Machine Learning for IoT
Homework II

Lorenzo Melchionna
Politecnico di Torino
Data Science and Engineering
s304651@studenti.polito.it

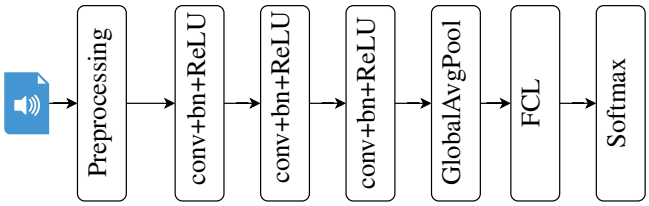
Edoardo Marchetti
Politecnico di Torino
Data Science and Engineering
s303873@studenti.polito.it

Edgar Gaytán
Politecnico di Torino
Data Science and Engineering
s300164@studenti.polito.it

***Abstract*—In this report, we describe how we trained, deployed, and optimized a Deep Learning model to classify 'Go' and 'Stop' words from an audio file to match given accuracy, latency and TFlite size constraints, and how we integrated it in a Voice User Interface for a Smart Battery Monitoring system.**

1. Model description

The base model used in this study is the one reported below:



Each convolutional layer is composed of 256 filters with a kernel size of 3x3. Moreover, the first convolutional layer has a stride of 2 while the second and the third have a stride of 1. Each training was performed with the following training parameters.

Batch_size	Intial_learning_rate	Final_learning_rate	Epochs
20	1e-2	1e-5	20

2. Training & Deployment of a "Go/Stop" Classifier

Before starting training, we had to select the best feature to find the best compromise between the latency and accuracy of a basic model. The available options were the audio *spectrogram*, the *log-mel spectrogram*, or the representation in *Mel Frequency Cepstrum Coefficients* (**MFCCs**). The final choice fell on the MFCCs representation as it includes two types of information that are (i) the glottal pulse and (ii) the spectral envelope. This first analysis was performed using the coefficients provided during the in-class laboratories.

The next step was the **hyperparameters tuning** for the extraction of the MFCCs. To reduce the search space, we analyzed the difference between the mean MFCCs of the 'go' audio and the mean MFCCs of the 'stop' audio and how it varies as function of a coefficient. Studying the impact of the num_mfccs_coefficients it was observed that, after the 10th coefficient, there is no longer any difference between a go and a stop (Fig. 1). For this reason, we decided to consider only the first 10 coefficients.

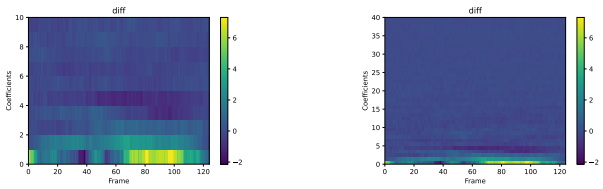


Figure 1. Comparison of the difference of the average mfcc for audio go and stop. On the left are the first 10 coefficients, and on the right all 40.

The same analysis was performed also to understand what is the best value for the *upper_frequency* (Fig. 2). It was found that using a higher frequency of 8000 Hz the *diff MFCCs* had a wider range. The wider range could be useful to better distinguish between the two categories of audio.

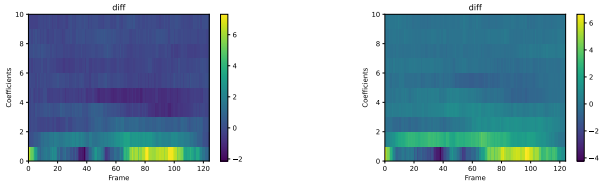


Figure 2. Comparison of the difference of the average mfcc for audio go and stop. The left one was computed using an upper frequency equal to 4000 Hz while the right one using an upper frequency of 8000 Hz.

