CDHI Challenge

Building android deployed speech classifier

Alexander Büsser

March 2020

Summary

To meet the objective of building a android deployable speech classifier that can distinguish **speaking**, from **singing** and **silence** a **convolutional net on the raw audio signal** was trained inspired by the work of Wei Dai. et al. [1] .

A visual inspection of the raw waveforms of the different audio showed notable differences. It was hence decided to deviate from the more commonly found approach in literature, namely deriving different representation of the raw signal such as spectrograms or Mel Frequency Cepstral Coefficients (MFCC).

Additional practical consideration favored applying a CNN on the raw signal as complex feature engineering make it harder to deploy the solution on different runtime environments (e.g. Swift, Android).

This work is **a proof of ability to meet specifications under a tight time frame.** It is **discussed what should have been done differently** given adequate time and computing resources.

1: https://arxiv.org/pdf/1610.00087.pdf

Data Preparation

1. The raw audio files were downloaded on Zenodo[2]

2. As the raw data only contained **singing** and **speaking** examples, **silence** samples were recorded manually. The recording was copied **1080** times to match the frequency of singing and speaking and thus avoiding a class imbalance.

3. Due to the different recording sources of singing & speaking vs. silence the properties **sample rate**, **number of channels and bit depth** were investigated. As expected differences can be observed in the sample rate as well as the number of channels

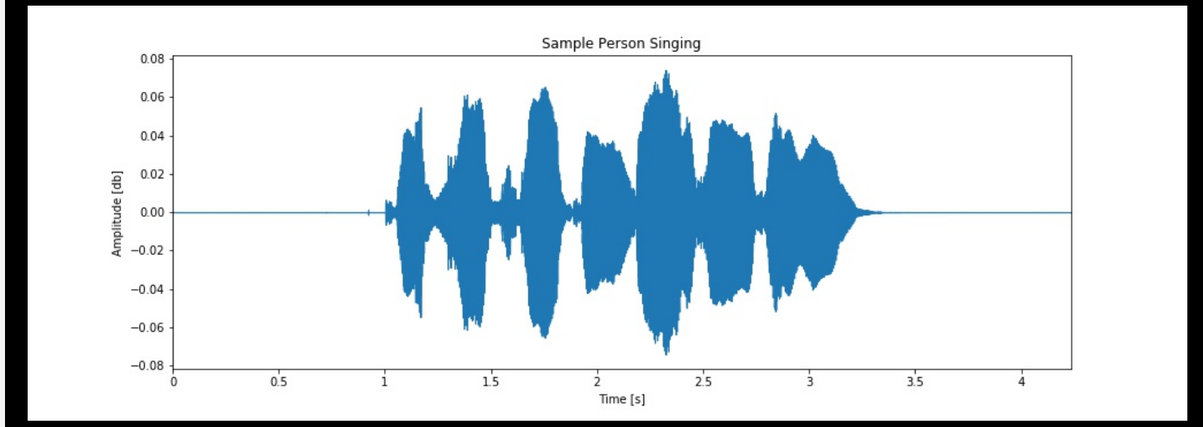
```
48000    0.698974
44100    0.301026
Name: sample_rate, dtype: float64
1    0.697263
2    0.302737
Name: num_channels, dtype: float64
16    1.0
Name: bit_depth, dtype: float64
```

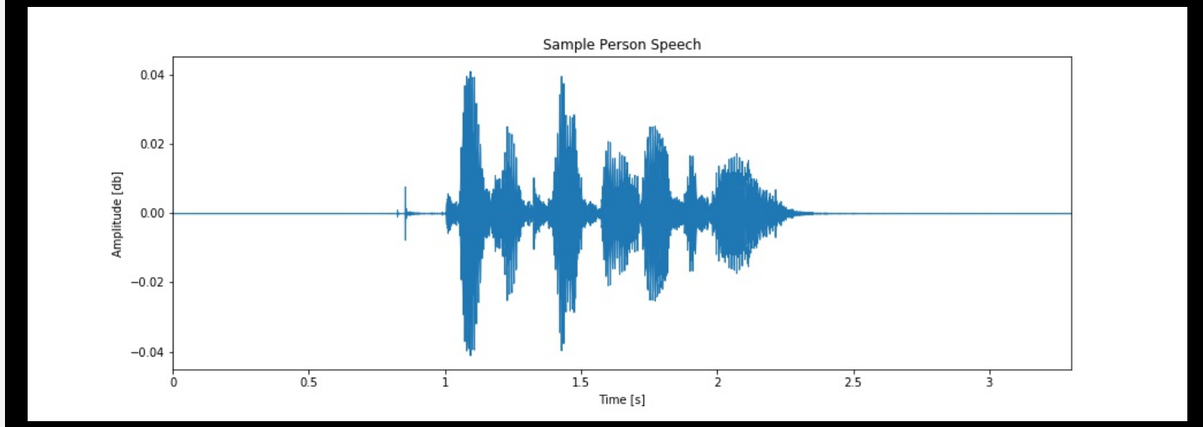
4. The waveform as well as the spectrograms of the raw samples were created. Having had little experience with audio recognition it was difficult to recognize clear differences between the signals. The wave form on the other hand gave us enough confidence that the raw signal should be interpretable by an algorithm without having to process it with mfcc.

