Machine Learning for IoT – HW2
Davide Napolitano, Edoardo Lardizzone – Group20

The target of the first exercise is to predict the values of temperature and humidity of 6 consecutive moments of time (e.g. days) given the values of the 6 previous moments.
For the two versions of Ex1 we start from the model presented in the laboratory, first of all we change the Window Generator class in order to have a 6x2 output and we create a class that is able to compute the MAE on a multi-output prediction. This is done by taking *sequence_lenght*=12 such that in the first 6 values are taken as input and the remaining 6 as labels. Instead for the MAE the values are reduced before on dim 1 and then on dim 0 with the *.reduce_mean()*, i.e. [32,6,2]->[32,2]->[2]. Starting from the two model (CNN and MLP) we notice that the dimension of the created file are always very high with respect to the constraint we have to respect; in order to slightly decrease the size we apply some techniques seen during the course: first of all we apply the pruning (*Polynomial Decay* with *initial_sparsity*:0.3, *final_sparsity*:0.9, *begin_step*:len(train_ds)*5, *end_step*:len(train_ds)*15 + *LearningRateCallback* as described below) to a model trained for 20 epoch in order to lose the parameters that are not essential for our predictions, applying only the pruning is not enough to be under 2 kB so we also apply the quantization to int8 that is able to led us to the needed dimensions, these method were applied identically on both models. Other techniques like Quantization for Weights+Activations(on MLP), Quantization Aware(on Dense Layers) and Weights Clustering(on Dense layers) have been tried where possible, however none of them provides good results both applied alone both combined with others.
For both the models we use the hyperparameters presented in the laboratory (*optimizer*:adam, *loss*:MSE, *metric*:our_MAE, *epochs*:20). The only modification is that we insert the *LearningRateCallback* callback that decreases the learning rate by a factor *exp(-0.1)* after the 12th epoch because we notice that the model is more stable by doing this. For the version **a** we use a CNN with the same layers of the one presented in Lab3, the difference is that we applied a structured pruning, decreasing the number of filters for the convolutional and the dense layer finding that the constraint were respected using a CNN with a Conv1D layer with 16 filters,a Dense layer with 24 layers and final Dense of 12 reshaped to (6,2), the dimension reached after the zipping of the file (essential in this case) was of **2017 Bytes**, which is under the requested 2048 and having as MAE on the two measurements **[0.4502,1.7888]**. For the version **b** we use a MLP with the same layers of the one presented in the Lab3, same as for version **a** we applied a structured pruning that led us to have the same values of version a for the filters (16,24 and 12 reshaped as before). The aforementioned procedure let us have a TfLite file, zipped again, with a dimension of **1734 Bytes**, under the constraint of 1740, that had a MAE of **[0.4739,1.8009]**.

The target for the second exercise is a typical Keyword spotting problem. For all the versions in the second exercise we start from the Lab3_Ex4 and we add the constraint for the splits by reading the *.txt* file with pandas and then by passing them to the make_dataset method.
Let's start with version **a** and **b,** the class *SignalGenerator* has not been touched, so to make the dataset we used the same hyperparameters as in lab3 (*sampling_rate*=16000, *frame_length*=640, *frame_step*=320, *num_mel_bins*=40, *lower_frequency*=20, *upper_frequency*=4000, *num_coefficients*=321).
About the choice of the preprocessing, we go with *mfcc* since with *stft* we aren't able to achieve the constraint for accuracy. For the model we use a DS-CNN as described in the presentation of lab3, we only change the number of filters, respectively 96-64-64 for the Conv2D layer, in order to obtain an accuracy > 90% (91.24% on the test) with the smallest possible initial model size (other combination of #filters have been tried, however by going below the previous combination we aren't able to get accuracy > 90%).
The model is compiled with the same hyperparameters of lab3 (*optimizer*=adam, *loss*=SparseCategoricalCrossentropy, *metric*=SparseCategoricalAccuracy, *epochs*=20), while for the fit we add the *ModelCheckpoint* callback on the *val_sparse_categorical_accuracy* in order to have at the end the model that performed better on the validation set.
After the standard train we try different approaches to reduce the size of the model, however we see that, by applying only the quantization to int8 and then by zipping the model, the size constraint is respected: 242757B(Std model) -> 28016B(Lite model) -> **22810 Bytes(Zip model)**. The final tflite model provided a score equal to **91.23%**.
This workflow is used to build the model for version **a,** however by testing the same model to check the accuracy we obtain an *Inference Latency* about **1.08±0.02ms**. So the model is fine also for version **b** and this can be explained by the fact that our model is built with few layers and overall with few parameters, reducing consequently the inference latency. Instead for version **c** we try different solutions, in particular from the *kws_inference.py* we see how the total latency time with *stft* is about 20ms, however by trying different NN architectures with *stft* and by combining them with different hyperparameters and dimensionality technique, we aren't able to be compliant with all the constraints at the same time (in particular with the size, since we have found that only big models are able to obtain an acc > 90%, however we don't discard the feasibility to do it with *stft*). The other solution seen, that is indeed the one we use, is done by adopting *mfcc* and then by resampling the audio to 8000 Hz (other combination of hyper for the inference time are tried, like reducing the *#bins*, but they provide at most a gain of 4/5ms) which lead to less parameters to compute for the *stft* and as consequence for the *mfcc*.

So to apply the downsample, inside the *pad* function of the class *SignalGenerator* we add a call to the numpy function that does the resample (*signal.resample_poly()* + *np.float32* cast) with the *.numpy_function()* of tensorflow.
The hyperparameters to generate the class are changed as a consequence: frame_length: 640->**<u>320</u>**; frame_step: 320->**<u>160;</u>** num_coefficients: 321->**<u>161.</u>** The NN is the same as before, it is only added a block of Depth-Conv2D(96)-BatchNorm-ReLu after the initial one. All the other steps are the same as for **<u>a</u>** and **<u>b,</u>** excluded that now the model is trained for 30 epochs. The final size is of **<u>42400 Bytes</u>**(no zip) with an accuracy on the tflite of **<u>90.73%</u>** and a *Total Latency* time equal to **<u>38.5±0.7ms,</u>** (hyper for *kws_inference.py*: *mfcc*, *rate*:8000, *length*: 320, *stride*: 160).