Audio classification with transformers

Project report

Wojciech Klusek, Aleksander Kuś

May 06, 2024

Abstract

This document contains a project report regarding audio classification with transformers. The task was to build a model for classifying audio from the **TensorFlow Speech Recognition Challenge** dataset and test the influence of changing hyper-parameters and network architectures. Several research hypotheses were introduced and a series of experiments was conducted to test their relevance, including changing the learning rate parameter and merging all insignificant classes into one. We concluded that transformers can be used to solve audio classification problems and to achieve high accuracy. Changing the learning rate parameter did not change our results, but merging all classes into one improved them slightly.

Contents

1 Task description

The goal of this project is to classify speech commands employing neural networks, with a primary emphasis on utilizing Transformer architectures. The classification task will utilize a dataset consisting of recordings of various spoken commands. Multiple network architectures will be compared using the accuracy metric. A part of the task is also investigation of the influence of changing hyper-parameter values related to the training process.

1.1 Dataset description

The **TensorFlow Speech Recognition Challenge** dataset is an audio dataset containing 1 second clips of voice commands, with the folder name being the label of the audio clip. The labels that need to be predicted in are yes, no, up, down, left, right, on, off, stop, go. Everything else should be considered either unknown or silence. The folder *_background_noise_* contains longer clips of "silence" that need to be broken up and used as training input.

All audio files in the dataset belong to exactly one of the 31 classes, which are: bed, bird, cat, dog, down, eight, five, four, go, happy, house, left, marvin, nine, no, off, on, one, right, seven, sheila, silence, six, stop, three, tree, two, up, wow, yes, zero.

The task is to build a model that correctly classifies as many audio files as possible from the test set, based on training on data. In our tests we decided to only used to split data into test and train datasets with 80-20 proportion. All training of our networks was made on the train set and all evaluation of trained networks was made on the test set using the accuracy metric.

2 Network description

In this project, we employ Transformer architectures for audio classification tasks. Unlike traditional neural networks, Transformers utilize self-attention mechanisms to effectively capture dependencies within audio sequences.

Transformers have demonstrated success in various domains, including natural language processing and audio classification. Their ability to model long-range dependencies and capture contextual information makes them well-suited for audio classification tasks.

By leveraging self-attention layers, Transformers can efficiently process audio sequences of varying lengths, capturing both local and global dependencies. This allows for accurate classification of audio commands, making Transformers a promising choice for this task.

2.1 Architectures taken into account

In this project, we have leveraged two pre-trained architectures provided by Facebook AI: Wav2Vec and HUBERT.

2.1.1 Wav2Vec 2.0

Wav2Vec [1] is a transofmer-based architecture designed for self-supervised pre-training on raw audio data. The architecture of wav2vec 2.0 consists of a multilayer convolutional neural network, a quantizer, and a transformer. The convolutional neural network processes the raw waveform of the speech audio to get latent audio representations. The quantizer then chooses a speech unit for the latent audio representation from an inventory of learned units. About half of the audio representations are masked before being fed into the transformer. The transformer adds information from the entire audio sequence. Finally, the output of the transformer is used to solve a contrastive task, which requires the model to identify the correct quantized speech units for the masked positions.

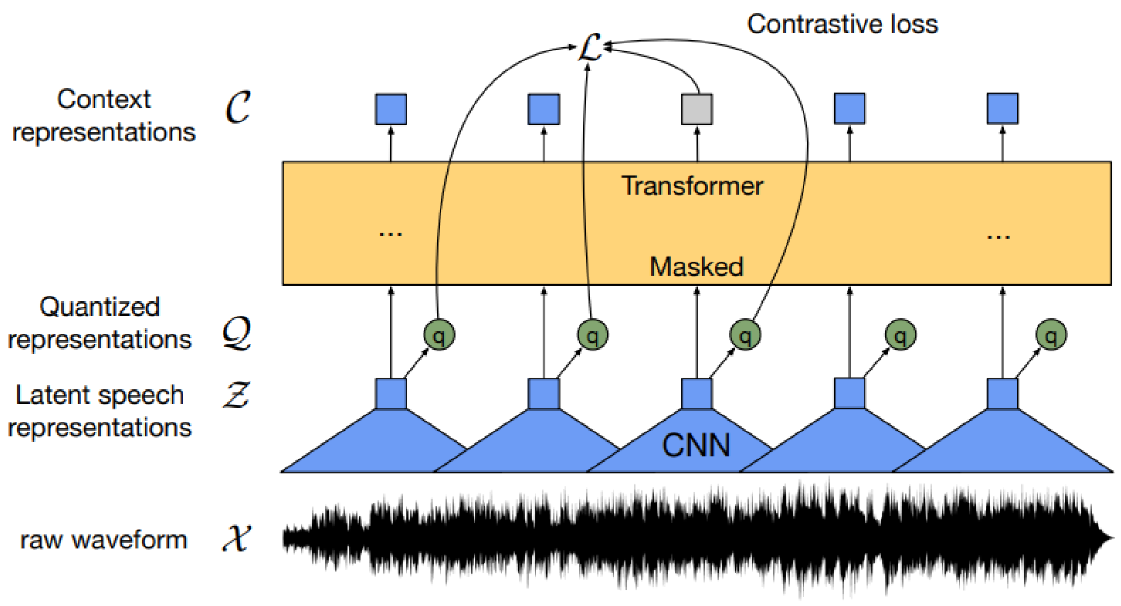


Figure 1: Wav2Vec

2.1.2 HUBERT

HUBERT [2], short for HUman Benchmark for Unsupervised Evaluation of Representations for Transformers, is another pre-trained audio model developed by Facebook AI. It uses masked prediction loss, akin to BERT, to capture speech's sequential structure. Before training, it employs offline clustering to generate noisy labels. This helps in learning representations of unmasked inputs. HuBERT learns acoustic and language models. It encodes audio into continuous representations and captures long-range temporal relations. It emphasizes the consistency of clustering mappings, enabling it to focus on sequential structure. Through iterative clustering and prediction steps, HuBERT progressively refines its representations, improving its ability to understand and model speech.