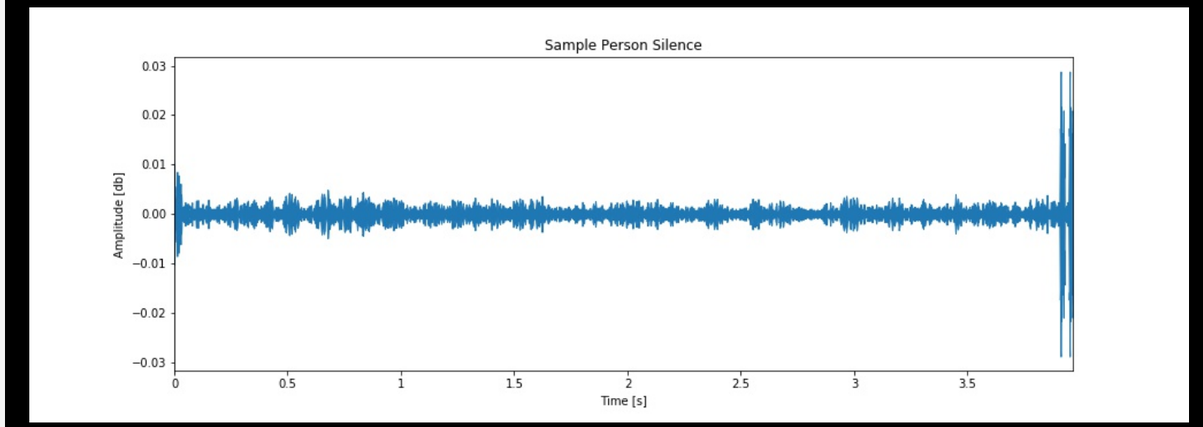
5. The librosa library was used to normalize the audio signals with a sample rate of **8000 Hz.** Furthermore an offset of one second was chosen, due to little activity as seen in the waveforms, and a duration of two seconds according to the specifications. Data was split into **80% train and 20% test.**

Should have been done differently: Data should be unmeetable. Due to the absence of silence data manual involvement was necessary, however the replication of the silence sample should have been performed by a script. Specific operations have been performed to meet the input specification of the model . This however, does not make the data preparation agnostic to other model. Central feature store, where derivatives (spectrogram, mfcc) are stored , should be preferred in an ideal setting.

2: https://zenodo.org/record/1188976#.Xmz_9UPTXyI







Modelling

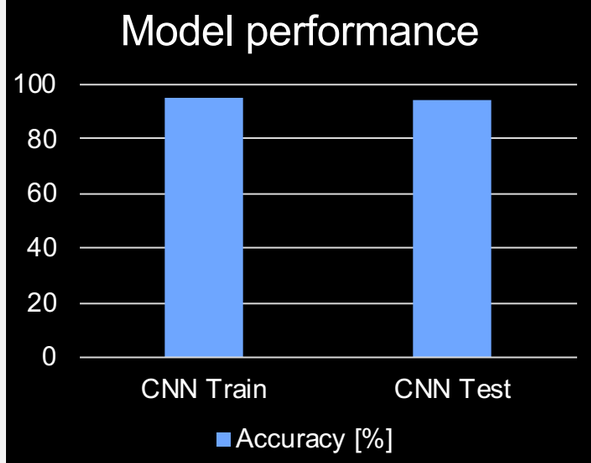
1. A short investigation of the field showed that methods usually focus on **1.) extracting interpretable features from the raw signal** or 2.) building **complex models with the hope to combine the feature extraction** along with classification. The latter was chosen due to promising results in Wei Dai. et al. [1] papers, along with the added benefit of easing deployment onto the mobile device as feature extraction methods are not needed at runtime.

