Machine Learning for IoT - Politecnico di Torino

Homework 2 report

Paolo Aberto
StudentID : s278098

Lorenzo De Nisi
Student ID: s276545

Carmine De Stefano
Student ID: s278176

1 Multi-Step Temperature and Humidity Forecasting

After implementing a WindowGenerator class returning 6 output steps, we started to train the models. We settled on a weights-only quantization, and after tweaking the pruning parameters, we found two versions of the CNN model that were able to respect the given constraints. The parameters changed in the 2 models were the final sparsity (%) at which pruning ends, and the width multiplier α.

We adopted a callback function to stop the training when the validation set performances were not improving anymore. We adopted the standars Keras checkpoint approach, and we monitored the mean of the two MAE metrics for the checkpoint.

In the following table we can see the results for the two models, that were both trained for 20 epochs, with learning rate 0.001:

Model	**Quantization**	**Final sparsity**	α	**Size**	**Compressed size**	**Temp MAE**	**Hum MAE**
A	Weights only	0.745	0.12	5246 B	1992 B	0.4812 °C	1.786 %
B	Weights only	0.75	0.07	4480 B	1689 B	0.4795 °C	1.879 %

Table 1: Metrics for the final models obtained (version A and B)

2 Keyword Spotting

For keyword spotting on the mini speech command dataset, we used the DS-CNN model with weights-only quantization, using the following hyperparameters:

Model	**Quantization**	**MFCC**	**Final sparsity**	α	**Epochs**	**Learning Rate**
A	Weights only	YES	0.9	0.7	20	0.0075
B	Weights only	YES	-	0.3	30	0.005
C	Weights only	NO	-	0.4	30	0.01

Table 2: Hyperparameters for the final models (version A, B and C)

Like before, we adopted a callback function to monitor the accuracy metric on the validation set. In order to obtain the performances requested in version B, we used a quite low parameter α, to reduce inference computation. For version C instead, we switched to STFT in order to dramatically reduce processing time. STFT and MFCC were computed with the same parameter used in the labs.

The final results are shown in the table:

Model	**Size**	**Compr. size**	**Inference Latency**	**Total Latency**	**Test Accuracy**
A	77248 B	21764 B	9.03ms	66.71ms	0.91375
B	26960 B	23233 B	1.16ms	58.72ms	0.91
C	40320 B	34831 B	4.52ms	21.07ms	0.9175

Table 3: Metrics for the final models (version A, B and C)

We noticed that on different OS or different PCs, the way python lists the directories changes. This means that the order of the labels could change if it is obtained directly from the directory names. To solve this issue, the order was fixed with a list.
For both models, compression was performed thanks to the zlib library. Final memory performances were calculated by considering the compressed model. For both tasks, we used the original architectures provided in the laboratory.