0	0	0	0	0	0	0	0	0	0	0
yes	0	0	0	0	0	0	0	0	0	0	0

Figure 13: First type of model - LSTM



	unknown	known
unknown	3588	633
known	793	1784

Figure 14: Second type of model - LSTM



	down	go	left	no	off	on	right	stop	up	yes
down	172	30	3	29	2	6	2	14	0	6
go	32	160	0	31	8	7	1	8	4	9
left	11	7	153	4	7	3	10	9	1	42
no	17	74	1	142	3	1	2	6	6	18
off	7	20	6	9	140	19	4	22	25	4
on	16	28	2	0	11	176	8	8	8	0
right	14	8	10	6	1	19	180	11	7	0
stop	16	28	3	9	21	4	6	133	11	15
up	5	9	4	6	35	8	11	15	160	7
yes	8	5	13	15	5	1	0	5	4	205

Figure 15: Third type of model - LSTM

7.2.2 GRU

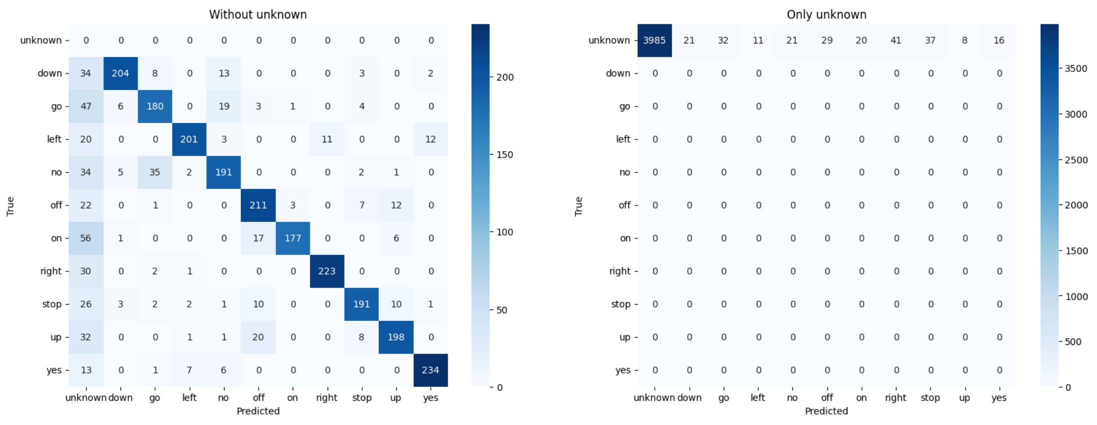


Figure 16: First type of model - GRU

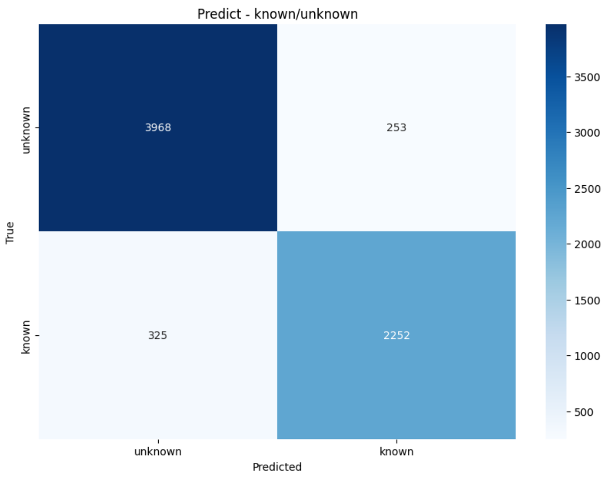