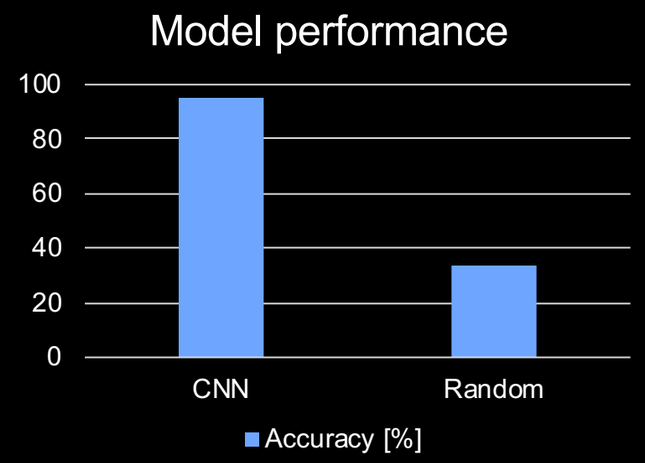
2. The high-level intuition of the chosen CNN architecture is that the convolution window is able to learn features from the time domain waveform. It is hypothesized that depth is needed to learn high level features. Max Pooling, a technique to extract the highest value in the window retains the most important information and subsequently reduces the dimensionality. Dropout layers are further used to maximize representation learning. Large receptive fields are used in the first layers to mimic a bandpass filter, smaller ones subsequently to control the model capacity.

3. Due to limited computing resources we significantly **reduced the number of epochs** compared to the paper from **400 to 40**. At training time, a loss of the oscillation was observed, which resulted in a significant impact on the test accuracy at the respective epochs. As a result a callback function to store the weights of the epoch with the highest performance on the test set was implemented.

4. No access to GPU limited experimentation of the different hyperparameters. The used set of hyperparameters is noted in the appendix.

5. The best results were achieved at epoch 25. The train and test results are noted below indicating a good ability of the model to generalize . Furthermore it can be seen that our model shows a performance **increase of around 61 percentage points compared to a random model.**