Iterating the same reasoning for the other hyperparameters, we did not find a significant impact on *diff MFCCs*. We, therefore, decided to perform a grid search on the values given in table 1, using the model described above. The best combination found is:

- **downsampling_rate** = 16 kHz,
- **frame_length_in_s** = 0.016,
- **overlap** = 0%,
- **num_mel_bins** = 40,
- **lower_frequency** = 20 Hz,
- **upper_frequency** = 8000 Hz,
- **num_mfccs_coefficients** = 10

The results obtained are the ones reported in table 3 associated with *Baseline*.

TABLE 1. VALUES USED FOR THE GRID SEARCH OF THE HYPERPARAMETERS

Parameter	Values
lower_frequency (Hz)	[20,40,60,80]
overlap (%)	[0.0,0.25,0.5,0.75]
num_mel_bins	[10,20,30,40]

Once we found the best set of hyperparameters for preprocessing, we applied three different optimization techniques, trying to reduce the size of the model:

1) **Depthwise Separable Convolution (DSC)**: it allows for reducing the number of parameters for the convolution from $(k * k * C_{in} * C_{out})$ to $C_{in} * (k * k + C_{out})$;
2) **Width Scaling (WS)**: it allows to reduce the number of filters to a factor α. In our experiments we had considered $\alpha = 0.25$;
3) **Weights Pruning (WP)**: it allows to eliminate some weights and relative input/output connections. For our experiments, we applied it by exploiting a polynomial decay with initial sparsity set at 0.2 and final sparsity equal to 0.7.

We first applied the three techniques separately and then tried to combine them. The results are shown in table 3 and figure 3.

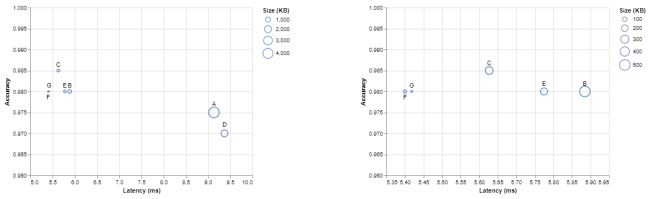


Figure 3. Pareto graph where the results of table 3 are visualized. On the left are reported all the models, while on the right is reported a zoom on the best ones.

As can be seen in table 3 all three techniques allow for reducing the final dimension of the model once it is zipped, but only DSC and WS also decrease the latency. This because the DSC and WS reduce the number of operations that the model has to perform, while the WP does not change the model's structure but it allows for better compression of the model.

TABLE 2. MODEL COMPARISON

Model Name	Label	Accuracy	Latency (ms)	Tflite Size (KB)
Baseline	A	0.975	9.14	4295
DSC	B	0.980	5.89	508
WS	C	0.985	5.72	273
WP	D	0.970	9.37	1811
DSC+WP	E	0.980	5.77	228
DSC+WS	F	0.980	5.40	39
DSC+WP+WS	**G**	**0.980**	**5.41**	**20**

3. Conclusion

The final model that we deploy has the architecture reported in figure 4. To reach the given constraints we had to combine all three techniques described in the previous section.

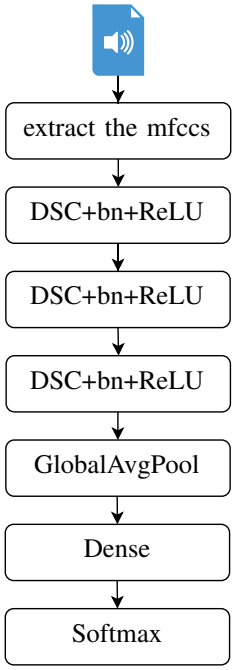


Figure 4. Final architecture. With respect to the initial one, now we have three layers that perform before a depthwise convolution and then a pointwise convolution, with α set to 0.25 and final sparsity 0.70.

TABLE 3. NUMBER OF PARAMETERS

	Total Params	Trainable params	Non-trainable params
Baseline	1,185,538	1,184,002	1,536
DSC+WS+WP	10,818	10,434	384
Reduction (%)	99.1	99.2	75

Thanks to the optimization process we were able to reduce the number of parameters of the baseline model to the 99%.