Machine Learning for IoT - Homework 03

Francesco Di Salvo s282418
Francesco Lacriola s292129
Gianluca La Malfa s290187

Exercise 1 - Model Registry

The goal of the exercise was to build a model registry that will store and manage ML models and will monitor the model performance on a specific task. As required, a **RESTful** API has been developed to manage the synchronous communication between a client and a server.

First, the client will send one tflite model per request through a PUT request at '**/add**' via a base64 encoding. This model will be decoded and stored on the *models* folder. The PUT request has been chosen because we assume that a model may be replaced if it will receive another model with the same name. Then, through a GET request to '**/list**' the client will request a list containing all the models present in the model registry. The GET request is necessary since it only requires retrieving information from the server.

Finally, the core of the architecture lies on the '**/predict**' request. The server will start recording new temperature and humidity measures through a DHT11 sensor and will make inference every 1s, after a window of six recordings has been built. If the **Mean Absolute Error** (MAE) between the prediction and the ground truth will be higher than a threshold specified by the client, the system will send an alert to external monitoring application(s). To do so, the connection between the previous client and the server will be still active in order to constantly make predictions. Since we are not suppose to send data to the registry client or to store data on the server, we used a POST request with model name and thresholds as parameters. Moreover, in order to send the alerts to one or other external monitoring clients, we exploited a **MQTT** connection. Therefore, anytime the server needs to send an alert, it will publish both actual and predicted values through a JSON body using the SenML format. Every second, at most one alert will be published containing two (four) events, for temperature or (and) humidity. Hence, the **monitoring** clients will just need to subscribe to the requested topic and the server will adjust the text and print it to the user in a human-readable way, when needed.

Exercise 2 - Edge-Cloud Collaborative Inference

The goal of the exercise was to build a computational paradigm where the inference workload is distributed across interconnected device to improve the prediction accuracy. The model provided for both device was a DS-CNN. Considering that the communication paradigm is synchronous we decided to develop a **RESTful application**. The **slow service** receives a base64 audio string in a SenML+JSON format, runs inference and returns the output label in JSON format under the **slow prediction** key. The **fast client** hosted on the raspberry iteratively reads audio files, runs inference and under certain conditions sends a PUT request to the **slow service**. In particular the **success checker** invoke the slow service only when the difference between the probabilities of the two most likely classes is less than 0.2 which made us able to reach a communication cost of 1.80MB. This threshold has been tuned in order to balance the model performance with the communication cost.

Finally, in order to reduce the latency, we preprocessed the audio in the following way: resampling phase at 8kHz with frame length = 20ms , frame step = 10ms and number of mel bins = 16. This configuration allowed our pipeline to reach a collaborative accuracy of 91.88% with a fast total latency of 38.30ms.