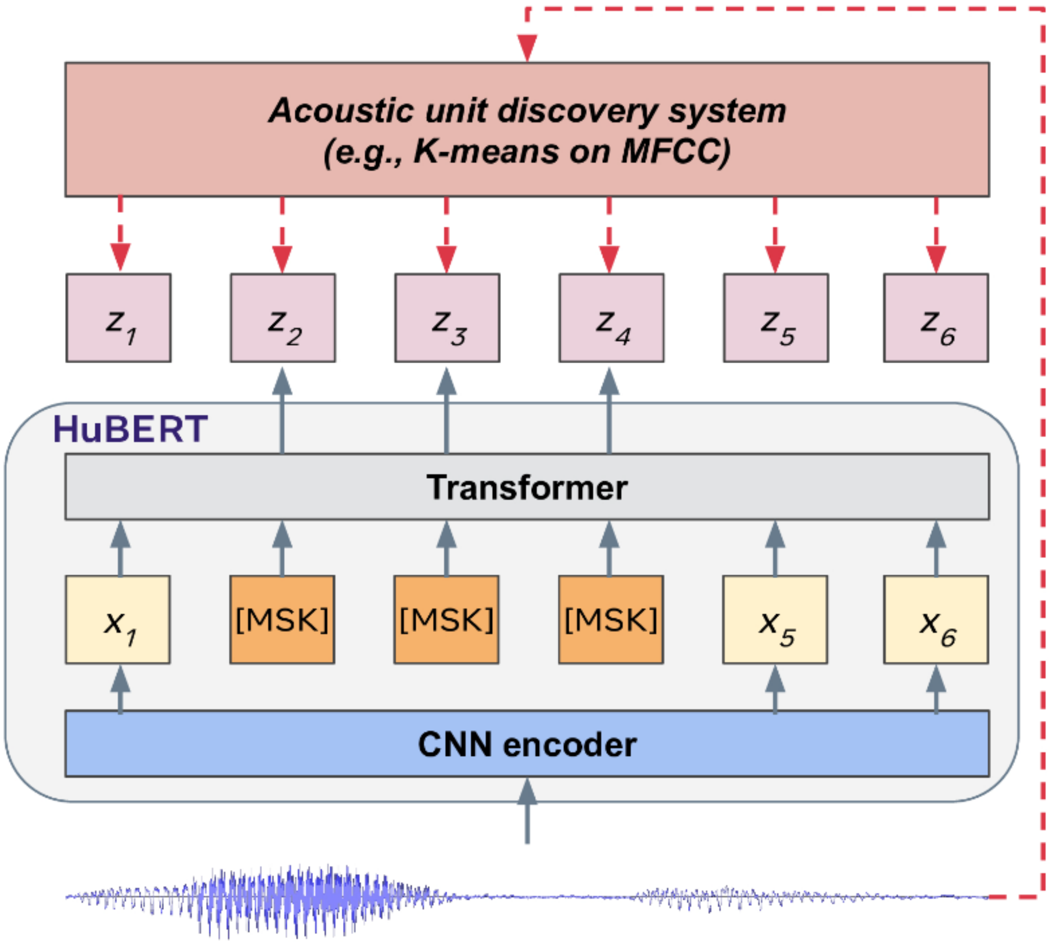


Figure 2: Hubert

2.2 Network parameters

In our tests we can modify three different hyper-parameters connected with the training process, which are:

- *NUM_TRAIN_EPOCHS* - The number of epochs during training. An epoch represents one complete pass through the entire training dataset. When all samples in the training set have been passed forward and backward through the network once, an epoch is completed. Training a network for more epochs means the network has more opportunities to learn and adjust its weights based on the given data.

- *LEARNING_RATE* - This parameter controls how much the weights of the network are updated during training. Higher learning rate might cause quick convergence, but risks overshooting the minimum loss value. Lower values of this parameter mean slower convergence, but the process is more precise.

- *PER_DEVICE_TRAIN_BATCH_SIZE/PER_DEVICE_EVAL_BATCH_SIZE* - The batch size specifies the number of training samples to be fed to the network in one forward and backward pass.

- *WARMUP_RATIO* - The proportion of training steps dedicated to "warming up" the learning rate scheduler. During the warmup phase, the learning rate gradually increases from a small value to its target value, allowing the model to stabilize before proceeding with full training.

- *GRADIENT_ACCUMULATION_STEPS* - This parameter controls how often gradients are accumulated before updating the model's weights. It allows for larger effective batch sizes, which can be beneficial for training with limited GPU memory or when using smaller batch sizes for fine-tuning pre-trained models.

