Machine Learning for IoT
Homework 2 - Group 8

Rosi Gabriele	Fabio Tecco	Merlo Matteo
s291082	s285569	s287576

1 Multi-Step Temperature and Humidity Forecasting

The first optimization script aims at building, training and deploying an optimized multi-output model able to perform time series forecasting using the Jena Climate Dataset as requested. We proposed two models: the first one (A) takes 6 consecutive values of temperature and humidity, and outputs the next 3 values and the second one (B) takes 16 consecutive values of temperature and humidity, and outputs the next 9 values. The models used in both version are trained with the following structure:

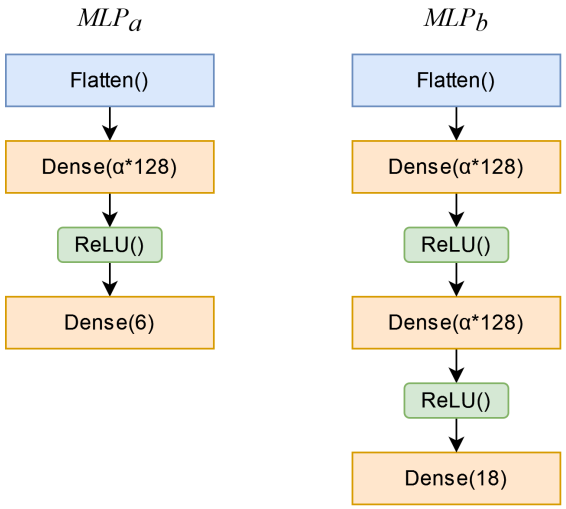

Figure 1: Models used for version A and B

Starting from the MLP model suggested in Lab03, the output has been reshaped in order to match the shape [None, $label\_width$, $num\_features$]. In addition, for version A a dense layer has been removed to meet size constrain. Furthermore, the training hyperparameters have been modified as showed in Table 1, in order to obtain an higher starting accuracy before the applied optimizations. Since the tflite model size without optimizations is around 70kb, a more than 35x reduction is needed. To do so in the code 3 types of optimizations are used: *Weight-Only Post-Training Quantization, Structured Pruning, Magnitude-Based Pruning. Magnitude-Based Pruning* with PolynomialDecay allow reducing a lot the compressed size of the model with a small impact on the loss. In the *Structured pruning* we use a α value that scale the network layer. This reduces a lot the model size but have some drawback on the loss. The *Weight-Only Post-Training Quantization* allow reducing the size of the model of a factor of x3/x4 with a not too high increasing of the loss. A post training quantization to float16 on the weights was applied to the model. We used in each pipeline the zlib compression.

Version	Model	Training Parameters	α	Final sparsity	Size (KB)	Mae T	Mae H
A	MLP_a	LR=0.1 (divided by 2 every 10 epoch) Batch size = 512, Epoch = 80 Input width = 6, Labels width = 3	0.2	0.93	1.477	0.29	1.18
B	MLP_b	LR=0.01 (divided by 2 every 10 epoch) Batch Size = 512, Epochs = 20 Input width = 16, Labels width = 9	0.1	0.90	1.78	0.55	2.33

Table 1: Hyperparameters and results for each version

2 Keyword Spotting

The aim of the script is to train models for keyword spotting on the original Mini Speech Command Dataset, keeping track not only of accuracy and size constraints but also latency. We used the DS-CNN suggested in Lab03 instead of CNN or MLP, since it is better in both accuracy and size. The three networks used are illustrated in Figure 2.

Starting from Lab03 model without any optimizations, we obtain a tflite file that is around 550kB. Clearly, a reduction is needed in order to satisfy size constraints.

As the same in first exercise a *Magnitude Based Pruning, Structured Pruning and Post-training Quantization* have been used in all the versions. The parameters (alpha, final sparsity, number of epoch, LR) have once again been found through a grid search. In version C in particular, the last dense layer of the DS-CNN is halved to meet the size constrain. In versions A the model is allowed to be bigger, so we therefore decided to apply *Magnitude Based Pruning* with a lower final pruning sparsity, since it affects a lot the model accuracy.

