Homework 2

GROUP 17

Juan Aragon (s291466) Hadi Nejabat (s246601) Rostislav Timuta (s280238)

I. Multi-Step Temperature and Humidity Forecasting

Model	α	Sparsity	Compressed File size (kB)	T MAE (°)	Rh MAE (%)
A	0.04	0.90	1.20	0.26	1.20
B	0.04	0.90	1.72	0.64	2.46

TABLE I: Results ex 1

The goal of this exercise was to implement 2 multi-output models for temperature and humidity forecasting, to infer multi-step predictions while satisfying some constraints regarding the output's accuracy and memory consumption. Both models were trained, validated and tested using the Jena Climate Dataset[1], with a 70%/20%/10% split ratio. Initially, in both versions it was considered the standard models introduced on lab3: *MLP* and *CNN*. In general, it was proven that *CNN* outperformed *MLP* for the given constraints, since it was capable of retrieving more accurate results even with significant reductions on its number of filters (less memory space).

For the first model, a modified version of the original *CNN* was implemented, by removing the 1D-Convolutional layer and decreasing the number of filters by a factor of α, so the total amount of filters in the Dense layer was reduced from 64 units into 2 units. For the second model, all layers were considered as in the standard form, but the number of available filters was also modified by using the same value for the parameter α.

Then, in order to increase the sparsity, it was used the approach of *magnitude-based pruning*, with a *PolinomialDecay* to schedule how much sparsity to enforce in a training step. In both cases the sparsity started from an initial value of 0.3 and acheived a final value of 0.9.

After training our model in float32 we performed a *F16 quantization* in TFlite, the weights were converted to float16. In the end, to see the model size advantage due to pruning we compress our model with *zlib*.

From the TABLE I we can observe the different hyperparameters chosen and the results obtained for the two model versions.

[1]https://storage.googleapis.com/tensorflow/tf-keras-datasets/jena_climate_2009_2016.csv.zip

II. Keyword Spotting

Model	α	Compressed File size (kB)	Accuracy(%)	Latency (ms)
A	0.25	94.84	0.921	-
B	0.23	47.07	0.915	23.96
C	0.15	23.36	0.916	22.90

The purpose of this exercise was to train three different models for keyword spotting on the mini speech command dataset[2] seen on lab 3. The models were required to meet some constraints regarding the output accuracy, model size and the latency measured on the edge device.

For all three models it was chosen to apply the MFCC algorithm at the preprocessing step since it offered a higher accuracy than the STFT version. For the MFCC parameters, it was noticed that by using *frame_length = 256*, *frame_step = 128* and the rest of the default values[3], the latency constraints were satisfied on versions B and C.

Regarding the models, for all scenarios it was considered a modified version of the DS-CNN. The inner blocks of the model were repeated 3 times (originaly it was done only 2 times), but the kernel size was changed from 3x3 to 2x2. It was added also a Dropout layer at the end of every block, since it was noticed a huge overfitting effect on the training step. As in the previous exercise, here it was introduced the parameter α to reduce the number of filters for each model.

In version-A, to reduce the size of the final file, it was used $\alpha = 0.25$, and a *Float16 Quantization* approach. No magnitude-based prunning was introduced in any of the models since the accuracy of the model was badly affected by its implementation.

On the contrary, for versions B and C, rather than using the *Float16 Quantization* it was used a *Weights + activations quantization* approach, declaring a generator function for providing the representative data.

All latency values were obtained after running the *kws_latency.py* file with the model versions b[4] and c[5] on the edge device.

[2]http://storage.googleapis.com/download.tensorflow.org/data/mini-_speech_commands.zip

[3]'sampling_rate': 16000, 'frame_length': 256, 'frame_step': 128, 'mfcc': True, 'lower_frequency': 20, 'upper_frequency': 4000, 'num_mel_bins': 40, 'num_coefficients': 10

[4]**cmd:** python kws_latency.py –model ./models/Group17_kws_b.tflite.zlib –rate 16000 –bins 16 –length 256 –stride 128 –mfcc

[5]**cmd:** python kws_latency.py –model ./models/Group17_kws_c.tflite.zlib –rate 16000 –bins 16 –length 256 –stride 128 –mfcc