Homework 3 Report

Group 24

1 Exercise 1

We developed the web-server (*registry_service.py*) by using the RESTful api offered by HTTP protocol. For saving the CNN and MLP models in the server, we used the **PUT** method, that allows to write the payload on the client side.
As for the second request, we choose to implement it using the **GET** method since no payload was sent from the client and we only needed a returned payload from the server.
Finally, for the 3rd request we used the **POST** method where data are collected in the web-server and fed to the network specified in the body sent from the client (in our case always the CNN) for the inference. To send the alert we choose to exploit the MQTT protocol, suitable for event-driven communications, that in our case is triggered by higher difference between *y_prediced - y_measured* and a threshold value. The MQTT-publisher is instantiated and it runs in the **POST** class and it publishes the alerts to the *monitoring_client.py* based on two different constraints on the absolute error between the prediction and the actual measure: one for temperature and one for humidity.
Since the CNN model developed in lab3 was trained on normalized data, the predictions obtained were very far from the actual values, so we trained again the CNN without normalization and used the new model to make inference, obtaining more suitable predictions for this task.

2 Exercise 2

First a web-server service (*slow_service.py*) and a client script (*fast_client.py*) were developed through the RESTful protocol using the POST method.
We used this api since in our setting we had a client that only connects to one server only when some constraints are not fulfilled. Then the client sends a request and awaits for a response from the server.
For the pre-processing, first the audio's spectrogram is computed through the STFT, then MFCCs were extracted. For the slow pipeline we adopted the following parameters: frame length of 40ms (640 samples per window); frame step of 20ms (320 samples per window); the bound for the frequency included in the mel spectrum goes from 20 Hz to 4 kHz; 40 filters applied on the frequency bound; first 10 MFCCs were retrieved.
For the fast pre-processing we changed the following parameters:

1. frame length of 29.4ms (470 samples per window)
2. 32 filters applied on the frequency bound

The "success checker" policy was designed to check a threshold value corresponding to the difference between the first most likely label and the second most likely label computed by the client.
First the client makes an inference, then the *success_checker* function extract the prediction's Softmax which are sorted from the less likely label to the most likely one. Then we take the last two elements of the previously sorted vector and compute the difference assuring at least a 20% difference in the two probabilities ($threshold = 0.2$). If the constraint is not respected, the client send a request to the server (IP:virtual machine used) to make a new inference with the slow pre-processing. The final configuration reached an accuracy of 91.12% on the test-set with a communication cost of 1.53 MB and a latency in the range 39.18-39.70 ms (tested through several runs using the python script *kws_latency.py* of Homework 2 with parameters –mfcc –length 470 –bins 32).