3 Used datasets

When training and evaluating our models, we have used two dataset, described below:

1. *train* - the original training dataset from the task description. We have split this set to our own train and test sets with 80/20 ratio. The reason why we didn't use the test set from the task is that it is unlabeled and we wouldn't know the accuracy of our model anyways. The only modification we made to this set is adding the *silence* class, which we did by splitting audio files from the *_background_noise_* folder into one second bits.

2. *train-unknown* - after adding the silence class, as described above, we also wanted to test out a different way of training our models. We have moved every file not belonging to the classes described in section 1.1 to a separate class called *unknown*. Models trained on this dataset are described in section 5.2.

4 Research hypotheses

Before conducting any experiments we created the following research hypotheses:

1. Models trained on the Wav2Vec architecture will achieve higher accuracy scores than the ones trained on HUBERT.

2. Merging all insignificant classes to one class named "unknown" will allow the models to achieve higher accuracy scores.

3. Increasing the learning rate parameter will make the models achieve higher accuracy scores.

5 Conducted experiments and results

To verify our hypotheses, we conducted various experiments. Each experiment was run with the seed for the pseudorandom number generator set to 1 to ensure reproducibility. We describe each of our experiments below.

5.1 Network architecture comparison

To verify the network architectures described in section 2.1 we built two machine learning models and compared their accuracy. The models were trained on the *train* dataset, as described in section 3. The exact values of hyper-parameters used can be seen in the source code. On figure 3 we can observe the value of the loss function for both models during training. Both of the loss functions decreased significantly during the first epochs and remained on a similar level in further epochs. On figure 4 we can see the accuracy values on the test dataset after every epoch.