Figure 17: Second type of model - GRU

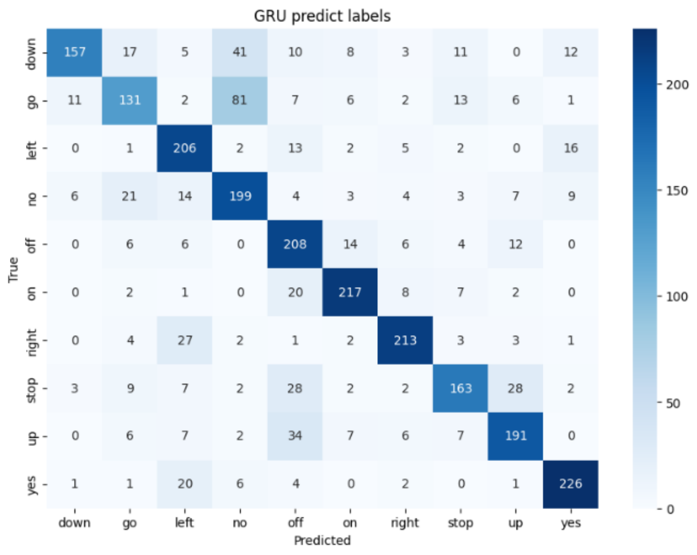
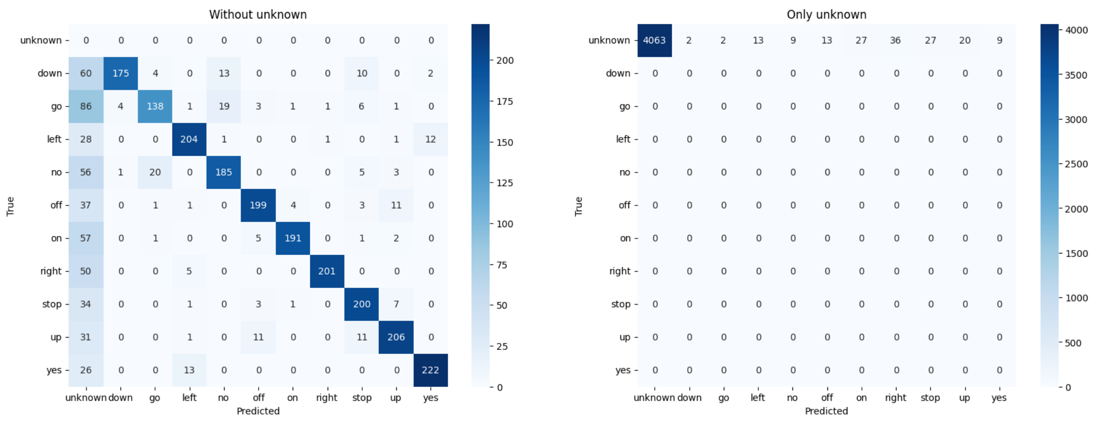


Figure 18: Third type of model - GRU

7.2.3 TRANSFORMER



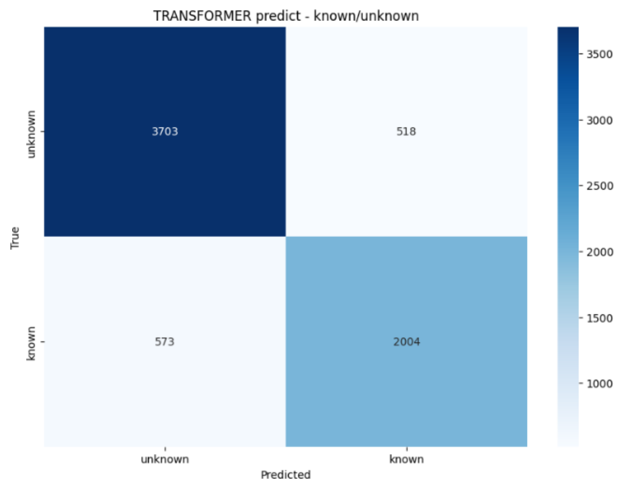
Without unknown (rows: True, columns: Predicted)

	unknown	down	go	left	no	off	on	right	stop	up	yes
unknown	0	0	0	0	0	0	0	0	0	0	0
down	60	175	4	0	13	0	0	0	10	0	2
go	86	4	138	1	19	3	1	1	6	1	0
left	28	0	0	204	1	0	0	1	0	1	12
no	56	1	20	0	185	0	0	0	5	3	0
off	37	0	1	1	0	199	4	0	3	11	0
on	57	0	1	0	0	5	191	0	1	2	0
right	50	0	0	5	0	0	0	201	0	0	0
stop	34	0	0	1	0	3	1	0	200	7	0
up	31	0	0	1	0	11	0	0	11	206	0
yes	26	0	0	13	0	0	0	0	0	0	222

Only unknown (rows: True, columns: Predicted)

