Machine Learning for IoT - Homework 2

Silva Bashllari
Student number: 299317

Davide di Mauro
Student number: 306089

Lal Akin
Student number: 300192

I. Choosing the best hyper-parameters

According to the requirements, our model must reach an accuracy of 0.97, as a functional metric, and for the extra-functional metrics: the total latency, a median of 8 ms and memory wise, the size of the TFLite smaller than 25 KB. One of the first decisions that we made was to determine the type of technique we would use. We decided on the Mel Frequency Cepstral Coefficients (MFCCs), due to its proven efficiency in automatic speech recognition or de-noising audio [1], [2], [3]. In addition, there naturally a trade-off to be considered between using the MFCC and STFT spectograms in terms of speed and quality. In our case, to meet the constraint on accuracy we used MFCC.
Regarding pre-processing hyper-parameters, we utilized some knowledge and rationale that we derived also from homework nr.1. Furthermore, we tried a greedy approach (Table I, Figure 1) and created a grid search function, testing the performance of different combinations of the hyper-parameters and came to the following conclusions (Table II, III, V):

- ***down_sampling rate***: we fixed its value to 16000 Hz as it proved to be optimal.
- ***frame_length_in_s***: we used power of 2 values due to the build-in tensorflow FFT implementation, that if the input is not in the power of 2 range, it will try to convert it to one, thus increasing the latency. Experimentally, we observed that choosing either 0.016 or 0.032 seconds makes no substantial difference in accuracy and latency.
- ***frame_step***: after experimenting with different values, it was shown that having overlapping didn't make a substantial difference in accuracy, thus we opted for having 0% overlapping in order to reduce latency as much as possible.
- ***num_mel_bins***: experimentally, it appears to have no impact on accuracy in our case, but latency wise, we think that taking a lower number of mel bins should improve latency, since you are dividing in less groups your spectrogram, and consequently reducing its size. Empirically is difficult to prove it due to the way we are computing latency in DeepNote. In our tests, the difference between 10 and 40 *num_mel_bins* is around 0.2 ms
- ***lower_freq*** & ***upper_freq***: we tested values between 20 and 80 Hz for the first, as suggested by the guidelines and between 2000 to 8000 Hz for the latter. Experimentally, it was shown that choosing 20 and 8000 Hz, respectively, worked better accuracy wise. This is reasonable because having a wider range will give us more information related to the signal. However, having a wider range may be a potential a pitfall latency-wise. In our case, this didn't really matter because even with this range, the latency constraint is met.
- ***num_coefficients***: the values considered were between 10 and 40. Considering that number of coefficients must be less or equal to the number of mel bins and that taking more then 10 mel bins had no benefits on accuracy, we take 10 MFCC's coefficients. Our choice is motivated by the fact that a lower number of mel bins (and consequently of coefficients) reduces the size of the input of our model, reducing the inference latency.

TABLE I
Hyper-parameters Grid

Hyper-parameters	**Values**
downsampling_rate	{**16000**}
frame_length_in_s	{0.016, **0.032**}
frame_step_in_s	{0.016, **0.032**, 0.064}
num_mel_bins	{**10**, 25, 40}
lower_frequency	{**20**, 80}
upper_frequency	{2000, 4000, **8000**}
num_coefficients	{**10**, 13, 40}
filters	{**128**}
alpha	{**0.5**, 0.75, 1, 2}
final_sparsity	{**0.70**, 0.80, 0.90}

TABLE II
Hyper-parameters - **Model**

batch_size	***initial_learning_rate***	***end_learning_rate***	***epochs***
20	0.01	1.e (-5)	10

II. Model architecture and optimization

We used the proposed model architecture during the labs and focused on the optimization instead. Trying to chose between the three proposed methods of: width scaling, DS convolution and weights pruning, we decided to use all three, opting for the best possible results. All the methods decrease the size of the model in different ways.