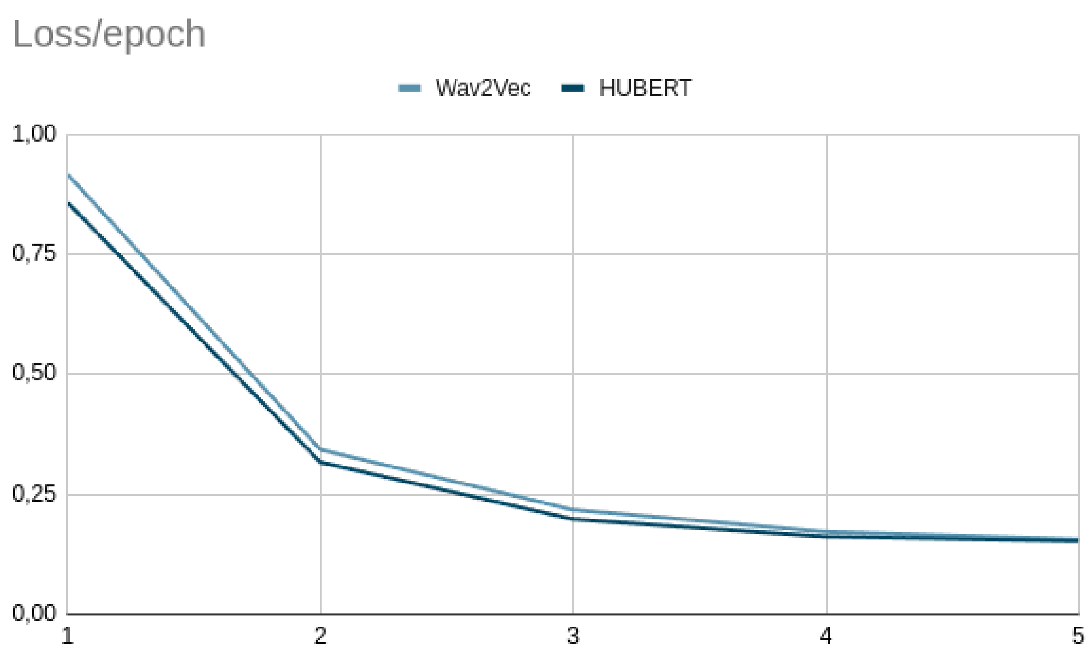


Figure 3: Loss function value per epoch

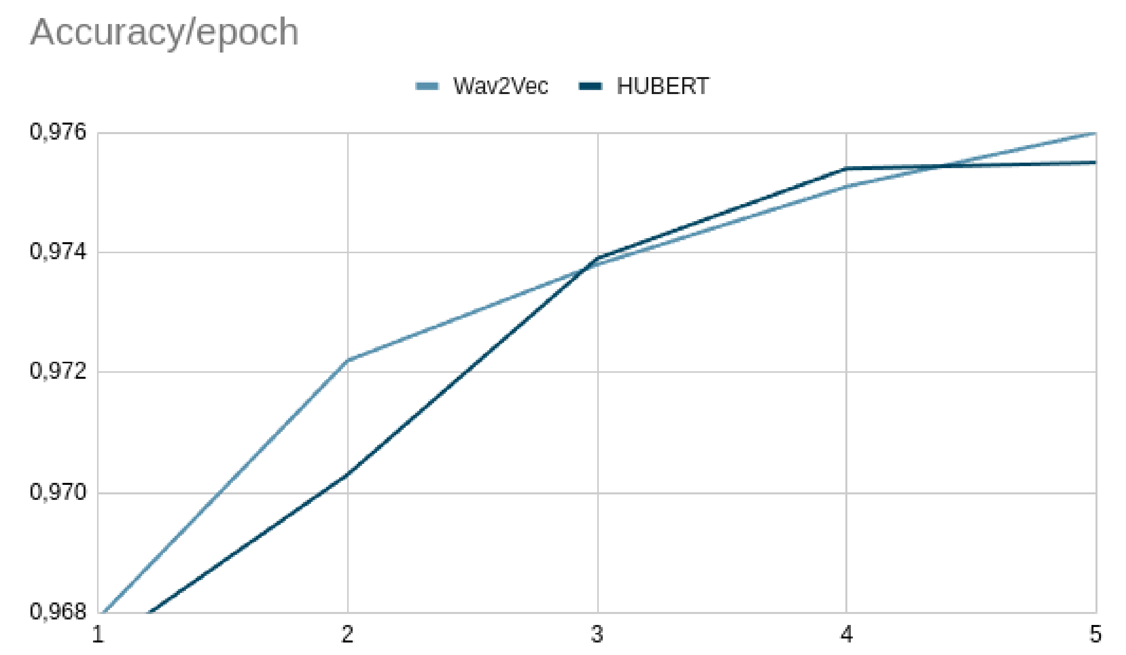


Figure 4: Accuracy value per epoch

On figures 5 and 6 we can see the confusion matrices for the Wav2Vec and HUBERT models respectively. We also tried filtering out all test samples with labels not matching the ones specified in the task. The list can be found in section 1.1. We obtained confusion matrices that can be examined on figures 7 and 8.