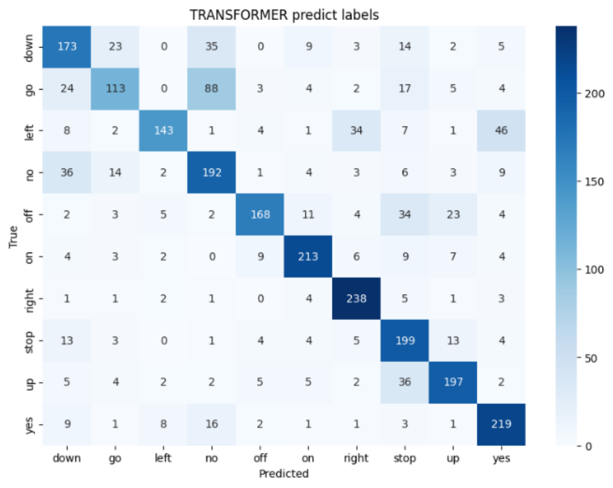
	unknown	down	go	left	no	off	on	right	stop	up	yes
unknown	4063	2	2	13	9	13	27	36	27	20	9
down	0	0	0	0	0	0	0	0	0	0	0
go	0	0	0	0	0	0	0	0	0	0	0
left	0	0	0	0	0	0	0	0	0	0	0
no	0	0	0	0	0	0	0	0	0	0	0
off	0	0	0	0	0	0	0	0	0	0	0
on	0	0	0	0	0	0	0	0	0	0	0
right	0	0	0	0	0	0	0	0	0	0	0
stop	0	0	0	0	0	0	0	0	0	0	0
up	0	0	0	0	0	0	0	0	0	0	0
yes	0	0	0	0	0	0	0	0	0	0	0

Figure 19: First type of model - Transformer



	unknown	known
unknown	3703	518
known	573	2004

Figure 20: Second type of model - TRANSFORMER



	down	go	left	no	off	on	right	stop	up	yes
down	173	23	0	35	0	9	3	14	2	5
go	24	113	0	88	3	4	2	17	5	4
left	8	2	143	1	4	1	34	7	1	46
no	36	14	2	192	1	4	3	6	3	9
off	2	3	5	2	168	11	4	34	23	4
on	4	3	2	0	9	213	6	9	7	4
right	1	1	2	1	0	4	238	5	1	3
stop	13	3	0	1	4	4	5	199	13	4
up	5	4	2	2	5	5	2	36	197	2
yes	9	1	8	16	2	1	1	3	1	219

