Machine Learning for IoT
Homework 3

Di Nepi Marco S277959 | Benedetto Irene S276200 | Falletta Alberto Maria S277971

1 Big/Little Inference

The goal of this exercise is to build a web service in order to perform Keyword Spotting exploiting the architecture knob of Big/Little model.

1.1 The communication protocol

The solution is implemented through a REST protocol that allows the communication between the little client (on which the little model runs) and the big server, that makes inference using the big model only when is needed according to a success checker policy. The protocol chosen, REST, is a better alternative with respect to MQTT for the following reasons: we are in a server-client approach, therefore, a REST architectural style is preferred since it has fast performances, reliability and interoperability.
In our case, we don't need to send messages to multiple servers on the network, but only to a specified one, the server which hosts the big model. With this protocol we are sure to obtain always a response, or an error message in the case in which the server is not running. It is not possible to send a raw audio signal as a URL parameter, and therefore a POST method is used instead of a GET. In addition, the SenML + JSON format was used for both request and response.

1.2 The policy

A success checker was necessary in order to decide when to call the big server. An optimal policy has to be found being a trade-off between accuracy and communication cost function of the number of requests. In our case, for each sample the score margin is computed on the output of the little model:

$$SM = LargestProb - 2ndLargestProb \tag{1}$$

When SM is lower than an optimization parameter θ, meaning that the Little model is uncertain, the audio is therefore sent to the Big model for inference.
Given the communication cost constraint of 4.5 MB, since each request has a size of around 0.08 MB, the maximum number of request is 56. We found the best θ to be 0.35, but an additional $if - statement$ checks that the communication cost is not going to exceed the limit and will avoid the call.

1.3 Big/Little models and their parameters

The Little model is trained on audio signal STFT representations, while the Big model accepts the MFCC representations. In the Big model we selected an *alpha* equal to 1.3 and add more layers: this way, increasing the size of the architecture in terms of number of features and depth, we are able to achieve an higher accuracy. Instead, for the small model we meet the constraints in size and latency with these modifications:

- Width Scaling factor of 0.27;
- Magnitude-based pruning with an initial sparsity of 0.3 and a final sparsity of 0.58 (and consequently we performed a compression);
- A resize of STFT of $[64, 64]$ dimension: getting higher resolution so an increase of accuracy.

The final accuracy obtained is **93.5%** with a communication cost of **4.439 MB**. The tables below shows the models used and their parameters.

Version	Training	Alpha	Sparsity	Size	Mode	Total Latency	Accuracy
Small	Epoch = 35, LR = 0.02 (divided by 10 at epochs [20,25]) frame_step=128 frame_length=156 resize = [64,64]	0.27	0.58	29.8 kb	STFT	36ms	0.915
Big	Epoch = 32, LR = 0.01 (divided by 10 at epochs [20,25])	1.3	None	-	MFCC	-	0.945

Table 1: Hyperparameters for each version

Small	Big
Conv2D(filters=256*alpha)	Conv2D(filters=128 * alpha)
BatchNormalization()	BatchNormalization()
ReLU()	ReLU()
DepthwiseConv2D(),	DepthwiseConv2D()
Conv2D(filters=256*alpha)	Conv2D(filters=256 * alpha)
BatchNormalization()	BatchNormalization()
ReLU()	ReLU()
DepthwiseConv2D()	DepthwiseConv2D()
Conv2D(filters=alpha * 256)	Conv2D(filters=alpha * 512)
DepthwiseConv2D()	BatchNormalization()
Conv2D(filters=alpha * 128)	ReLU()
BatchNormalization()	DepthwiseConv2D()
ReLU()	Conv2D(filters=alpha * 512)
GlobalAvgPool2D()	BatchNormalization()
Dense(18)	ReLU()
Dense(output$_s hape$)	GlobalAvgPool2D()
	Dense(512)
	Dropout(0.1)
	Dense(256)
	Dropout(0.1)
	Dense(128)
	Dense(output$_s hape$)

Table 2: Models used for version A and B

Notice that in order to run the scripts, the correct IP address must be inserted in both little_client.py the big_server.py.