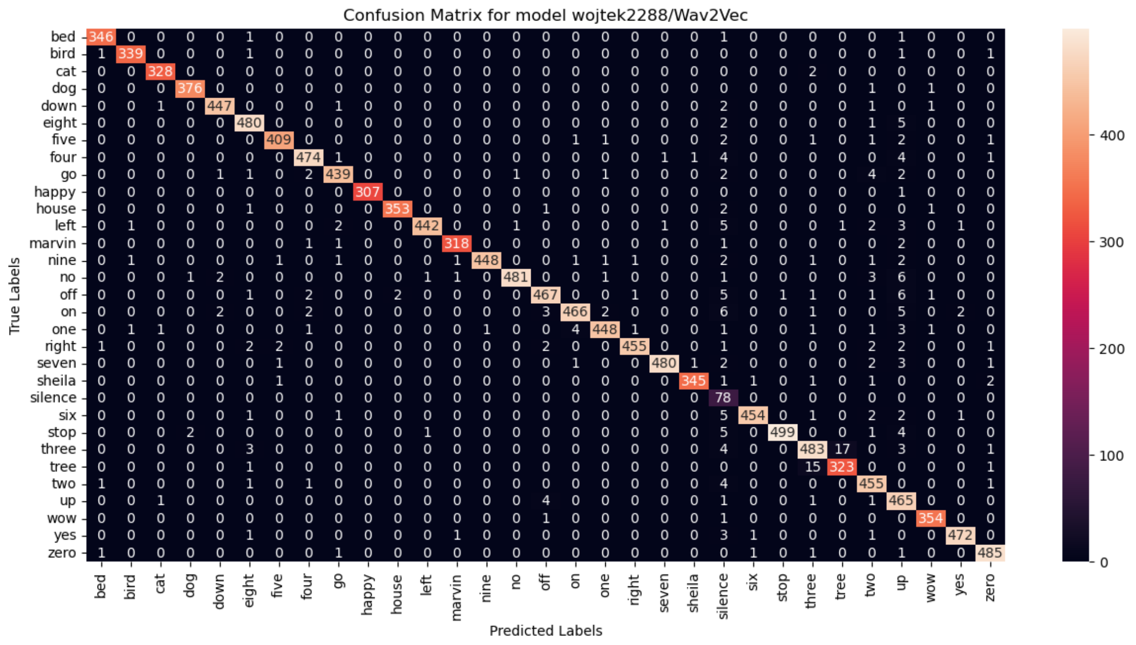


Figure 5: Confusion matrix for the Wav2Vec model

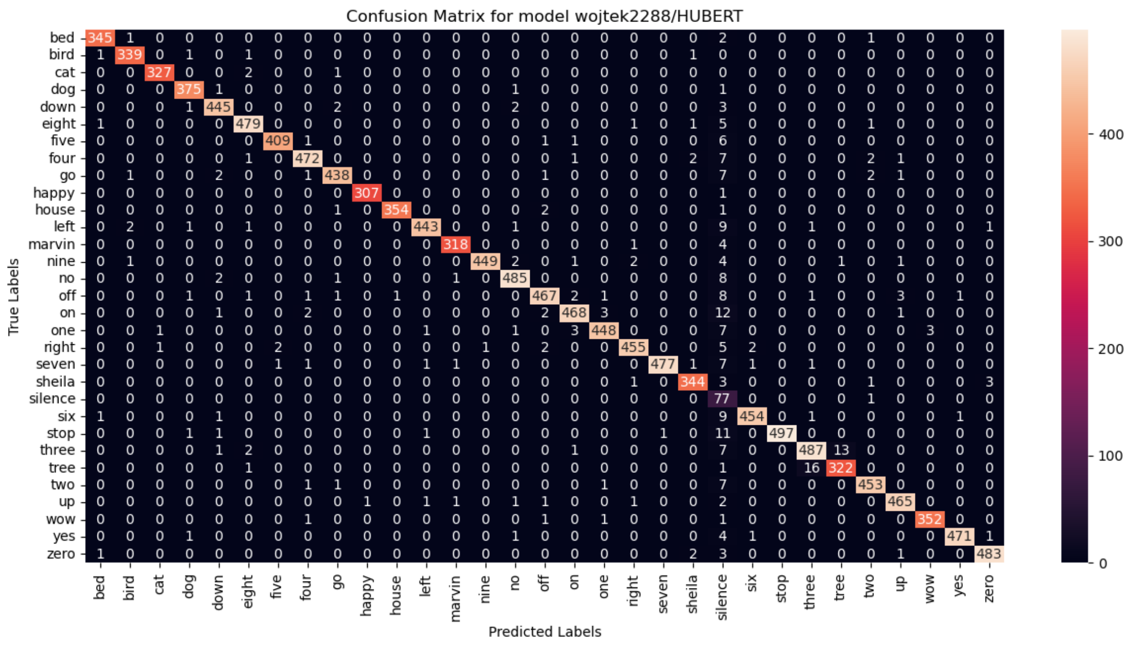


Figure 6: Confusion matrix for the HUBERT model

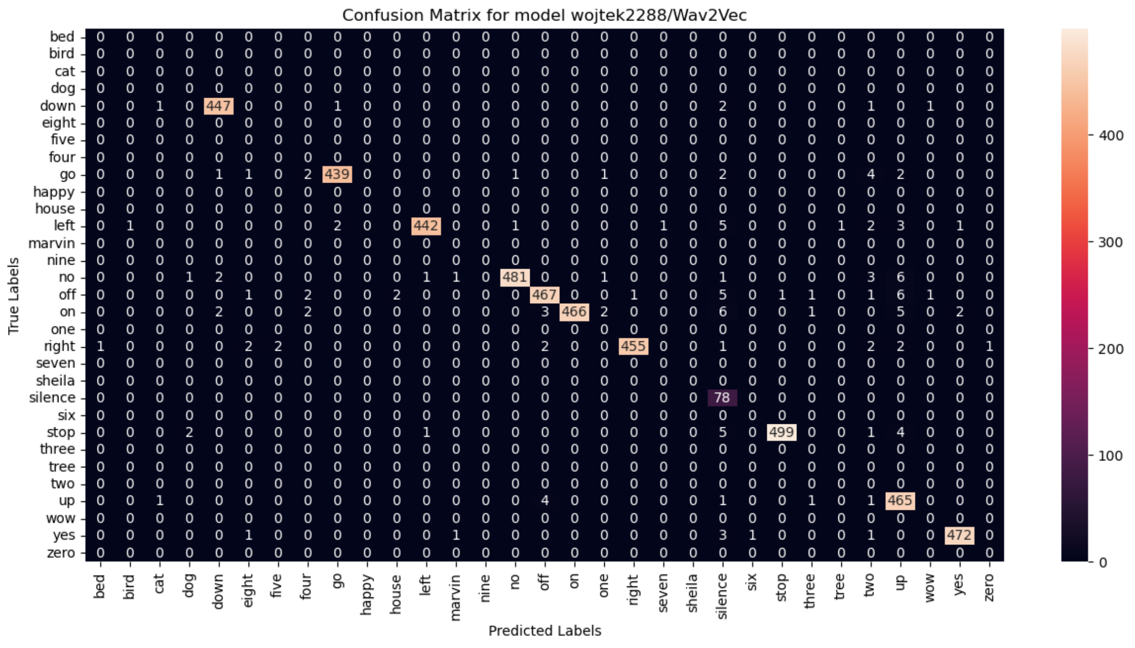


Figure 7: Confusion matrix for the Wav2Vec model for filtered test data

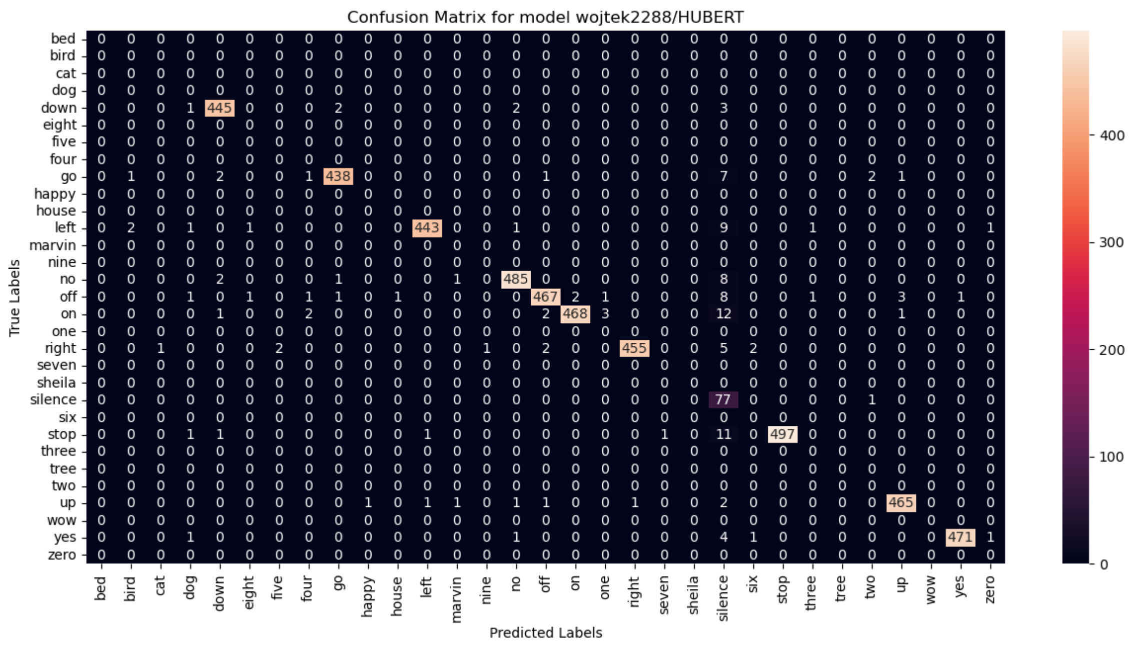


Figure 8: Confusion matrix for the HUBERT model for filtered test data

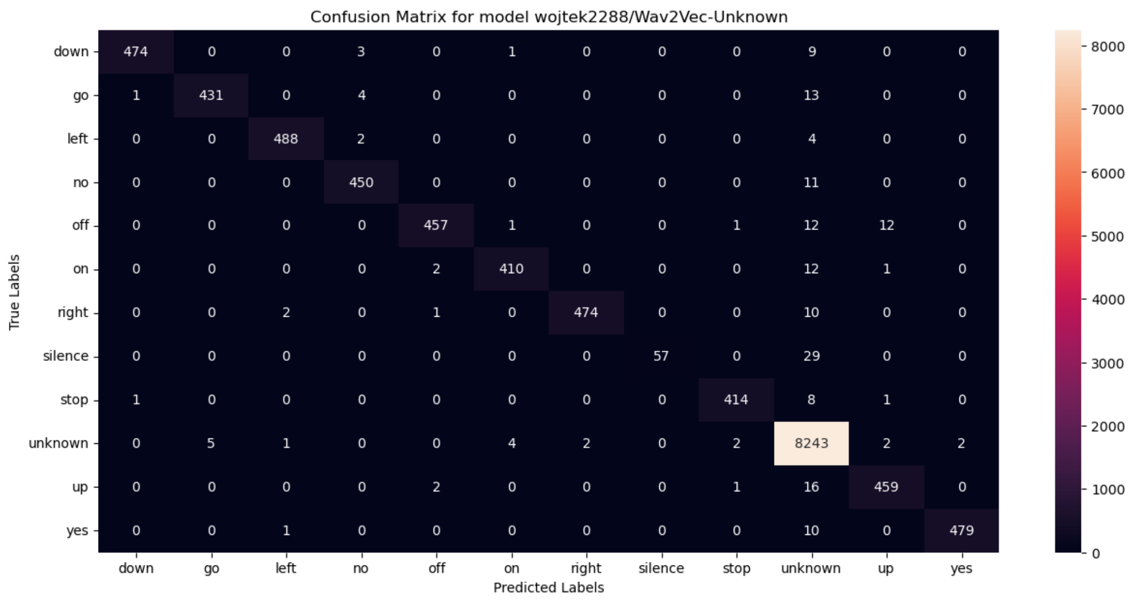
Finally, the overall accuracy for the Wav2Vec model was 0.98 for both the training and test sets. For the HUBERT model these values were the same.

