Homework 3

GROUP 17

Juan Aragon (s291466) Hadi Nejabat (s246601) Rostislav Timuta (s280238)

I. Model Registry

For this exercise it was required to implement a web application to store and manage pre-trained tensorflow models for predicting temperature and humidity data. There were three different services made accessible through a RESTful architecture running on a Raspberry Pi as declared on the *registry_service.py* file:

- Add Service [*/add*]: Stores on the local directory *./models/* the pre-trained tensorflow models provided from the client. The client sends a PUT HTTP request with a body in json format, containing the name of the model and the binary file encoded in base64. It was decided to use PUT instead of POST in order to give the possibility to the client of updating an already existing model, sending on the payload the name of the file to modify and the new binary file.

- List Service [*/list*]: Returns a json body containing the list of all the models' names available on the local directory *./models/*. The client receives this response by sending to the server a GET HTTP request with no additional parameters on the query.

- Predict Service [*/predict*]: The client sends a GET HTTP request specifying the name an existing model and two threshold values as parameters, for temperature and humidity respectively. On the other side, the web service validates that the name received corresponds to an existing model and sets up the referred model using the interpreter of tf-lite. Then, by using a *DHT-11* sensor connected to the Raspberry pi, it collects 6 weather samples and uses the given model to infer the 6th element from the first 5 measurements. Finally, it evaluates the error of the prediction for the temperature and humidity. In case it surpasses the given thresholds it sends an alert using the MQTT message protocol, publishing the relative information (in SenML format) in the topic: */Group17/alert*.

Additionally to the registry-server and registry-client already mentioned, it was required to implement a monitoring-client, subscribed to the topic */Group17/alert* using the MQTT message protocol. Its job is to receive the alerts sent by the registry-server and to display the relative information extracted from the SenML payload.

II. Edge-Cloud Collaborative Inference

The goal of this exercise was to implement a collaborative inference framework for the keyword spotting task we saw in the previous homeworks. In this case, implementing a fast inference preprocessing pipeline on the edge device, that sequentially reads the audio files from the mini-speech dataset and obtains their mfccs values which are then given to the pre-trained model already available for predicting the proper label of each audio.

In order to minimize the energy consumption on the edge device, without affecting noticeably the accuracy of the predicted labels, it was decided to reduce the sampling rate and the number of mel_bins used to generate the mfccs images. The motivation behind this choice is provided on the report of the homework 1, where it was seen that by using a sampling rate of 8KHz and 16 mel_bins[1] the quality was well preserved while the execution time decreased noticeably.

The exercise required to improve even more the accuracy of the model by using the collaborative inference framework. For this goal, a success_checker() function was implemented, analysing the difference between the two highest values from the model's output, and checking the confidence of the model for the predicted output. The main function of the success_checker() is to decide if it is necessary to send the audio signal to the cloud service for a deeper analysis in order to get a more accurate label prediction. It was noticed that by using a threshold of 0.25, the overall accuracy was fine for the purpose of the homework.

When the success_checker determines that it is necessary to send the audio to the cloud service, the original audio signal is encoded into a payload using the SenML+json format, and delivered to the web server with a POST HTTP request, waiting for the proper answer containing the label of the audio that was just sent.

Once the cloud server receives the raw signal of the audio file, it preprocess the data using a slow inference pipeline, where the sampling rate and mel_bins parameters[2] are kept on their maximal values, 16KHz and 40 respectively.

The output obtained shows that the accuracy of the system is 91% with a total communication cost of 0.80 MB. Finally, regarding the fast inference pipeline it was obtained an average execution time of 36.75 ms.

[1] fast inference pipeline: –rate 8000 –bins 16 –length 320 –stride 160
[2] slow inference pipeline: –rate 16000 –bins 40 –length 640 –stride 320