Machine Learning for IoT
Homework 3 - Group 8

Rosi Gabriele s291082 | Fabio Tecco s285569 | Merlo Matteo s287576

1 Model Registry

The first exercise aims to build a model registry. To interact with the registry we have three main function: *add*, *list* and *predict*.

The ***add*** function has been implemented using REST and a PUT request since we are creating a new resource on the registry (the model that we want to store), but it is also possible to update an existing one.

The ***list*** function has been implemented using REST and a GET request, since we are requesting data to the registry, specifically the list of the model stored.

The ***predict*** function instead has been implemented mixing REST and MQTT. In particular: REST along with a GET request is used to start the prediction loop, and MQTT is used to send the asynchronous alerts. An alert is an event that it is unknown in advance when it will happen and for this reason, MQTT is the obvious choice due to its event-driven nature. Furthermore, there can be multiple clients that want to receive the alert and with a synchronous system (i.e. REST) all the clients need to connect to the server, and they need to wait for a new alert. Indeed, in this way, a client send the predict request through REST. The other clients just need to subscribe to the correct MQTT topic if they want to receive the alerts.
Here, a GET request is the most appropriate one, since no data has to be sent from client to service. The only data needed for inference are gathered from the DHT11 sensor connected to the Raspberry Pi (on which the service is running). The alerts are sent using the SenML+JSON format.

2 Edge-Cloud Collaborative Inference

The second exercise aims to build a Collaborative Inference framework for Keyword Spotting. Two pipelines have been created: a slow one running on the notebook and a fast one running on the Raspberry Pi. They differ only by the number of mel_bins and frame length. In order to meet the latency and accuracy constrain, through a grid search, the *frame_length* and the *mel_bins* found are respectively 480 and 32. So the final accuracy achieved is 91.25% and the total latency measured on Raspberry Pi is 39.5ms. The command used to measure the latency is the following:

```
python3 kws_latency.py --mfcc --model kws_dscnn_True.tflite --length 480 --bins 32
```

The communication between the two pipelines is implemented through a REST protocol. In particular, the fast pipeline send the audio file to the slow pipeline with a POST request according to a success checker policy. REST is a better alternative with respect to MQTT because we do not need to send messages to multiple servers on the network, but only to the server which hosts the slow pipeline. In addition, we need to receive a reply from the slow pipeline in order to implement the collaborative inference. A POST request is required since we are creating a new object on the server that runs the slow pipeline.

The success checker policy has been implemented by computing the Softmax of the model's prediction and then by subtracting the two largest probabilities in the prediction vector. When the resulted difference is under a given threshold, the fast pipeline is uncertain and therefore the audio must be sent to the slow pipeline. In order to meet the communication cost constraint, the threshold is again been found through a grid search. The best threshold found is 0.10 and the final communication's cost is 1.59 MB. Both the request and the response of the slow pipeline use the SenML+JSON format.