As we can see in our results, both of our models performed very well and gave us satisfying results in terms of accuracy on the test set.

5.2 Merging unknown class

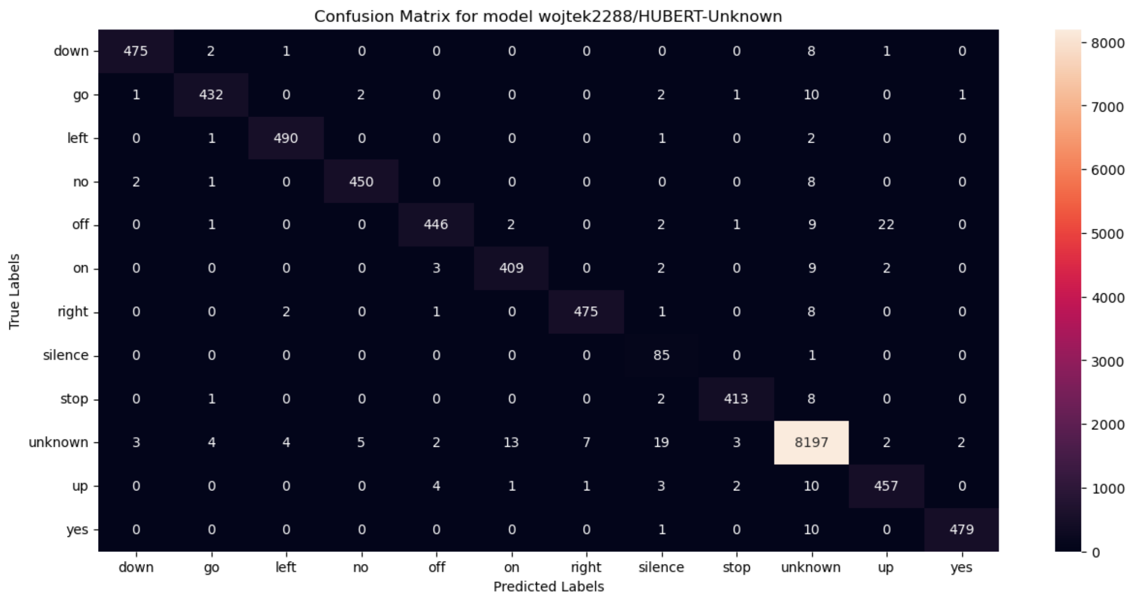
We have also tested if using the *train-unknown* dataset, described in section 3, would produce better results. The confusion matrices for the Wav2Vec and HUBERT models can be found on figures 9 and 10. The overall accuracy for these models was as follows:

