Homework 2 - Group 11

Feraud Elisa (s295573), Scalenghe Luca (s303452), Tchouamou Pangop Christelle Elsa (s295753)

Machine Learning For IoT, Politecnico di Torino, A.A.2022-2023

1 Exercise 1.3 Reporting

The purpose of this homework is to define an optimization model in such a way to satisfy the required constraints. To do this, we focused on some optimization techniques in order to understand firstly their singular behaviours and then how to combine them together. Firstly, we proposed the implementations of four different models in order to give us a better feeling of what could have worked in the later stages. The first model was our baseline architecture: a three layer CNN with a filter size of 256. Using this model we tried to understand which preprocessing technique was better and we compared the Log-Mel Spectrogram with the MFCCs representation. We trained the base models feeding them with the two different representations, training for 10 epochs with exactly the same parameters (also the same that are used in the final solution) and obtained an accuracy of 0.8950 for Log-Mel Spectrogram and of 0.9650 for MFCCs on the test set. As suggested by these results we decided to use MFCCs in order to do the preprocessing of the audio signals. Specifically, MFCCs stands for Mel-Frequency Cepstral Coefficients. In literature MFCCs are often chosen for speech recognition applications because they capture the spectral characteristics of speech in a compact and efficient way. They are relatively insensitive to the specific characteristics of the speaker's voice, such as their pitch and timbre. This means that MFCCs can be used to represent the content of the speech, rather than the specific characteristics of the speaker. Unfortunately, MFCCs are sensitive to the noise, thus in order to obtain better performances, it may be useful normalizing the audio signal values, but we did not explore this route. Therefore, we defined three optimizations: the Weight pruning CNN model, the Depth-wise CNN model and the Width pruning CNN model. We implemented each of them on the base model described above.
The *Width pruning CNN model* is the easiest implementation and also conceptually very easy to understand. It is a reduction of the filter size. Practically, the implementation consists of a multiplication of a parameter alpha in the second and third layer of the convolution. This alpha can be then tuned to exploit the best architecture.
The *Weight pruning CNN model* uses the model optimization library of Tensorflow *tfmot.sparsity.keras.prune_low_magnitude* to set up a model that can be pruned during training, this means that it sparsifies the layer's weights during training putting them to zero. It produced only compression gains while the accuracy and the total latency did not improve. Only using pruning did not give a smaller model size but by using the zip compression, which is not lossy, we were able to exploit the model sparsity, reduce its dimension and preserve the performance. In the *Depth-wise separable convolution* the filters are applied separately to each channel of the input tensor, rather jointly across all channels. By doing in this way, the model learns separate filters for each channel. After that it includes a point-wise convolution, which is a standard convolution operation to combine the output channels into a single tensor. The point-wise convolution allows the model to learn relationships between the channels produced by the depth-wise convolution. This approach significantly reduced the number of parameters in the model and made it more efficient to compute. The importance of this implementation was immediately clear to us, since with no extra cost it is possible to reduce the number of operations by a factor of $\frac{1}{C_{out}} + \frac{1}{k^2}$ (where C_{out} is the output size and k is the filter size), in such a way to have a free reduction in parameter size. The Depth-wise CNN reduced latency and memory but it did not improve the accuracy.
Therefore, all the singular models developed until here presented a good total latency, but the constraints regarding the accuracy and the TF lite size were not satisfied. We decided to exploit all the benefits of each model by defining a "mixed" model composed by the three optimizations. With a small grid search (reported below) we were able to select the correct hyper-parameters in order to be compliant with the constraints.

filters	alpha	final_sparsity	frame_len	frame_step	epochs	accuracy	TF lite size zipped (KB)	Tot Latency (ms)
256	0,6	0,8	0,032	0,032	30	95%	70,547	5,2
256	0,6	0,7	0,032	0,032	30	96%	89,007	5,2
256	0,5	0,8	0,032	0,032	30	96%	51,785	5,2
256	0,4	0,8	0,032	0,032	30	96,50%	22,386	5,1
256	0,3	0,8	0,016	0,016	30	98,50%	22,357	5,5
256	0,2	0,8	0,016	0,016	30	96,50%	12,557	4,9

Figure 1

The grid search was performed using the hyperparameters in Figure 2 and 3 with exception of the frame_length_in_s and

frame_step_in_s. The sparsification in combination with the model zipping gave a good reduction with a minimal loss in accuracy, but the key to achieving the specified size was the alpha parameter that reduced drastically the size of the layers.

downsampling_rate	frame_length_in_s	frame_step_in_s	lower_frequency	upper_frequency	num_mel_bins	num_coefficients
1600	0,016	0,016	20	4000	40	10

Figure 2: The preprocessing hyperparameters used in the final solution.

batch_size	initial_learning_rate	end-learning_rate	epochs
20	0,01	1.e-5	30

Figure 3: The training hyperparameters used in the final solution.

Accuracy	TF lite Size zipped(KB)	Tot Latency (ms)
98,50%	22,357	5,5

Figure 4: Performances of the final solution.