To meet the latency constrain in models B and C we reduced the frame length to 480. For all version we used MFCC mode to achieve the accuracy required. We notice that with SFTF the total latency decrease a lot due to a faster preprocessing, but in contrast the accuracy does not perform so well. The total latency then is measured on a Raspberry Pi 4 Model B with 4GB RAM (the command line used are below) using the following commands:

```
B) python3 kws_latency.py --mfcc --model Group8_kws_b.tflite --length 480
C) python3 kws_latency.py --mfcc --model Group8_kws_c.tflite --length 480
```

Version	Training Parameters	α	Final sparsity	Size (KB)	Accuracy	Mode	Total latency (ms)
A	Epoch = 30, LR = 0.01 (divided by 2 every 5 epochs)	0.6	0.6	108.03	93.15 %	MFCC	-
B	Epoch = 30, LR = 0.03 (divided by 6 every 10 epochs)	0.5	0.8	31.20	93.35 %	MFCC	37.5
C	Epoch = 30, LR = 0.02 (divided by 5 at epoch 20 and 25)	0.4	0.7	21.55	92.90 %	MFCC	38.3

Table 2: Hyperparameters and results for each version

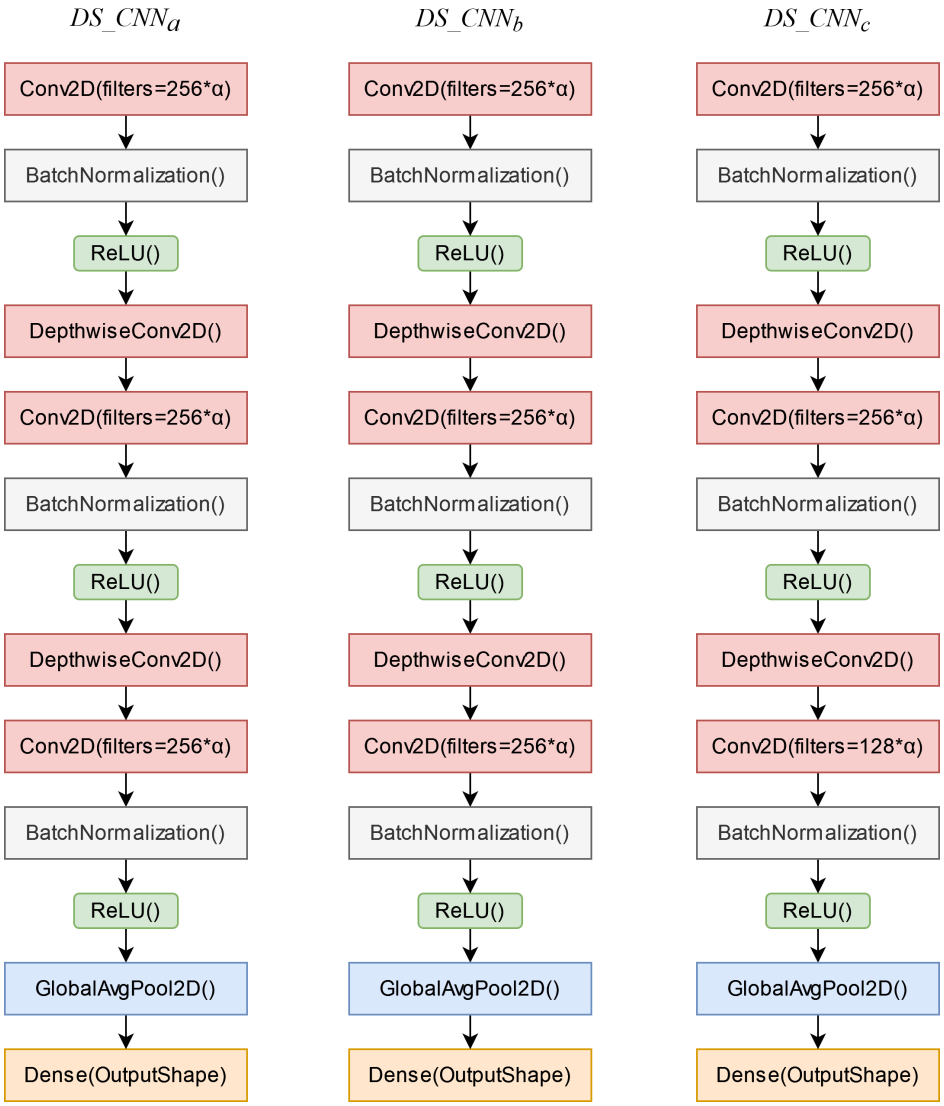

Figure 2: Network architecture for each version