HW2:

Machine Learning for IOT

Group 2

Dri Emanuele s280133

Malan Erich s267475

Nicolì Alessandro s278091

Exercise 1

Objective: design a Python script to train multi-output models for temperature and humidity forecasting to infer multi-step predictions. The number of output steps will be 6. Two different model versions are requested, each one meeting the following constraints, respectively:
Version a): T MAE < 0.5°C, Rh MAE < 1.8% and TFLite Size < 2 kB
Version b): T MAE < 0.6 °C, Rh MAE < 1.9% and TFLite Size < 1.7 kB

Execution: we used the *Jena Climate Dataset* with a 70%/20%/10% split. The first step was to slightly modify the `WindowGenerator` and `MultiOutputMAE` classes, defined in the previous labs, in order to have 6 output steps. Then looking also at the results of the already mentioned labs, we decided to try firstly the 1-dimensional CNN which proved to be indeed the best model among the three. Regarding the optimization part we tried combinations of three different methods: *Weights-Only Post Training Quantization, Structured Pruning via Width Scaling* and *Magnitude-Based Pruning*. By far the better results were obtained by setting a parametric width below 0.1, at this point too few units were remaining in our layers to perform effectively the PT Quantization, so our last tests regarded the magnitude-based pruning. We observed that, to respect all the constraints, we needed to set a low final sparsity (es. 0.30) but by doing this, the benefits in terms of space were offset by the headers introduced by this type of pruning (see table 1). Therefore at the end our final submission makes use only of the structured pruning with an optimal width parameter that we found to be 0.05 for both versions.

Exercise 2

Objective: design a Python script to train models for keyword spotting. The script must be able to train three different versions, each one meeting the following constraints, respectively:
Version a): Accuracy > 90% and TFLite Size < 25 kB
Version b): Accuracy > 90% and TFLite Size < 35 kB and Inference Latency < 1.5 ms
Version c): Accuracy > 90% and TFLite Size < 45 kB and Total Latency < 40 ms