- **Width scaling**: is a uniform reduction of the number of channels in all convolutional layers, using a parameter α

TABLE III
HYPER-PARAMETERS - **PRE-PROCESSING**

downsampling_rate	***frame_length_in_s***	***frame_step_in_s***	***num_mel_bins***	***lower_frequency***	***upper_frequency***	***num_coefficients***
16000	0.032	0.032	10	20	8000	10

TABLE IV
FINAL RESULTS

original_size	***tflite_size***	***compressed_size***	***preprocessing_latency***	***model_latency***	***total_latency***	***original_accuracy***	***tflite_accuracy***
223.33 KB	44.02 KB	**20.64 KB**	4.64 ms	0.11 ms	**4.75 ms**	0.990	**0.985**

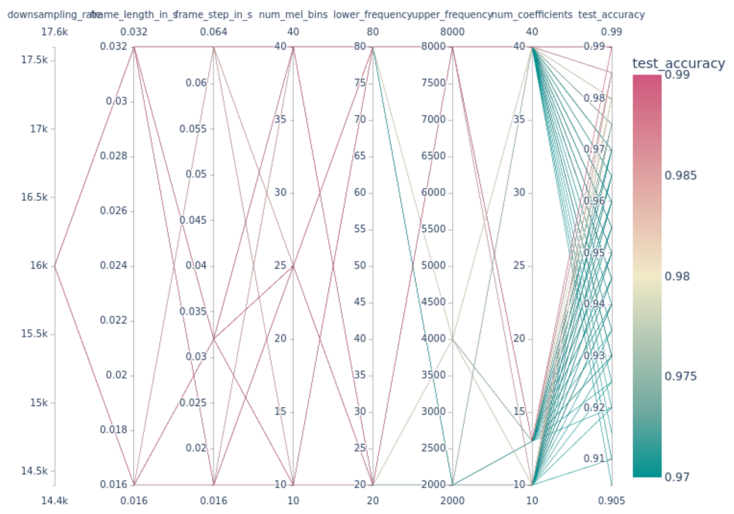

Fig. 1. Parallel Coordinates Graph

TABLE V
HYPER-PARAMETERS - **OPTIMIZATION**

filters	***alpha***	***final_sparsity***
128	0.5	0.7

and eventually also reducing the number of weights of the final fully connected layers. In the actual implementation, we declare a hyper-parameter α and then multiply it with the number of filters in each convolutional layer. This technique decreases the inference time at the cost of accuracy.

- **Depthwise Separable Convolutions**: This method consists on splitting the traditional convolution in two distinct operations: the depthwise convolution and the pointwise convolution. In the first one the kernel iterates over one single channel at a time instead of all of them. Then, the pointwise convolution uses a kernel of size 1×1 on all channels to cover the depth dimension (i.e. combine the different channels). Doing this, we reduce significantly the number of computations needed to perform each convolution in our model, more precisely by a factor $R = \frac{1}{C_{out}} + \frac{1}{K^2}$. Thanks to this, we are able to reduce both the size of the model and the latency (since we do less multiplications), while slightly, decreasing the accuracy.
- **Weights Pruning**: This technique exploits the redundancy of DL models, and tries to eliminate some weights and the corresponding input/output connections. With Magnitude-based weight pruning we gradually zero out model weights during the training process to achieve model sparsity. We will be able to observe a reduction in the size of the stored model **only when we compress it**, since a sparse matrix can be stored more efficiently than a normal one. Latency doesn't improve since we set the values of the weights to 0, but the number of computations remains the same; with dedicated hardware it is possible to skip those multiplications, improving also latency during inference.

III. CONCLUSIONS

In conclusion, we demonstrated how we were able to train and optimize a model respecting all the constraints (Table IV). In terms of pre-processing parameters, we noticed that the ones impacting accuracy the most where: *frame_lenght_in_s*, *frame_step_in_s* and *upper_frequency*. In terms of optimization techniques, we decided to use all three of them in order to met the requirements. We did attempt to try not to use all of them, but different two-by-two combinations, hoping to not impact accuracy too much but that would lead in a too of large size of the model. During our tests, we where able to find different hyper-parameters configurations that satisfied all constraints, but we choose the one that performed better accuracy and latency wise.

REFERENCES

[1] C. Ittichaichareon, S. Suksri, and T. Yingthawornsuk, "Speech recognition using mfcc," in *International conference on computer graphics, simulation and modeling*, vol. 9, 2012.

[2] N. Dave, "Feature extraction methods lpc, plp and mfcc in speech recognition," *International journal for advance research in engineering and technology*, vol. 1, no. 6, pp. 1–4, 2013.

[3] Mlearnere, "Learning from audio: The mel scale, mel spectrograms, and mel frequency cepstral coefficients," Apr 2021. [Online]. Available: https://towardsdatascience.com/learning-from-audio-the-mel-scale-mel-spectrograms-and-mel-frequency-cepstral-coefficients-f5752b6324a8