Machine Learning for IOT: Homework III, Group 16

1st Francesco Capobianco
Polytechnic of Turin
ID: *s281307*

2nd Pierluigi Compagnone
Polytechnic of Turin
ID: *s288301*

3rd Carmine De Cristofaro
Polytechnic of Turin
ID: *s291129*

I. Model Registry

registry_service.py

The application is made up by three classes: AddModel, ListModels and Predict. **AddModel** provides the manage of the path "add" and it is used to store a tflite model and its name on the server. We implement this functionality through the **POST** method because the entity enclosed in the request body is going to be a new subordinate of the web resource. **ListsModels** implements the path "list" and returns a list of the saved models. We choose the GET method because we just retrieve the needed information from the server without any other effects on web resources and moreover we are asked to not use request body. **Predict** functionality handles the path "predict". It uses one of the stored model, in particular the one specified in the url, to perform temperature and humidity forecasting based on a 6 values window (temperature & humidity) measured with the DHT-11 sensor every 1s. It is implemented by means of the GET method because, also in this case, the stored models -our web resources- are not modified by the application but are only interpreted for the inference pipeline. Moreover, this application an alert to the **monitoring_client** every time the MAE between the prediction and the actual value of temperature or humidity is above the specified thresholds. To send this alert we use the MQTT protocol because in this way we are not forced to stop the execution of the application, thus we do not need to re-build the 6 values window each time an alert occurs.

registry_client.py

Registry client is the application that handle the request for the web service. First of all, it sends two add requests, one for each model to upload on the server. The model and its name are packed in a json file, which is the request body, in order to be correctly sent to the service. Subsequently, the client invoke a list request to obtain the list of the model names and eventually it sends a predict request in order to start the forecasting framework. The model used for the prediction and the needed thresholds are sent as query in the predict url.

monitoring_client.py

The monitorin_client is a MQTT subscriber, signed up to the topic on which the predict class publishes the alerts, that shows these alerts on screen.

II. Edge-Cloud Collaborative Inference

We use **MQTT** protocol, instead of REST, for three main reasons: MQTT is maximized on power-constrained and small devices; its messages are not heavily affected by limited connectivity; it ensures higher reliability thanks to the quality of service. In addition MQTT protocol is way better than REST in One-to-Many communication that could be useful to perform audio inference on parallel devices.
Both applications work as publisher and subscriber, indeed, the client is publisher when sends audios deemed not confident and subscriber when it collects the inferences from the service, while the service is subscriber in getting not confident audios and publisher in sending back its own predictions.

fast_client.py

It reads the input wave files to be classified, then it processes them and performs a multi-labels inference in order to match the latency constraint.[1] Moreover, it checks up the not confident predictions according the success policy checker and it sends the ones refused by the policy to the slow_service application packed in SenML+JSON strings format. After that, the application receives the slow_service predictions and finally it computes the collaborative accuracy by means of its own inference and the ones it obtains from the server.

slow_service.py

It receives the audio files which the client is not confident of, it computes a different preprocessing pipelines which is slower but ensures a higher quality representation of the audio signal. Afterwards, it carries out its own predictions with model used before and it sends back the results to the client application.

success checker

It checks whether the classification of the fast client can be considered confident or not according to the top 1 policy. This policy takes as input the softmax output vector and it validates the top value with respect to a defined threshold equal to 0,45. If the top 1 value is greater than 0,45 the client prediction is considered reliable, otherwise the audio files is sent to the service for a further inference.

acc.: 91,625%, com_cost: 1.347 MB, fast_inf.: 38.91 ms

[1] the preprocessing parameters for both slow and fast inference are the same used and reported in homework II.