Figure 21: Third type of model - TRANSFORMER

7.3 Ensemble

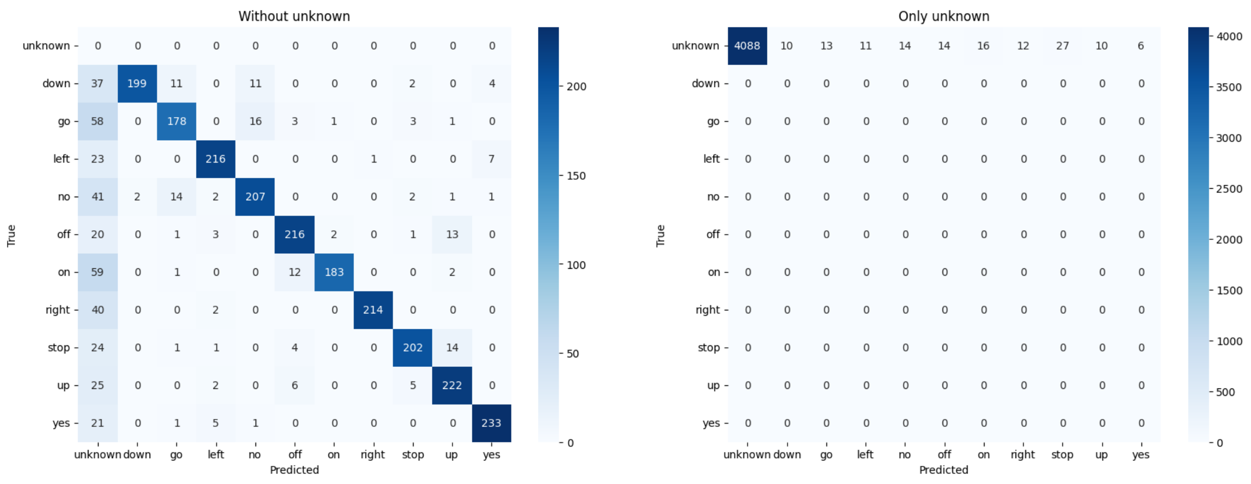


Figure 22: Ensemble mean

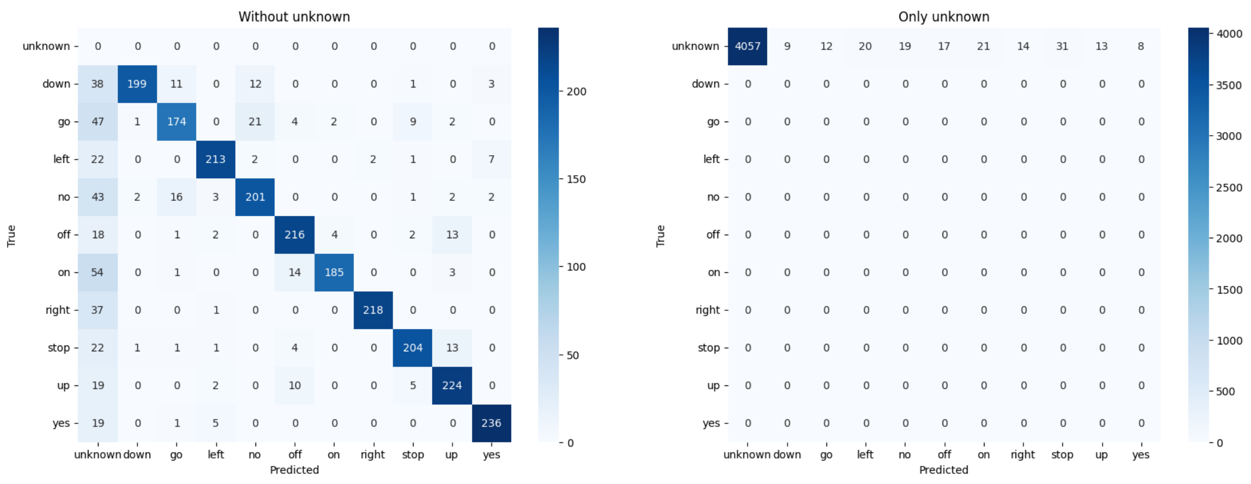


Figure 23: Ensemble max

7.4 Summary of results

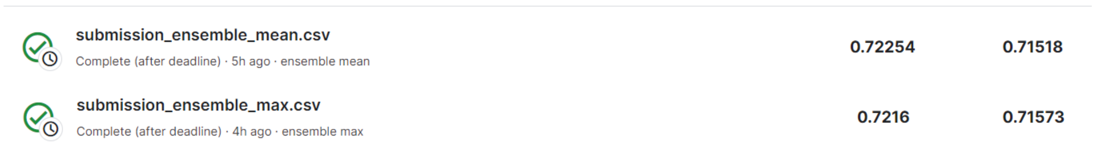
1. In most cases label detection models had the biggest problems with distinguishing **no** from **go** and **up** from **off**. This result is understanable, as these words could sound very similarly.
2. In some cases we can also observe mistakes with **stop** and **up** words. We suppose it is due to the fact, that ending of **stop** sounds like **up** and it led the models to misclassification.
3. The task of distinguishing between **known** and **unknown** labels was executed exquisitely by **GRU** classifier, which yielded over 90% of accuracy. Overall, our GRU architecture performed very well for all 3 models.

7.5 Kaggle scores

We evaluated 2 types of model conglomerates on test set:

1. The first one, hereafter referred to as **SKL**, is composed of three stages: predicting silence, predicting whether the label is known or not, and finally assigning the label. The name is an acronym of the respective stages.
2. The second model will be referred to as **SL**, as it checks for silence at the beginning, and then uses a model that predicts the label, but this time the model also predicts the label unknown at the same time.

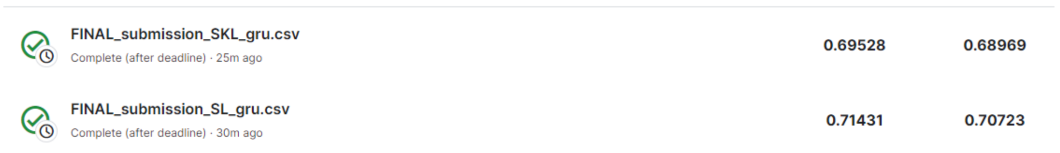
7.6 Ensemble



submission_ensemble_mean.csv Complete (after deadline) · 5h ago · ensemble mean	0.72254	0.71518
submission_ensemble_max.csv Complete (after deadline) · 4h ago · ensemble max	0.7216	0.71573

