Machine Learning for IoT
Homework 2 - Group 8

Rosi Gabriele s291082 | Fabio Tecco s285569 | Merlo Matteo s287576

1 Multi-Step Temperature and Humidity Forecasting

The first optimization script aims to train a multi-output models for temperature and humidity forecasting to infer multi-step predictions. The models used in both version are trained with the following structure:

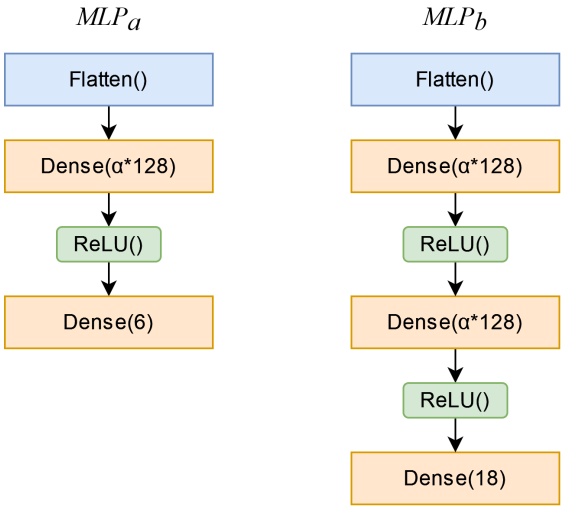

Figure 1: Models used for version A and B

- **Version A** A Multilayer Perceptron (MLP) is chosen as architecture since it has proven to be more suitable for regression task as suggested in Lab03. Since a huge reduction of the model size is needed, we combined three different optimization techniques: *structured pruning* via width scaling, *magnitude base pruning* with polynomial decay and *post training quantization*. In the *Structured pruning* we choose a very small width multiplier α, thus reducing the number of units of each Dense layer. This reduces a lot the model size but have some drawback on the loss. Then, *magnitude-based pruning* with polynomial decay is applied to the model: after each training step, we remove the links with the smallest weights, improving both the size (reducing it) and the accuracy (increasing it). Subsequently, the most suitable optimization pipeline, that allows us to reduce size without too much loss in accuracy, is the *magnitude-based pruning* followed by a 16-bits float weights-only PTQ (thus weights are stored as float16, instead of float32). Finally, in order to reduce the model size, we compressed them through *zlib*.
- **Version B** The increase in the output steps may influence the performance of the model, leading us to increase the number of layer in the network. The optimizations used are the same for version A. All the hyperparameter and results are displayed in the Table 1[1].

[1]The python version used for testing is 3.8, all other package with the version in requirements.txt, as requested

Version	Architecture	Optimization			Training time	Results		
		Structured Pruning	Quantization	Magnitude based pruning	Epochs of training	MAE (T)	MAE (H)	TfLite size [kB]
A	MLP	α=0.2	weights-only	init spars: 0.30 final spars: 0.85 init step: 5 final step: 15	30	0.295	1.183	1.455
B	MLP	α=0.1	weights-only	init spars: 0.30 final spars: 0.85 init step: 5 final step: 15	20	0.666	2.333	1.634

Table 1: Hyperparameters and results for each version

2 Keyword Spotting

The goal of the script is to train models for keyword spotting on the original Mini Speech Command Dataset, keeping track not only of accuracy and size constraints but also latency. We used the DS-CNN suggested in Lab03 instead of CNN or MLP, since it is better in both accuracy and size. The networks used in all version is illustrated in Figure 2. In this exercise *Structured Pruning and Post-training Quantization* have been used in all the versions. The parameters (alpha, number of epoch, LR) have once again been found through a grid search. The starting learning rate in version A is 0.1 divided by 4 every 5 epochs and in version B is 0.01 remaining constant through all epochs. A combination of optimization techniques is used both for the model and the audio files. In fact, in order to reach the constraints in B and C we performed a resampling phase to decrease the total latency of the preprocessing step and as a consequence also the size of the model. For all the versions we used MFCC approximation because of the need of a high accuracy of the model.

The total latency then is measured on a Raspberry Pi 4 Model B with 4GB RAM (the command line used are below) using the following commands:

```
B) python3 kws_latency.py --mfcc --model Group8_kws_b.tflite --
   rate 8000 --length 240 --stride 120
C) python3 kws_latency.py --mfcc --model Group8_kws_c.tflite --
   rate 8000 --length 240 --stride 120
```

Version	Preprocessing	Architecture	Optimization		Training time	Results		
	MFCC		Structured Pruning	Quantization	Epochs of training	Total latency [ms]	Accuracy [%]	TfLite size [kB]
A	Default	DS-CNN	α=0.85	weights-only	25	/	93.25	86.60
B	sampling Rate: 8000 frame length: 240 frame step: 120	DS-CNN	α=0.25	weights-only	25	33	92.0	18.076
C	sampling Rate: 8000 frame length: 240 frame step: 120	DS-CNN	α=0.25	weights-only	25	33	92.0	18.076

Table 2: Hyperparameters and results for each version

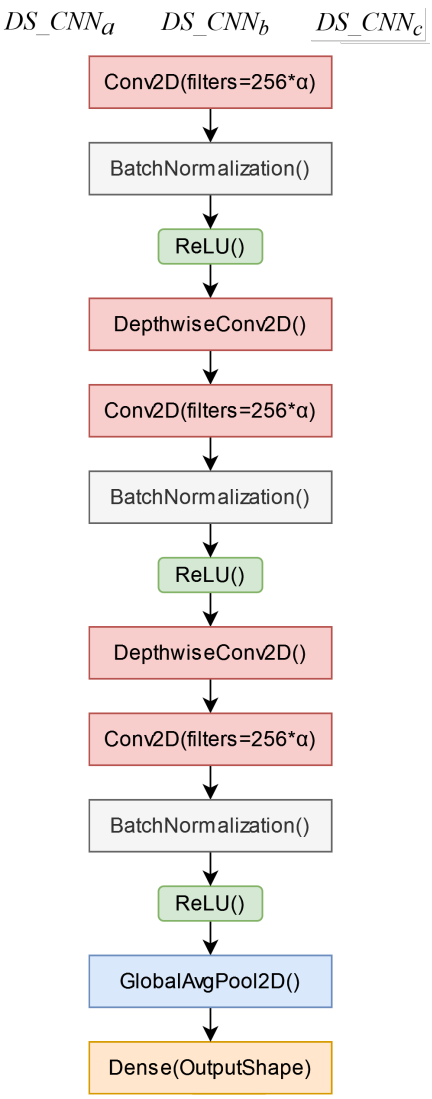

Figure 2: Network architecture used for each version