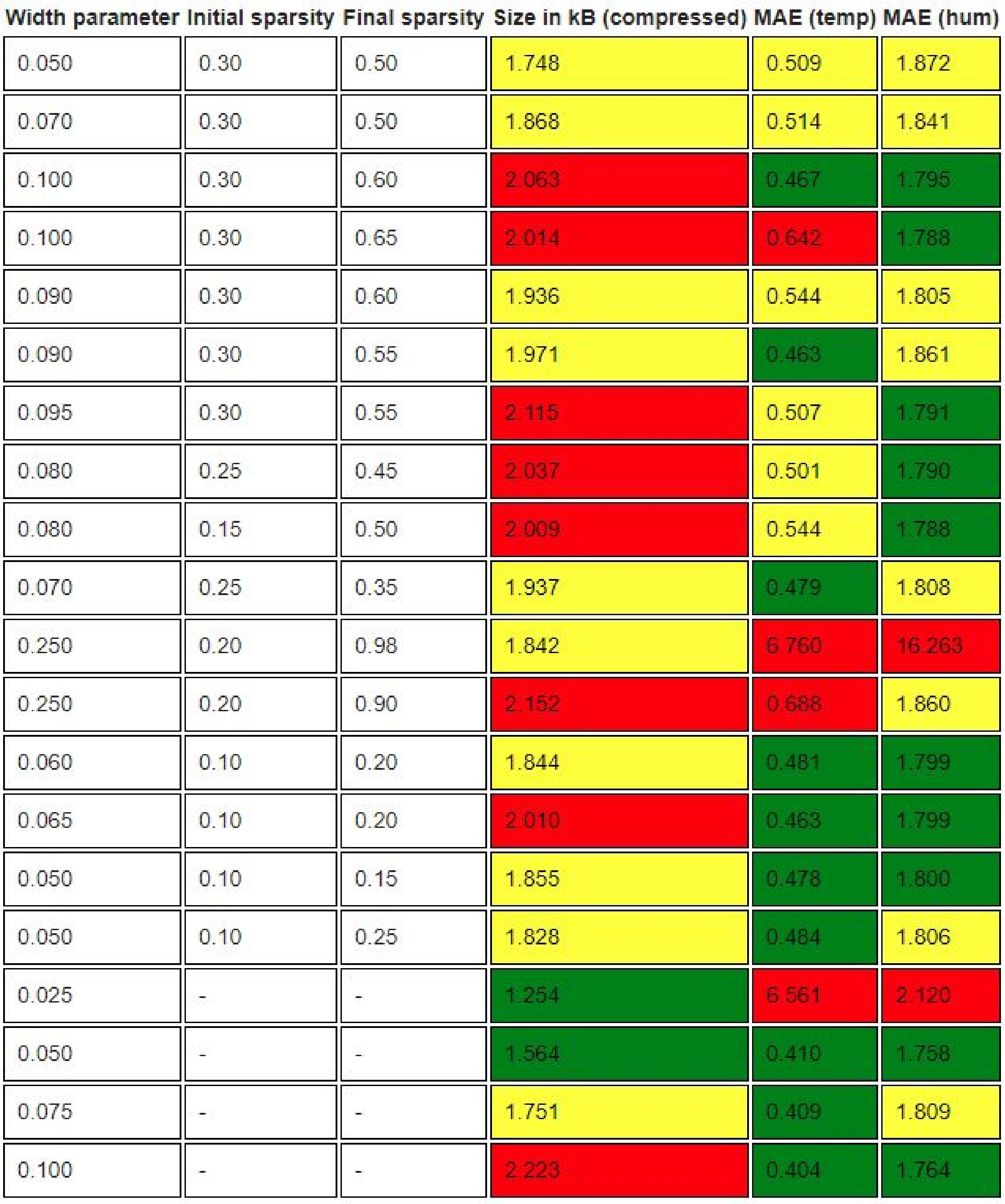
Execution: we used the *Mini Speech Command Dataset* and it was splitted according to the provided splits. We modified the `SignalGenerator` class, defined in the previous labs, only in the *read* and __*init*__ methods since we added a resampling option. We used MFCCs for every version. We used a slightly modified version of the DS-CNN that has a dropout after every batch normalization and after the global average pool. It also has 128 filters in every Conv2D layer instead of 256. For every version we train and save the best model before optimization. After this, the execution of the script depends on the version of the model.
Version a and b): The number of MFCCs coefficients is 8 instead of the default 10 and this is the only parameter that we changed. We load the already trained model and we retrain it while using a polynomial decay as pruning method with a final sparsity of 0.7. After the training, we optimize the size using the default optimization given by tflite and compress the file using zlib. In this way the final model has 91% of accuracy on the test set, its size is 22.7 kB and the inference latency on the Raspberry Pi is 1.39 ms. These characteristics are good for both version a and b, so we can use this model for both purposes.
For the version b an additional interesting result has been found without dropout, using masking layers after ReLUs, no pruning and width_scaler = 0.2, the training time was 4x faster but the final result was a little worse.
Version c): For this version we modified three parameters: sample rate equal to 8000 instead of 16000, frame length equal to 320 instead of 640, frame step (stride) of 160 instead of 320. In this case we optimize the model without pruning it and we save the model without the compression since its size is 44,8 kB which already satisfies the requirement. With this model we got 91% accuracy on the test set and the total latency is 38 ms on average and, with this, we satisfy the requirements for version c.

Note: the labels are alphabetically ordered before training so the labels indexes are always the same when new models are created.
```
['down' 'go' 'left' 'no' 'right' 'stop' 'up' 'yes']
```
To run kws_inference.py correctly, use the following commands:

Version a, b
```
kws_inference.py --model Group2_kws_a.tflite --coeff 8 --mfcc
kws_inference.py --model Group2_kws_b.tflite --coeff 8 --mfcc
```

Version c
```
kws_inference.py --model Group2_kws_c.tflite --length 320 --stride 160 --mfcc
--rate 8000
```

Table 1: results for exercise one (green = satisfy constraint for both versions)



Width parameter	Initial sparsity	Final sparsity	Size in kB (compressed)	MAE (temp)	MAE (hum)
0.050	0.30	0.50	1.748	0.509	1.872
0.070	0.30	0.50	1.868	0.514	1.841
0.100	0.30	0.60	2.063	0.467	1.795
0.100	0.30	0.65	2.014	0.642	1.788
0.090	0.30	0.60	1.936	0.544	1.805
0.090	0.30	0.55	1.971	0.463	1.861
0.095	0.30	0.55	2.115	0.507	1.791
0.080	0.25	0.45	2.037	0.501	1.790
0.080	0.15	0.50	2.009	0.544	1.788
0.070	0.25	0.35	1.937	0.479	1.808
0.250	0.20	0.98	1.842	6.760	16.263
0.250	0.20	0.90	2.152	0.688	1.860
0.060	0.10	0.20	1.844	0.481	1.799
0.065	0.10	0.20	2.010	0.463	1.799
0.050	0.10	0.15	1.855	0.478	1.800
0.050	0.10	0.25	1.828	0.484	1.806
0.025	-	-	1.254	6.561	2.120
0.050	-	-	1.564	0.410	1.758
0.075	-	-	1.751	0.409	1.809
0.100	-	-	2.223	0.404	1.764