Machine Learning for IoT - Homework 2

Luca Campana
Student id: s290065

Gianvito Liturri
Student id: s290464

Alberto Morcavallo
Student id: s290086

EXERCISE 1: MULTI-STEP TEMPERATURE AND HUMIDITY FORECASTING

For the first task, the first thing to do is to generate the appropriate transformation over the *Jena Climate Dataset*, obtaining both samples and labels. This is done by opportunely modifying the *WindowGenerator* class. Then, for the actual training phase, we referred to models discussed in Lab3, i.e. Fully Connected, CNN, and LSTM. Since we found out that most of the optimization tools we have are not suited for the latter model, we decided to carry on the study only for the first two architectures, training them properly with *adaptive LR* and *early stopping*. Regarding the methodology adopted to decrease model sizes, the approach was the following: firstly, we tried to get models that were able to perform the task with the lowest possible number of parameters; then, we went on trying optimizations like *PTQ*, *Structured Pruning* and *Magnitude-based Pruning*, at various levels of intensity, to find the better combination of model size and MAEs. For both the versions, we found out that Dense layers responded to optimizations in a much better way respect to convolutions, so we decided to focus on FC architectures, obtaining the following results.

Version a: 3-step forecasting

For this version, the architecture designed is composed of two FC layers, respectively of 16 and 8 neurons, other than the final *Dense(6)*: in total, 398 parameters. This is then structurally pruned, with a final sparsity of 80%, and weights are quantized into the *Int8* format. The model, compressed in *zlib* format, has a size of $1.45kB$, and gets a MAE of $0.27°C$ for temperature and 1.16% for humidity, over the full test set.

Version b: 9-step forecasting

In this case, the architecture is the same of the previous one, with the only difference of the final layer, a *Dense(18)*, for a total amount of 506 parameters. To meet the constraints about size, the pruning process is pushed up to 85%. Also in this case, *Int8*-PTQ and compression are performed, and the resulting model achieves a MAE of $0.63°C$ and 2.31%, with a size of $1.51kB$.

EXERCISE 2: KEYWORD SPOTTING

For the second task, instead, we firstly have to generate, from the *mini speech command* dataset, samples that could be preprocessed, and submitted to algorithms, in less than $40ms$. By practical simulations, we found out that this solution was not achievable by maintaining the native sampling rate, therefore the only feasible way to obtain the samples is through track undersampling to $8kHz$. To keep the dimensionality steady, also parameters about window lengths were halved.

Following the results of previous labs, the analysis is carried on *MFCCs*, that have been shown to be not only better for the model final quality, but also less dimensional: in fact, the preprocessing of a single $1s$ audio track yields a matrix of shape $[49, 10]$. Moreover, *MFCCs* are fairly better than spectrograms also in filtering out noise components[1].

Provided with the previous labs model, we applied inferences on the input data: we discarded MLP solutions as they were not able to reach the needed accuracy standards, and CNN ones, since they got roughly the same performance of DS-CNNs, but involving much more parameters. So, once we decided to focus on the DS-CNN model involving three convolutional layers of 258 $[3 \times 3]$ filters each plus one final *Dense(8)*, we carried on with the same approach of the previous exercise, with the exception that this time the only optimizations performed are *PTQ* and *Structured Pruning* via *Width scaling*. This time, *Magnituded Based Pruning* is not needed, since the dimension constraints are easier to obtain with respect to the previous cases. This procedure allows us to complete the exercise, with the following results.

Version a

Here, we apply *Width scaling* with $\alpha = 0.75$ over convolutions: as a result, our model is composed of 82760 parameters. After performing *Int8*-PTQ, the model *.tflite* size is reduced to $86.9kB$, and the accuracy over the test set is equal to 93%.

Version b

In this case, to meet the size constraint we must increase α up to 0.5: the resulting architecture has a total of 38792 parameters. Also in this case this was followed by *Int8*-PTQ: final model size is $43.5kB$, the accuracy is 94.5%, and the total latency is about $39ms$.

Version c

For this last task, the considerations are roughly the same as the previous one, but with $\alpha = 0.25$. The resulting architecture is made of 11208 parameters, and weighs $22.5kB$. Its performances are 92.7% accuracy and $37ms$ total latency.

[1] for the exposed reasons, the command to run to evaluate the latency (for versions *b* and *c*) is: `python kws_latency.py --rate 8000 --mfcc --length 320 --stride 160 --model model.tflite`