Figure 24: Ensemble scores

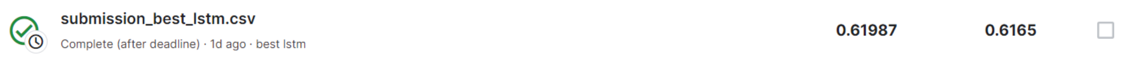
7.7 GRU



FINAL_submission_SKL_gru.csv Complete (after deadline) · 25m ago	0.69528	0.68969
FINAL_submission_SL_gru.csv Complete (after deadline) · 30m ago	0.71431	0.70723

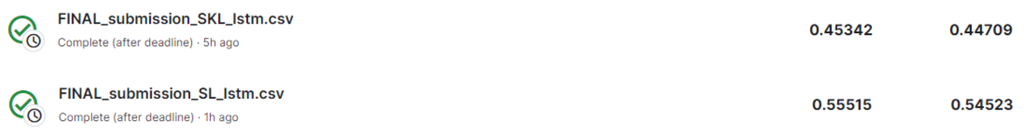
Figure 25: Final results of GRU type models.

7.8 LSTM



submission_best_lstm.csv Complete (after deadline) · 1d ago · best lstm	0.61987	0.6165

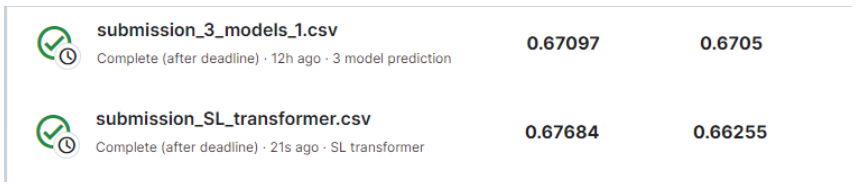
Figure 26: Best SL LSTM score.



FINAL_submission_SKL_lstm.csv Complete (after deadline) · 5h ago	0.45342	0.44709
FINAL_submission_SL_lstm.csv Complete (after deadline) · 1h ago	0.55515	0.54523

Figure 27: Final results of LSTM type models.

7.9 Transformer



submission_3_models_1.csv
Complete (after deadline) · 12h ago · 3 model prediction
0.67097 0.6705

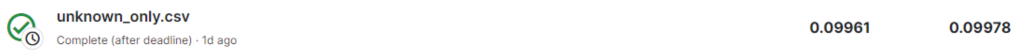
submission_SL_transformer.csv
Complete (after deadline) · 21s ago · SL transformer
0.67684 0.66255

Figure 28: Best SKL and SL Transformer score.

8 Conclusions

Ultimately, the best model was found to be a GRU model in the SL format and the best score was achieved on ensemble of this model. The gains achieved through better prediction of each stage were likely negated in the SKL model by the filtering of data going to the next layers.

Additionally, we checked that in the test set, the number of 'unknown' observations is only around 10%, so this model would likely to perform better if the amount of such observations was more significant in the tested data set.



unknown_only.csv
Complete (after deadline) · 1d ago
0.09961 0.09978

Figure 29: Percentage of 'unknown' labels in the whole test dataset.

In conclusion, the main lesson learned from obtaining the final result is that data processing is crucial for successful analysis and building a neural network. By carefully designing the processing steps and selecting appropriate features, we can significantly improve the performance of the model in real-world scenarios.

8.1 Reason of success

1. Good understanding of data, which led to proper pre-processing and feature extraction. Important factor was definitely a VAD detector, which extracted relevant parts of clips and detected silence with high accuracy.

2. Careful choice of RNN architectures with reccurent (attention) segments and extensive hyperparameter search, including grid and random searches.

3. Testing out different configurations of models.

8.2 Further research

In this section we propose research paths, which could bring the performance of our architecture to the next level and improve overall quality of models.

1. Data augmentation on known labels for SL type of models, as number of **unknown** classes outweighs the number of known labels in each group.

2. Hyperparameter search on larger space of parameters.

9 Bibliography

References

[1] https://www.youtube.com/watch?v=4_SH2nfbQZ8

[2] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, Jürgen Schmidhuber, LSTM: A Search Space Odyssey, link: https://arxiv.org/pdf/1503.04069.pdf

[3] https://www.youtube.com/watch?v=eCvz-kB4yko&t=956s

[4] https://www.kaggle.com/davids1992/speech-representation-and-data-exploration

[5] https://github.com/wiseman/py-webrtcvad/blob/master/example.py - we utilize code from this source.