2 Cooperative Inference

In this exercise, we trained $N = 5$ models, differing in typologies and sizes. These models receive a preprocessed audio signal, perform inference, and send back the logits of the last layer.

2.1 The communication protocol

In this case we used the MQTT protocol since it is more efficient in sending the audio's MFCC only once with an unique topic rather than sending N different POST requests with the REST protocol. In particular, the inference clients subscribe to the topic:

`/Team10/277959/mfcc`

and when they receive an mfcc they send back the responses with topic:

`/Team10/277959/result`

after having performed the inference.

It has been proven that MQTT is able to reduce the power consumption with respect to REST protocol (it is a lightweight protocol designed for IOT systems), as in the latter the most of energy is lost on the resources used on connection and disconnection: in this case the number of connection would have increased with the number of models, that, on the other side represents the strength of the cooperative inference approach. With MQTT the need to connect and disconnect for every data transfer is not required.
Another advantage is that the models can work in parallel without the necessity or waiting each others. The cooperative client, instead, sequentially preprocesses the audio signals and publishes the MFCCs. In order to synchronize the communication, the client waits until all N responses are published by all the inference clients and applies the chosen cooperative policy.

2.2 The policy

We chose to apply a majority voting approach, in which the most frequent label is considered.
In case of ties among labels, the label corresponding to the model whose predictions vector has largest score margin is chosen: this way we guarantee that the prediction will come from the most certain model.
All audio files share the same preprocessing, that is the computation of mfcc with num_coefficients equal to 32.
With this strategy the accuracy obtained on the test set is **94.785%**

```
for l in logits:
    Convert l to softmax(l)
    Softmax stored in a list of predictions
    Compute the SM as in (1)

Select the most predicted labels l*

if len(l) == 1:
    Predict with l*
else:
    Select the highest score margin sm*
    Select the predictions associated to sm*, and among them extract the most probable label l*.
    Predict with l*
```

Pseudo-code of the policy adopted

	1	2	3	4	5
Accuracy	94.5%	94.3%	93%	91.75%	91.5%

Table 3: Accuracies of the models

1	2	3
Conv2D(filters=128 * 1.3)	Conv2D(filters=128)	MobileNet()
BatchNormalization()	BatchNormalization()	
ReLU()	ReLU()	
DepthwiseConv2D()	Conv2D(filters=256)	
Conv2D(filters=256 * 1.3)	BatchNormalization()	
BatchNormalization()	ReLU()	
ReLU()	Conv2D(filters=512)	
DepthwiseConv2D()	BatchNormalization()	
Conv2D(filters=1.3 * 512)	ReLU()	
BatchNormalization()	GlobalAveragePooling2D()	
ReLU()	Dense(256)	
DepthwiseConv2D()	Dropout(0.1)	
Conv2D(filters=1.3 * 512)	Dense(128)	
BatchNormalization()	Dense(output_shape)	
ReLU()		
GlobalAvgPool2D()		
Dense(512)		
Dropout(0.1)		
Dense(256)		
Dropout(0.1)		
Dense(128)		
Dense(output_shape)		

Table 4: Pool of N models used to perform the cooperative inference

4	5
Conv2D(filters=1.3*256)	Conv2D(filters=256*1.3)
BatchNormalization()	BatchNormalization()
ReLU()	ReLU()
DepthwiseConv2D()	DepthwiseConv2D()
Conv2D(filters=1.3 * 256)	Conv2D(filters=256 * 1.3)
BatchNormalization()	BatchNormalization()
ReLU()	ReLU()
DepthwiseConv2D()	DepthwiseConv2D()
Conv2D(filters=1.3 * 256)	Conv2D(filters=1.3 * 256)
DepthwiseConv2D()	DepthwiseConv2D()
BatchNormalization()	Conv2D(filters=1.3 * 128)
ReLU()	BatchNormalization()
GlobalAvgPool2D()	ReLU()
Dense(32)	GlobalAvgPool2D()
Dense(output_shape)	Dense(18),
	Dense(output_shape)

Table 5: Pool of N models used to perform the cooperative inference