Report Homework 3

Machine Learning for IOT

Authors:

Antonio Dimitris Defonte s276033
Eleonora Carletti s280056
Filippo Cortese s273672

Exercise 1

In this exercise the goal was to implement a client application that had to sequentially read an audio signal, run inference on it by its own little model, check the quality of the prediction and possibly call the big model.
To make the client communicate with the web service running the big model we needed to exploit a suitable communication protocol. When the success policy is not satisfied (not so frequently) the client sends the raw audio to the server and waits for a response, only when it receives it it continues to read and work on the audios in a synchronous fashion. So we decided to adopt REST because it is designed for client-server communication and respects all of these requirements.
Then, we trained both the “little” and “big” models and we decided to derive them from the DS-CNN network.
To increase the capacity of a standard DS-CNN we increased its depth by adding another DS-CNN basic block(DepthwiseConv2D, Conv2D, BatchNorm, Relu) and we also increased the width by multiplying the number of filters with an alpha of 1.5. This “Big” network is able to achieve an ***accuracy of 94.63%*** on the test-set. For the “little” model, in order to decrease the total inference time under ***40ms (tot inference time=34.56ms)***, we applied a resampling to 8 kHz and then preprocessed the audio with a mfcc preprocessing (with window ***length = 240***, ***stride = 120***). In order to respect the size constraint, we applied a more aggressive width pruning with a parameter ***alpha = 0.25*** and post training quantization, this allowed us to lower the size of the compressed model down to ***18.316 kB***.
The “success checker” policy was implemented with a score margin. We fed the output of the last layer of the “little” model into a softmax (in order to have an output between 0 and 1). If the difference between the most confident class and the second one was less than a given threshold, we would send an ***http PUT*** request to the big model. We decided to adopt the ***PUT*** method because it looks like the most natural choice in our situation consisting in always updating the input to the same web service with the same ***URI***. Since this type of approach is very data dependent we manually tuned the threshold to respect our constraints on the test set. We obtained the following results:

Threshold	Accuracy	Communication cost
0.30	93.00	3.31 MB
0.35	93.25	3.68 MB
0.40	93.50	4.13 MB
0.50	94.38	5.15 MB

We decided to keep the ***threshold*** to a value of ***0.4*** in order to have a higher margin on the accuracy and communication cost.

Exercise 2

In this exercise we had to create an architecture characterized by several components communicating one with another by means of a protocol of our choice.
To implement the communication part we decided to use the MQTT protocol because of its intrinsic properties of being event driven and asynchronous. In fact, the cooperative client is able to send the preprocessed audio files meanwhile the inference clients do both the prediction and asynchronously send the output layers back. After sending all the data, the cooperative client waits just the time needed to receive the last messages, instead of waiting for the response each time an audio has been sent.
The architecture contains a number N of instances of the “inference client” which has the aim of getting a preprocessed audio signal, classifying its content by means of different neural networks (one for each instance) and sending as output the last layer. Inspired by machine learning ensemble methods we started by implementing just two models that differentiate in the structure and also in the width. The reasoning behind this is to have models tackling different features of our data, then with a cooperative policy we exploited the information extracted by both models.
In the first instance we runned a model equal in structure to the “big” model of the previous exercise but with a width-pruning hyperparameter **alpha=0.5** (**acc=93.50**).

For the second instance we decided to run a CNN with a structure similar to that of the Lab03, but with 3 additional layers (a convolutional, a batch normalization and a relu). We used width scaling pruning with **alpha = 0.4** (**acc=93.88**). We applied standard preprocessing.
Another component of the architecture was the so-called “Cooperative client” that has two principal functions: preprocessing an audio signal and combining the outputs coming from the inference clients by means of a cooperative policy. The “cooperative client” receives the last layer (label weights) of each model, translates them into class probabilities through a *Softmax* and finally derives their ***score margin***. Then, it performs a weighted sum, with the ***score margin*** as the weight, of all the last layers received and it selects the class with the highest score. This weighted average is already used in traditional ML ensemble methods and allows us to fuse predictions of both models proportionally to their expected performance (***the score margin***). By performing this operation we were already capable of reaching an accuracy of **94.38%,** thus, we decided to keep a number of client instances equal to **N=2**.