1. Wav2Vec - 0.99 for training and test,
2. HUBERT - 0.99 for training and for 0.98 test.



	down	go	left	no	off	on	right	silence	stop	unknown	up	yes
down	474	0	0	3	0	1	0	0	0	9	0	0
go	1	431	0	4	0	0	0	0	0	13	0	0
left	0	0	488	2	0	0	0	0	0	4	0	0
no	0	0	0	450	0	0	0	0	0	11	0	0
off	0	0	0	0	457	1	0	0	1	12	12	0
on	0	0	0	0	2	410	0	0	0	12	1	0
right	0	0	2	0	1	0	474	0	0	10	0	0
silence	0	0	0	0	0	0	0	57	0	29	0	0
stop	1	0	0	0	0	0	0	0	414	8	1	0
unknown	0	5	1	0	0	4	2	0	2	8243	2	2
up	0	0	0	0	2	0	0	0	1	16	459	0
yes	0	0	1	0	0	0	0	0	0	10	0	479

Figure 9: Confusion matrix for the Wav2Vec model with merged unknown class



	down	go	left	no	off	on	right	silence	stop	unknown	up	yes
down	475	2	1	0	0	0	0	0	0	8	1	0
go	1	432	0	2	0	0	0	2	1	10	0	1
left	0	1	490	0	0	0	0	1	0	2	0	0
no	2	1	0	450	0	0	0	0	0	8	0	0
off	0	1	0	0	446	2	0	2	1	9	22	0
on	0	0	0	0	3	409	0	2	0	9	2	0
right	0	0	2	0	1	0	475	1	0	8	0	0
silence	0	0	0	0	0	0	0	85	0	1	0	0
stop	0	1	0	0	0	0	0	2	413	8	0	0
unknown	3	4	4	5	2	13	7	19	3	8197	2	2
up	0	0	0	0	4	1	1	3	2	10	457	0
yes	0	0	0	0	0	0	0	1	0	10	0	479

Figure 10: Confusion matrix for the HUBERT model with merged unknown class

As can be seen from the results, we did manage to slightly improve the performance of our models, thanks to decreasing the number of classes.

5.3 Learning rate parameter increase

In our final experiment we wanted to test the influence of changing hyper-parameters during training on the overall accuracy of the model. We have selected the *learning_rate* parameter, as we recall that changing it had significant influence in our previous projects. The first two models described in section 5.1 were trained with the value set to $3*10^{-5}$. This time we set it to $1*10^{-4}$ and trained two more models.