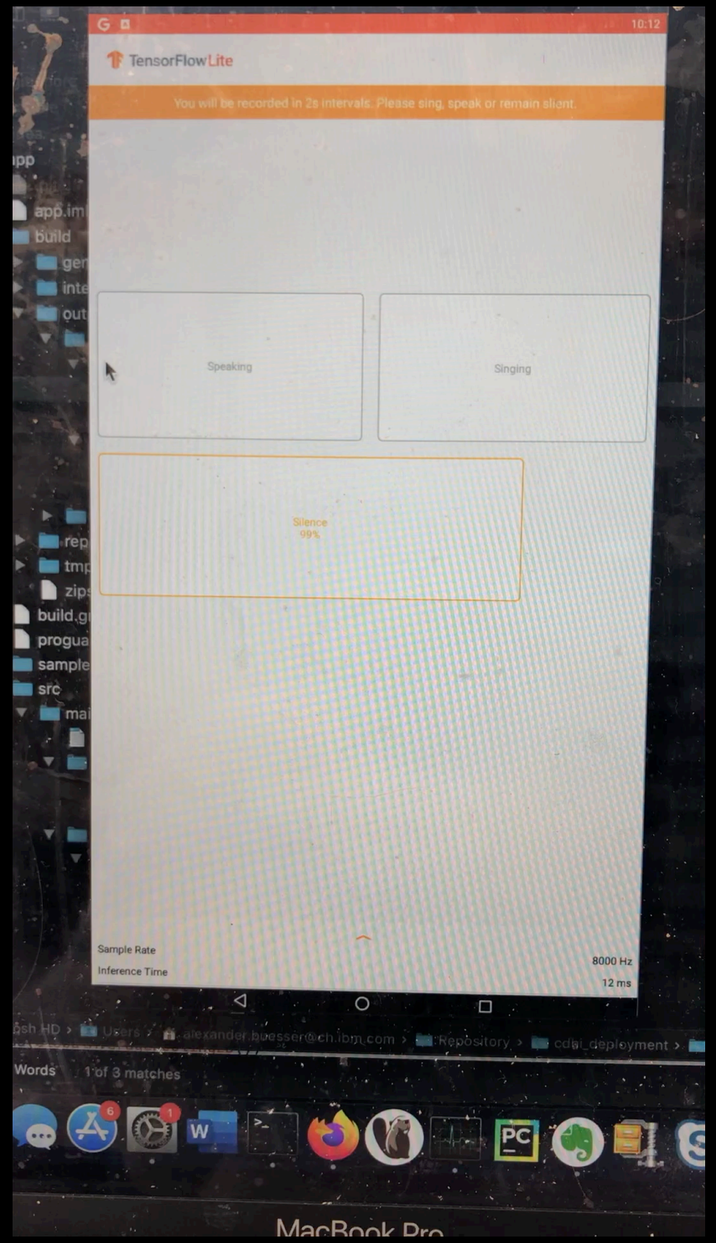




Should have been done differently: The model's strong deviation in performance between different epochs suggests learning difficulties. Such behaviour is not uncommon with un-scaled data. Subtracting the mean and dividing by the standard deviation of each column might have led to a more stable gradient and a faster convergence . Additionally the hyperparameters were chosen arbitrarily. A grid over a defined space of the most essential parameters (learning rate, epochs, regularizing term and batch size) should have been performed. Furthermore the model architecture could have likely been simplified and the space of 2, 3 and 4 layers should have been explored. It is further hypothesized that transformations beyond spectrograms and mfcc such as data augmentation ... i.e. including noise in the training data would yield a better performance in the real world (likely however at a loss on the test set due to changes through the data augmentation in the distribution). In the modelling pipeline (cdhi_model) we set the foundation for automatic experimentation

Deployment

1. To meet the specification under the given time frame we used the speech recognition demo in the tensorflow repository [3]
2. The model was converted to the tensorflow lite format and stored as conv_model_optimized.tflite.
3. Main changes were performed in the *app/src/main/java/org.tensorflow.lite.examples.speech/SpeechActivity file*. The sample rate as well as sample duration where adjusted to meet the specification of the training data. We further reshaped the floatInputBuffer, an array that holds the recordings of the microphone, to the input shape of the first layer of our CNN model.
4. Lastly we changed the layout in *app/src/main/res/layout/tfe_sc_activity_speech.xml*
5. The microphone continuously records in two second intervals and highlights the recognized box along with overlaying the prediction probability of the respective class.
6. The app was developed in Android Studio 3.6.1 and deployed in the emulator BlueStacks 4.160.10 on a Mac Mojave 10.14.5
7. Tests have shown that the model is not robust against environment noise. As noted in the modelling section with did not augment the dataset to make it more robust for real world setting. To get reliable predictions you may have to wait 5-10 seconds. We noticed that launching the application caused a high lowed on the CPU and the resulting fan activity caused for interference.



3: https://github.com/tensorflow/examples/blob/master/lite/examples/speech_commands/android/README.md

Appendix

Hyperparameters

batch_size = 64
epochs = 40
learning_rate = 0.000001

Regularizer l2 = 0.0001

Model architecture

Using Model M5 Model: "sequential_10"
_____ Layer (type)
Output Shape Param #
==========
conv1d_40 (Conv1D) (None, 4000, 128) 10368
_____
batch_normalization_40 (Batc (None, 4000, 128) 512
_____ activation_40
(Activation) (None, 4000, 128) 0
_____
max_pooling1d_40 (MaxPooling (None, 1000, 128) 0
_____ conv1d_41
(Conv1D) (None, 1000, 128) 49280
_____
batch_normalization_41 (Batc (None, 1000, 128) 512
_____ activation_41
(Activation) (None, 1000, 128) 0
_____
max_pooling1d_41 (MaxPooling (None, 250, 128) 0
_____ conv1d_42
(Conv1D) (None, 250, 256) 98560
_____
batch_normalization_42 (Batc (None, 250, 256) 1024
_____ activation_42
(Activation) (None, 250, 256) 0
_____
max_pooling1d_42 (MaxPooling (None, 62, 256) 0
_____ conv1d_43
(Conv1D) (None, 62, 512) 393728
_____
batch_normalization_43 (Batc (None, 62, 512) 2048
_____ activation_43
(Activation) (None, 62, 512) 0
_____
max_pooling1d_43 (MaxPooling (None, 15, 512) 0
_____ lambda_10
(Lambda) (None, 512) 0
_____ dense_10
(Dense) (None, 3) 1539
========== Total
params: 557,571 Trainable params: 555,523 Non-trainable params: 2,048