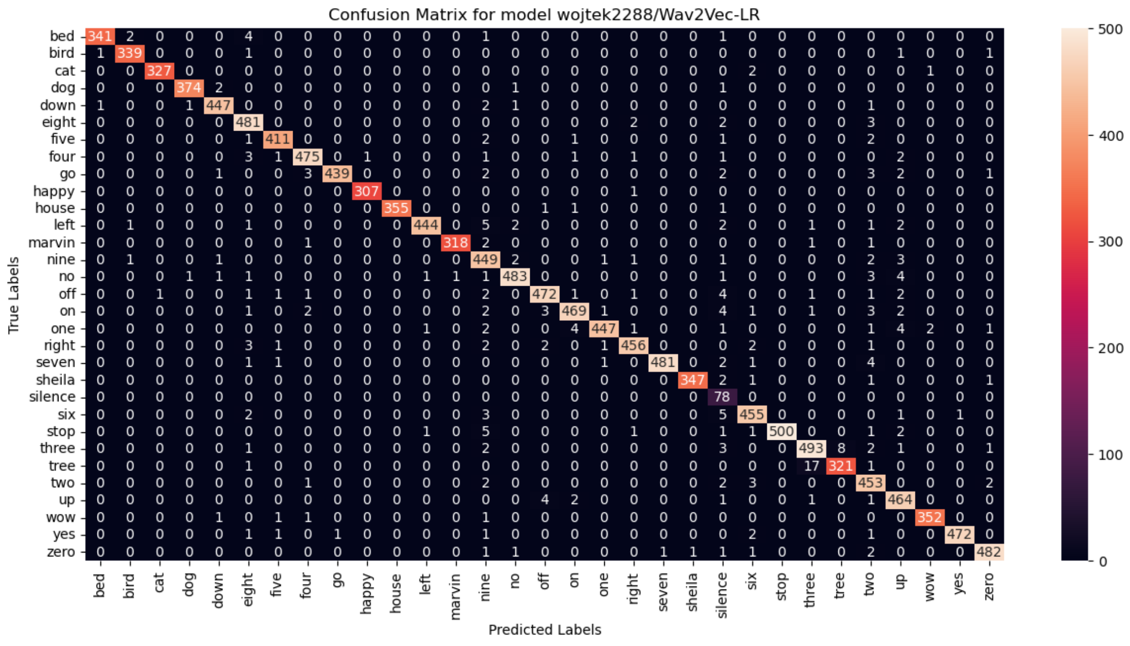


Figure 11: Confusion matrix for the Wav2Vec model with higher learning rate

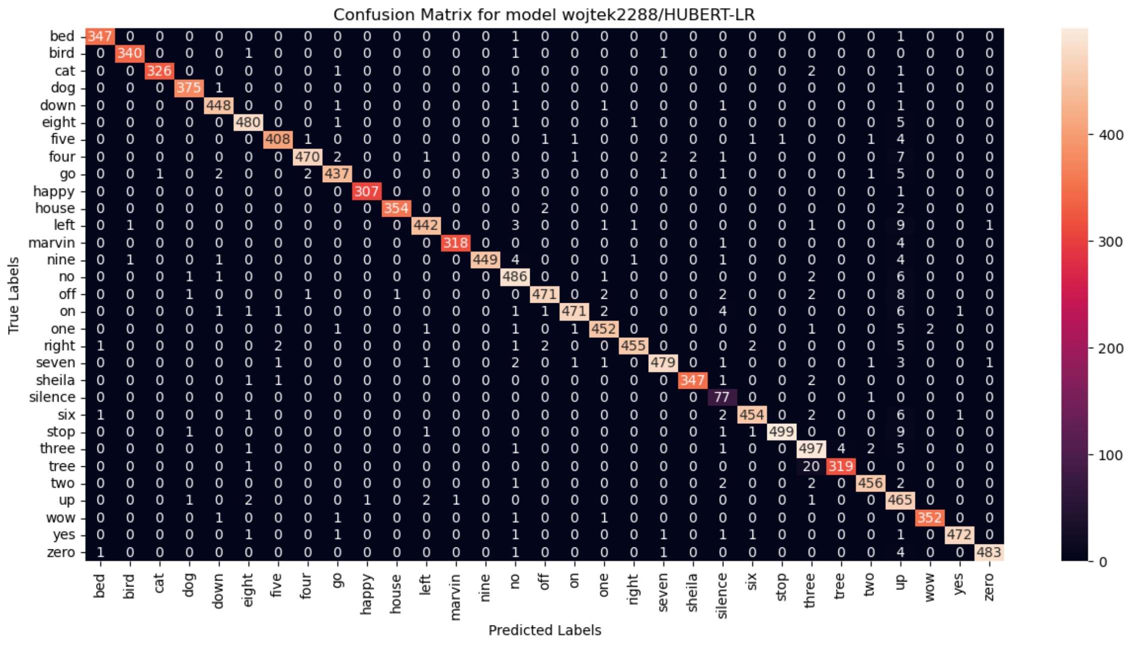


Figure 12: Confusion matrix for the HUBERT model with higher learning rate

The confusion matrices for test data can be found on figures 5 and 12. The overall accuracy on train and test data did not change and remained at 0.98 for both models. As we can see, changing the *learning_rate* parameter did not have an impact on the results.

6 Hypotheses verification

With the results of our conducted experiments we are able to verify our hypotheses:

1. Models trained on the Wav2Vec architecture will achieve higher accuracy scores than the ones trained on HUBERT - **REJECTED**. Both models performed very similarly and we cannot point out the better one.

2. Merging all insignificant classes to one class named "unknown" will allow the models to achieve higher accuracy scores - **CONFIRMED**. Lowering the number of classes improved our results slightly.

3. Increasing the learning rate parameter will make the models achieve higher accuracy scores - **REJECTED**. Changing the learning rate parameter in both models did not impact the overall accuracy scores of these models.

7 Conclusions

To conclude we can tell that models based on the transformer architecture are well suited for audio classification and speech recognition tasks. Be built two models based on different architectures and they performed very well, achieving over 98% accuracy on test data.

In terms of achieved results, we did not find major differences between the Wav2Vec and HUBERT architectures. The both performed equally well.

A surprising factor for us was the fact that changing the learning rate parameter did not influence the results. We suspect that if the parameter was changed more drastically, the results would probably differ.

We have tested two different approaches regarding the dataset. The second one, when we merged many classes into one, gave us a higher overall accuracy for both of our models. We suspect that the fact of lowering the number of classes has positive impact on the model accuracy, as it doesn't have to learn the differences between all the samples in this one big class.

8 Application instruction

We decided to use the Python programming language language was used to implement the solution because of the many libraries supporting work with neural networks and artificial intelligence issues. Our solution is available as a Jupyter Notebook, which allows for quick running and visualisation of our tests. We used the `hugging face` library when building and testing our models, which is a robust framework for working with Artificial Neural Networks which provides pretrained transformer models.

To run the provided application the following software must be installed:

1. Python programming language version 3.11 or higher,

2. `librosa`, `soundfile`, `accelerate`, `wandb`, `datasets`, `transformers`, `evaluate`, `huggingface_hub`, `torch`, `pydub` libraries.

8.1 Training

To train the models you should use `main.ipynb` file. Near the beginning of the provided Jupyter Notebook there is a section titled "Parameters", in which all hyper-parameters for the networks can be changed (described in section 2.2). To reproduce training please input the correct parameters in this section, run `wandb login` then `huggingface-cli login` and run the notebook. Additionally you can split background noise class into smaller chunks by creating `silence` folder in dataset and setting `SPLIT_SILENCE` variable to `True`.

8.2 Experiments

To view the results of our experiments you should open the `results.ipynb` file. It is a Jupyter Notebook, allowing to view the results in a convenient way. The datasets for the models shall be in two folders, *train* and *train-unknown*. The former is the original training dataset that we use for both training and testing. The other is the original dataset with every class moved to the unknown folder except the classes highlighted in section 1.1. In the file there are confusion matrices for each model as well as a graph showing accuracy per class. You can also view the overall model accuracy from there.

References

[1] Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations, 2020.

[2] Wei-Ning Hsu, Benjamin Bolte, Yao-Hung Hubert Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, and Abdelrahman Mohamed. Hubert: Self-supervised speech representation learning by masked prediction of hidden units, 2021.